**Elmar Stellnberger:**

## Finding all Solutions to a CNF:

# The DualSat SAT Solver

unentangled literal detection
conflict dependent and independent heuristic
backtracking with stack redo
dual data structures

---

MASTER THESIS

*submitted in fulfilment of the requirements for the degree of Diplom-Ingenieur*
*Programme: Master's programme Applied Informatics*

Evaluator: Univ. Prof. Dr. Martin Gebser

Alpen-Adria-Universität Klagenfurt, Institut für Angewandte Informatik

Klagenfurt, October 2020

In case of comments or problems, please contact the Department of Applied Informatics at the University of Klagenfurt, Austria (ainf@aau.at) or the author himself (estellnb@elstel.org).

# Affidavit

I hereby declare in lieu of an oath that

- the submitted academic paper is entirely my own work and that no auxiliary materials have been used other than those indicated;
- I have fully disclosed all assistance received from third parties during the process of writing the paper, including any significant advice from supervisors;
- any contents taken from the works of third parties or my own works that have been included either literally or in spirit have been appropriately marked and the respective source of the information has been clearly identified with precise bibliographical references (e.g. in footnotes);
- to date, I have not submitted this paper to an examining authority either in Austria or abroad and that
- the digital version of the paper submitted for the purpose of plagiarism assessment is fully consistent with the printed version.

I am aware that a declaration contrary to the facts will have legal consequences.

Elmar Stellnberger m.p. 10.10.2020, Klagenfurt am Wörthersee

# Acknowledgements

Many thanks to Univ. Prof. Dr. Martin Gebser who has guided and supported me at my work and who has given me valuable feedback.

# Table of Contents

# 1 Introduction

It was within a single decade that the practice of implementing programs for testing satisfiability of propositional formulas has made significant progress. While the first SAT-solvers were a practical curiosity, nowadays many real world and industrial scale problems can be solved with SAT-solvers. These programs are used in verification of hardware as well as software, in automatic test generation, in logical synthesis, in the analyzer of the Alloy software modeling language and in Artificial Intelligence for planning and automatic theorem proving [GuMi05], [MMZZ01]. Common SAT-solvers focus on finding just one possible solution to a propositional formula or by giving a refutation proof if such a solution should not exist. It may be sufficient to find that a given instance is unsatisfiable in order to prove the absence of bugs in a software or hardware design. There exists however a related problem, called #SAT (pronounce: 'sharpSAT') which goes for propositional model counting. A solution of a logical formula is called a model and thus the name model counting. It turned out that model counting is a particularly hard problem even for some polynomial-time solvable problems like 2-SAT which restricts a given conjunctive normal form (CNF) to clauses of length two. Model counting is used for bounded-length adversarial and contingency planning, probabilistic reasoning and Bayesian inference [BHMW08], [SBB04]. For some hard combinatorial problems the number of solutions provides further insights into the problem. While finding one solution can already be a challenge counting the number of solutions is even harder.

The problem of SAT-solving is NP-hard. That is you can test in polynomial time $O(n^x)$ where x is constant whether a given solution (here: to a logical expression) evaluates to true (here: in a fact in O(n) steps) but you need non-polynomial time to find such a solution. Given that you have n variables testing for all possible solutions has a worst case complexity of $O(2^n)$ for SAT or generally $O(b^n)$. This is in deed a very hard problem complexity class since you need twice the time to solve a problem with just one more variable. So if you have a computer that works eight times faster, you can only solve a problem with three more variables. Where commonly available instances of satisfiability problems have up to 10,000 variables (rutgers.edu/ssa6288-047.cnf [Ru00]), industrial scale problems being even larger, it would not be possible to get a solution here by simple truth value testing. $2^{10000}$ is a number with 3011 digits. Even so current SAT-solvers apply a number of techniques to make satisfiability testing of real world problems more tractable, the basic problem remains that any decision on a variable, where both branches need to be explored exhaustively, does effectively double the execution time. That is why there are always a number of solvable problems but still also a number of unsolvable problems for SAT. One problem known to be hard is the verification of binary integer multiplication, because every digit is multiplied with each other digit before the sub-results can be added together. While integer adding involves just O(n) steps, multiplication requires $O(n^2)$ different operations which will all need to be verified [By86]. While the rutgers.edu/par16* instances are still solvable, instances like rutgers.edu/par32* have turned out to be unsolvable by the SAT-solvers we could test in this work.

There has been considerable work on the formal analysis of NP-hard problems even before the first practically usable SAT-solvers have emerged [Ng13], [So08]. There are a number of NP-hard problems like the vertex cover problem, the n-clique problem, the independent set problem, the hamilton cycle problem, the k-coloring problem, the traveling salesman problem, the knapsack problem and the bin packing problem. When the fact that every problem in NP can be reduced to SAT in polynomial time was widely regarded to be of theoretical interest before, the emergence of efficient SAT-solvers makes it now possible to solve many of such problem instances as well. The programs which convert a given problem into SAT are usually called *encoders*. Sometimes also preprocessors are used to make the encoding more efficient and to reduce the complexity in writing such domain specific encoders [EeBi05]. Some problems however cannot be solved by simple SAT-solvers alone. That encompasses verification and proofs with recursive structures that are not

bound to a depth limit [DvPt60] or problems like answer set enumeration which require an extension of plain SAT solvers [GKAS07].

## *1.1 The Idea about DualSat*

The initial idea about DualSat was to write a solver that returns not just one but all solutions to a given problem. This is different from plain SAT solvers which only return the first solution they can find and actually different from merely counting solutions. This may be important if a problem cannot be expressed solely by SAT as a CNF but if you have a coherent optimization problem on the number of satisfiable instances. That is why DualSat aims to return solution classes that are as compact as possible. One solution class encompasses many solutions on which a postprocessor may f.i. execute a non-linear optimization problem later on. There are plenty of NP-hard problems in computer science and some are linked to not just return any solution but to find the best possible solution of a host of solutions.

A simple solution class defines a zero or one value for a number of variables and leaves some other variables unassigned. You may write such a solution class as 01xx1 which means $x_1$=0, $x_2$=1, $x_5$=1. In deed the very first solvers like CDP did already return such simple solution classes [BHMW08]. However the old style literal counting scheme of this implementation has some time been replaced with lazy data structures and watched literals. These require that all variables become assigned before a solution is detected [LyMS05].

Now look at a solution class with t assigned variables. It amounts for $2^{n-t}$ solutions when you have n variables. In a worst case the basic SAT instance of our optimization problem can be a tautology and this would require then $2^{n-t} = 2^n$ steps to enumerate all solutions. That may last longer than the age of the universe given that you have enough variables and that you enumerate solutions one by one.

The idea about DualSat was now to combine old style literal counting for detecting that all clauses are already satisfied and that the remaining variables can take arbitrary values with the new lazy data structures which are only used for learned clauses. Learned clauses help in pruning the search space of parts that are already known not to contain a solution. They just help to get out of these parts of the search space more quickly. However learned clauses are additional sentences that can be omitted when merely testing for the satisfiability of a given solution. That is why DualSat employs old style data structures for the core problem given as input and new lazy data structures for learned clauses. It has shown that the runtime penalty for this is manageable as most clauses in a modern SAT-solver are in deed learned clauses.

Even better the benchmark results of DualSat have shown that it is still competitive with solvers like ZChaff and Clasp when only finding one solution, a case DualSat has not prevalently been optimized for. ZChaff solved one more instances and Clasp two more instances of the benchmark set from rutgers.edu than DualSat in 60 seconds (16 unsolvale, 242 instances). For counting solutions DualSat was considerably better than Clasp. For problems like the optimization problem mentioned before DualSat may be the only option since Clasp returns results one-by-one, ZChaff focuses on a singleton solution and sharpSAT only returns a solution count.

As far about simple solution classes. With extended solution classes DualSat terminates for a given solution class not just as soon as all clauses are satisfied but even before when a given solution class is in so called DNNF (Decomposable Negation Normal Form) [DwDN01]. That is the case if the variables of all remaining clauses are unentangled. It means that each variable shows up in at most one remaining unsatisfied clause. The corresponding solution class can then be written like 11x(-2)(-2)(+2)x(+3)(-3). A solution class like 10(-2)(2)(-2) can be decomposed into 100xx, 10x1x and 10xx0. We show on how to count such a class in chapter 2.3.

Besides this DualSat realizes a couple of other new ideas which may prove beneficial for

future SAT-solvers. At first it does not merely rely on a conflict based heuristic like VSIDS [BFS15]. When there are little conflicts like in randomly generated SAT problems or the instance collection of Cachet, DualSat employs a refined version of BOHM's and the Jeroslow-Wang heuristic [MqSv99]. That way DualSat can solve about 40% of the model count problems from Cachet DQMR while Clasp could not solve any of them.

Another milestone of DualSat is its stack redoing feature. Current SAT-solvers employ non-chronological backtracking. That is if a conflict is encountered and a learned conflict clause is generated then the solver jumps back up to the assertion level where the new learned conflict clause becomes unit and can en-queue the sole unassigned non-false literal as derived fact. However by doing so the solver loses already made assignments and needs to begin starting from the assertion level from scratch. There is already early work which has been aware of this problem [PiDw07]. What DualSat does is to first go back to the assertion level to add the assertion literal but then to push all decision literals above again on the stack until the currently pushed decision literal leads to a conflict. When doing so it can also keep all inferenced literals that previously have been on that level. It only needs to add new literals and detect conflicts. We will call this process stack redo. Employing a stack redo DualSat can almost restart from where it had to break off due to a conflict.

The nogood handling of DualSat is also different from that of previous #SAT-solvers. Instead of recording nogoods for already visited solutions it creates nogoods on backtracking. Evidence has shown that shorter backtrack-nogoods do even speed up the solution process as they can prevent the solver from re-entering already explored parts of the search space. There is also a mechanism to get rid of nogoods as soon as possible. That is on variable complementation which appear due to search progress. Nogood deletion is also executed after a stack redo taking place directly after nogood creation when the nogood proves to have become implicitly true by all forthcoming literal assignments.

DualSat also performs restarts before the first solution has been reached. Restarting from scratch has proven to be beneficial as otherwise the solver can get stuck in a part of the search space which is hard to explore but does not contain any solution. Most modern solvers employ restarting and for #SAT this technique is especially beneficial on unsatisfiable instances. That way the solver learns as much as possible about the whole search space before it starts to enumerate solutions. Finally DualSat randomizes the initial assignment value between zero and one for problem instances where always trying zero first may be detrimental. It does so by a literal activity counter. BerkMin claimed this method to be slightly better than random initialization. Clause database reduction has been implemented very much the way as it is in BerkMin and that has proven to go well in practice [GoNk07]. If the solver would not forget old learned clauses that mostly contain information about already visited parts of the search space, the solver would work considerably slower.

Overall DualSat is a state of the art solver ready for use, which can unlock new problem types for SAT-solving. There are still a whole lot of things that could be improved about DualSat and continuing its development is likely to prove very worthwhile. However there are problems which DualSat cannot solve. Problems consisting of multiple contingency components are an example. Nonetheless such components can be detected very easily by adding all connected variables to a bitset, something which is f.i. doable by a preprocessor. Pipatsrisawat has tested for this use case by simply replicating the same problem four times. Even a common one-solution solver had been orders of magnitude slower on the compound problem [PiDw07]. This is due to the same problem having to be solved over and over again for each assignment the solver tries with the first contingency component of the compound problem. Current #SAT solvers like first Relsat know that checking for contingency components can pay off also after every step when it comes to the harder task of model counting. Each smaller individual subproblem only needs to be solved once. Additionally, as soon as a component is discovered unsatisfiable the whole problem is. Elaborate solvers specifically optimized for #SAT like Cachet or sharpSAT do also employ a technique called

component caching as the same component may be rediscovered once the solution graph falls apart by assigning new variables. In that graph variables in the same disjunctive clause need to be connected with edges while variables become vertices [BHMW08] .

# 2 SAT-Solver Technology

## *2.1 Preliminaries*

### 2.1.1 Encoding a Problem as SAT

A *propositional formula* is an expression made up of logical and, logical or and the unary not as well as propositional variables that can take the truth value one and zero or true and false respectively. The constants true and false can be evaluated against the and- and the or-operator due to the neutral element (0 for ∨, 1 for ∧) and the dominant element (0 for ∧, 1 for ∨). Such a constant should thus only be necessary in an expression in case of a tautology (always true) or a contradiction (always false). If we want to normalize an expression we can push the not-operator ¬ from bracketed expressions up to the leaves posed by variables or constants. The logical axiomon that implements this is called De Morgan ( $\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$, $\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$ ). It can be easily remembered as a rule to replace a repeat-until loop by a while loop: Fetch water until *a* and *b* are true or as long as *a* or *b* are not satisfied. The next important axiomon of a boolean algebra which we can apply is distributivity. Or is distributive against and. And is distributive against or. For multiplication and addition it is only the multiplication that is distributive against addition. So we have $a \vee (b \wedge c) \Leftrightarrow (a \vee b) \wedge (a \vee c)$ and $a \wedge (b \vee c) \Leftrightarrow (a \wedge b) \vee (a \wedge c)$. If we now apply either of these rules exhaustively we get a propositional formula that is in Conjunctive Normal Form (CNF) or in a Disjunctive Normal Form (DNF). As can be seen easily a problem in disjunctive normal form simply enumerates all solutions with or. That is if we succeed to bring a logical formula into DNF we have already won. However consider the cost of applying and-distributivity exhaustively. For the first two-variable-bracket the number of subexpression doubles and so on for any subsequent bracket resolution.

Unfortunately a problem is typically given in CNF. We have a set of expressions which need to be fulfilled (i.e. be true) all together. Let us look on how to encode an implication like $a \wedge b \rightarrow c$ into a CNF. First 'ex falso quodlibet' so either $a \wedge b$ can be false ($\neg a \vee \neg b$) or otherwise our conclusion needs to be true to maintain a true implication. That is $\neg a \vee \neg b \vee c$. If we have multiple clauses or axiomons we connect them with '∧'. Consider the CNF $(\neg a \vee \neg b \vee c) \wedge (\neg c \vee a)$. It can be written as a set of sets, that is { {¬a,¬b,c}, {¬c,a} }. This is the way modern SAT solvers store their input CNF. The symbols a, b, c are thereby replaced by integer numbers, i.e. { {-1,-2,+3}, {-3,+1} }. There have only been a very few solvers which do not accept their propositional input system as CNF. That were HeerHugo and Stålmark's algorithm which do also operate on the logical equivalence operator and on subexpressions. The whole CNF denoted by a set of sets is a *propositional sentence* while each individual subset joined by logical disjunction is called a *clause*. Each member of one of these subsets is called a *literal*. A literal can be a *variable* or its negation. The whole expression is called *satisfiable* if it has at least one solution. Otherwise it is unsatisfiable.

The whole problem of solving a CNF is logically isomorph to finding a set of literals where no clause is a subset ($\subseteq$) of these literals and no superset of this set can have a clause as its subset. For a full assignment it boils down to simple subset checking. A clause $a \vee b \vee c$ can also be written as $\neg(\neg a \wedge \neg b \wedge \neg c)$ and that is where the name nogood stems from. We use it here as a synonym for clause, although the subset test must be taken against the negation of the literals of a clause-set. In DualSat the term nogood is used for learned clauses that must not be forgotten in order to prevent non-disjoint or repeated solutions, which is necessary for solution counting. Two sets are disjoint if $a \cap b = \emptyset$.

Another common property is subsumption that is a clause is subsumed by a subset-clause and can then be left out, i.e. $a \vee b \rightarrow a \vee b \vee c$. The input CNF can and especially for DualSat should be an axiomon set without derived sentences that can be implied by the axiomons. We have already

mentioned in the introduction that operations for input clauses are more expensive to DualSat than operations on learned clauses. Before DualSat was developed, subsumption checking has already been one of many operations that can be performed by a preprocessor [EeBi05].

### 2.1.2 Resolution, Preprocessing

A common operation applied to clauses is resolution: $F \Leftrightarrow (a \vee b \vee c) \wedge (\neg a \vee d \vee e) \Rightarrow b \vee c \vee d \vee e$. Intuitively variable a can be true or false. If it is true then $d \vee e$ need to be valid, otherwise $b \vee c$ needs to be valid. Formally *tautology* $\Leftrightarrow (a \vee \neg a) \Rightarrow_{(F)} F|a \vee F|\neg a \Leftrightarrow (d \vee e) \vee (b \vee c)$ whereby $F \wedge p \Rightarrow F|p$. F|p is pronounced *F under p* and means F where all occurrences of p are replaced by true and all occurrences of ¬p are replaced by false. $(b \vee c \vee d \vee e)$ is called the *resolvant*. Learned clauses are obtained from the clause that caused a conflict (contradiction with the actual assignment) by resolving it against the 'cause' of other contained literals. Such a cause is another clause with one of its literals negated. We will clarify later on how it works.

Clause learning is not the only application of resolution. The *Davis Putnam* algorithm is an algorithm that fully relies on resolution [BHMW08]. It eliminates each variable one by one. If more than two clauses with *a* and ¬*a* appear, then each clause containing *a* needs to be joined with each clause containing ¬*a,* while the original clauses that contained an *a*-literal can be forgotten.

This step is *refutation complete*. It means if $F \Leftrightarrow (A \vee p) \wedge (B \vee \neg p) \wedge R$ has at least one solution then also $(A \vee B) \wedge R$ needs to have at least one solution. If F is *inconsistent* then it needs to be false whether p amounts to false or true. In the first case F reduces $A \wedge R$, and in the second case to $B \wedge R$. So F is inconsistent if and only if $A \wedge R$ as well as $B \wedge R$ are both inconsistent. That is exactly the case if $(A \wedge R) \vee (B \wedge R)$ is inconsistent. The or-operator only yields false when both of its operands yield false. Now we know that $(A \wedge R) \vee (B \wedge R) \Leftrightarrow (A \vee B) \wedge R$ is inconsistent [DvPt60]. A and B can both be a conjunction of subclauses. If they are resolved distributively against each other then this does exactly what the Davis Putnam algorithm does: Combine each clause with an *a* with each other clause that contains a ¬*a*.

The final solution is obtained by going back each step in reverse order. Up to then all variables except *p* are already known. By substituting the known variables you get a logical expression solely containing *p*. Simplify it. Then set *p* true or false respectively. The final expression can also resolve to a tautology not containing *p*. Then *p* may take any value.

It can be shown that the Davis Putnam algorithm does not just find any solution but it can be used to find all solutions. $(A \vee p) \wedge (B \vee \neg p) \Leftrightarrow (A \wedge B) \vee (A \wedge \neg p) \vee (B \wedge p)$. A solution for $A \wedge B$ logically also holds in $A \vee B$, i.e. $A \wedge B \rightarrow A \vee B$ (The latter could have even more.). A solution for $(A \wedge \neg p) \vee (B \wedge p)$ logically also holds in $(A \vee B)$. So no solution is lost by the transformation. We have shown before that the other direction, that each solution for $(A \vee B) \wedge R$ is also a solution for F, works in deed as well through back-substitution.

Today the Davis Putnam algorithm is no more applied in practice. It has high space requirements since an expression containing all solutions needs to be kept at once in memory till the end. The effectivity of Davis Putnam very much depends on the variable selection order. However there is only one order for the whole algorithm and not one independent order for each explored branch like in the Davis Putnam Loveland Longman (DPLL) algorithm which has nowadays become the dominant way for solving SAT. Obviously this is less flexible.

Nonetheless the method of the Davis Putnam algorithm is still widely used in preprocessing. That is before the DPLL algorithm is applied. The NiVER preprocessor only eliminates a variable when the number of literals in the resulting clause is reduced by the elimination [SuPr04]. It was reported to lead to a 74% decrease in the number of variables, a 58% decrease in the number of clauses and a 46% decrease in the literal count. More elaborate preprocessors can detect definitions like the definition of an and-gate $x \leftrightarrow a \wedge b$. This means that every occurrence of ¬*x* can be replaced

by ¬a∨¬b  (a∧b→x, *reverse implication*) and that every occurrence of *x* needs to be duplicated, once for *a* and once for *b (x→a∧b)* . This amounts to only compute resolvents with the member clauses of the gate definition. All other resolvents are logically implied by these primary resolvents [EeBi05]. We have already mentioned subsumption checking in chapter 2.1 which is another step taken by preprocessors. Sometimes one part of a gate definition needs to be completed f.i. ¬a∨¬b by ¬a∨¬b∨x to make the preprocessor detect the definition.

Preprocessing is already known to be of advantage for one-solution SAT-solvers. The benefit for DualSat would even be higher because of its dual data structures which require a well reduced set of core clauses. Unfortunately we did not have any preprocessor at hand to test DualSat with. We would expect considerable runtime gains since watched literals are known to be eight times faster than the literal counters still used for the core clauses [GuMi05].

## *2.2 DPLL Solvers*

We have already mentioned that today's solvers are widely based on the Davis Putnam Loveland Longman (DPLL) algorithm. It tries the other value for a variable if it did not already have success with the initially assigned value. That is like a depth first search so that there is no need to keep expressions for all possible solutions in memory. In its simplest form the DPLL algorithm employs chronological backtracking. Most modern solvers use conflict directed backjumping and clause learning. We will also discuss the strategy of DualSat. Stack redos can be seen as a new development based on conflict directed backjumping, overcoming the issue of lost assignments. We will also see how the basic DPLL algorithm can be modified in order to return all possible solutions instead of just one.

### 2.2.1 DPLL with Chronological Backtracking

We have already said that making a choice on a variable comes with a worst case performance penalty of doubling the execution time. One way to avoid unnecessary choices is *unit propagation* or *Boolean Constraint Propagation* (BCP). If all but one literals of a clause are false and the remaining literal is yet unassigned then make this literal true. If it would take the value false then the whole CNF would become unsatisfiable so we need to avoid this. The new literal assignment may now cause other clauses to become unsatisfiable so we need to avoid this. The new literal assignment may now cause other clauses to become unit and propagate their sole unassigned literal as true. This is exactly the case if the newly assigned literal appears in another clause in negated form.

```
proc DPLL( ϕ :CNF, α :assignment ):
  α = UnitPropagation( ϕ, α )
  if α is conflicting / there is an empty clause in ϕ: return unsatisfiable
  if all clauses in ϕ are satisfied: return α
  if there is a pure literal p: return DPLL( ϕ, αp )
  select p ∉ α, ¬p ∉ α
  result = DPLL( ϕ, αp )
  if result is a satisfying assignment: return result
  else: return DPLL( ϕ, α¬p )
```

Fig 1: recursive DPLL with chronological backtracking

Fig. 1 shows a recursive formulation of DPLL with chronological backtracking. The pure literal rule is only allowed if we want to find one solution. It says that if a literal does not appear in negated form in any clause, then we are allowed to make it true. However this does not mean that there would not also exist solutions with ¬p. Most current solvers do not implement the pure literal rule since checking for pure literals would be too costly.

```
proc DPLL( Φ :CNF, α :assignment ):
  α = UnitPropagation( Φ, α )
  if α is conflicting / there is an empty clause in Φ: return
  if all clauses in Φ are satisfied: yield α; return
  select p ∉ α, ¬p ∉ α
  DPLL( Φ, αp )
  DPLL( Φ, α¬p )
```

Fig 2: recursive DPLL returning all solutions with chronological backtracking

Chronological backtracking is only the most simple form to implement DPLL. In order to refine our algorithm further we need to formulate it iteratively.

```
proc DPLL( Φ :CNF, α :assignment ):
  initialize α with level 0 and push all atomic facts onto α
propagate:
  α = UnitPropagation( Φ, α )
  if α is conflicting / there is an empty clause in Φ: goto next_assignment
  if all clauses in Φ are satisfied:
    yield α
    goto next_assignment
  goto extend_assignment
next_assignment:
  while level(α)≥0
    remove all assignments from α back to the last decision literal
    if the decision literal has not been complemented yet:
      complement the decision literal
      break;
    remove the decision literal and go one level up in α
  if level(α)≥0: goto propagate
  else: return
extend_assignment:
  push a new decision level onto the stack of α
  select p ∉ α, ¬p ∉ α
  add p to α
  goto propagate
```

Fig 3: iterative DPLL returning all solutions, chronological backtracking

It would have been possible to formulate DPLL in fig. 3 iteratively without goto. However we believe it to be more clear with goto since that saves us from double checking whether α is conflicting or all clauses are satisfied. We will extend this algorithm further in the section about conflict directed backjumping and in the section about stack redos. The complexity of the code will increase and so will the quest for accurate flow control.

The code in *next_assignment* does nothing more than to explicitly take control of the assignment stack. Each level starts with a *decision literal* p, that is a literal which has explicitly been selected by *extend_assignment* as opposed to literals inferenced by unit propagation. What DPLL in fig. 2 does on return, is to pop the current activation record from the runtime stack. If *p* had not been complemented yet another call DPLL( ϕ, α¬p ) follows. Otherwise the procedure returns and another activation record is popped off the stack.

Level zero is special in this regard as it contains no decision literal. It contains atomic facts made up of input clauses of length one. If a decision literal is searched for on level zero in *next_assignment* then the only remaining level gets popped, the while loop is left and the control returns from the DPLL procedure.

### 2.2.2 Clause Learning

Every literal assigned through unit propagation has a reason. That is the clause that had become unit in order to yield the literal. Now if a conflict is detected the conflict clause is of the form {¬p,¬q,C} where *p* and *q* are two literals assigned at the conflict level and C takes the place of all other literals in the conflict clause. There need to be at least two literals assigned at the conflict level because otherwise the clause would have become unit or conflicting before. Now the conflict clause can be resolved against the reasons of literals *p* and *q*. The resulting clause contains neither variable *p* nor variable *q* but some other literals which had been false before *p* and *q* were assigned. These literals can also be resolved. Current SAT-solvers go back in the assignment stack α until there remains only one literal assigned at the conflict level. This point is called the 1-UIP. Such a point is guaranteed to exist since the solver can go back up to the decision literal.

The whole dependency can be viewed as a graph. A unique implication point is a point that dominates all paths between the decision literal of the current level and the conflicting clause. Variable assignments are vertices. For each reason and each false literal of the reason an edge goes from the false literal to the yielded unit assignment. The side with the decision literals is called the decision side and the side with the conflict clause is called the conflict side. Each specific resolution step corresponds to a cut in the graph. The desired cut for the 1-UIP cuts through all edges that originate there. Each step back replaces the edges originating in the resolved variable with those that end there. Examples of implication graphs can be viewed in [BHMW08] and [ZMM01].

The procedure for conflict analysis takes the conflict clause as input and iterates over all literals of the current clause. Yet unseen literals of lower levels are added immediately to the output clause while a counter is incremented for literals of the current level. Then it goes back in the assignments of the current level and takes the reason for the current assignment, makes the reason the current clause and decrements the counter of seen literals of the current level. Finally when the counter reaches zero the only literal of the current level is written into the output clause. An implementation of this procedure as well as other important procedures for solvers with lazy data structures can be seen in the article about Minisat [EeS03].

Why do almost all solvers focus on the 1-UIP? First of all the 1-UIP tends to have fewer literals than the 2-UIP as for each newly resolved clause literals of lower levels may be added. Second we want a clause with exactly one literal of the conflict level. This literal is called the asserting literal. Then most current SAT-solvers jump back directly to the lowest level where the newly learned clause becomes unit. This level is called the assertion level. It is the second highest level of all literals in the clause. The highest level is the conflict level from which the assertion literal stems from.

It can be regarded as ideal if the 1-UIP clause happens to locate on half way between the decision literal and the conflict. If it is too far away from the conflict it may be arbitrarily different showing little similarity with the conflict. Besides this a learned clause that is far away from its origin may not be useful because it is simply too long. The more literals that need to become false before the clause yields a unit literal, the less probable is that unit propagation executes on it. If it is too near to the conflict on the other hand it will be too similar and thus have little power to prevent further conflicts. That is why Jin and Somenzi suggest to generate an additional learned clause when the 1-UIP seems to be far away from what is needed [JiSo06]. Such a clause is only generated if there are multiple conflicts on the same clause and when the length of the new clause is appealing. The additional learned clause does not need to be asserting. This means it may contain multiple literals of the conflict level. So one clause is needed for backjumping and another clause for a better pruning of yet-to-be-visited swaths of the search space.

### 2.2.3 DPLL with Conflict Directed Backjumping

We have described in the previous chapter on how to create a 1-UIP learned clause. This clause is asserting as it only contains one literal of the conflict level. The second highest level of literals in this clause is the assertion level. Current solvers do now perform a backjump to the assertion level and en-queue the asserting literal there from the conflict level. After propagation of unit constraints the search continues from this level, selecting new literals possibly different from what has been selected before.

```
proc DPLL( Φ :CNF, α :assignment ):
  initialize α with level 0 and push all atomic facts onto α
propagate:
  α = UnitPropagation( Φ, α )
  if a conflict is detected:
    learnedClause = analyze(conflictClause)
    btlevel = second highest level in learnedClause
    remove everything down to btlevel from α (btlevel remains the same)
    add assertion_literal(learnedClause) to α with learnedClause as reason
    goto propagate
  if all clauses in Φ are satisfied:
    forall solutions s with s ∩ α ≠ ∅: α ≔ α\s
    if α≠∅: yield α
    goto next_assignment
  goto extend_assignment
next_assignment:
  while level(α)≥0
    remove all assignments from α back to the previous decision literal
    if the decision literal has not been complemented yet:
      complement the decision literal
      break;
    remove the decision literal and go one level up in α
  if level(α)≥0: goto propagate
  else: return
extend_assignment:
  push a new decision level onto the stack of α
  select p ∉ α, ¬p ∉ α
  add p to α
  goto propagate
```

Fig 4: DPLL with conflict directed backjumping

Fig. 4 shows pseudo code for that algorithm. The major advantage of this implementation is that it gets out of the conflict space quickly. With chronological backtracking it would only go back to the next previous uncomplemented level, the next level where both variable choices have not yet been explored. However this level may turn out to produce the same conflict in a similar subtree and the whole process starts over and over again. By non-chronological backtracking we can be sure that the solver gets out of the conflicting space by a singleton backjump. This saves valuable time.

We have added a code line to avoid returning duplicate solutions. For any non disjoint solution subtract that solution from the newly found solution. Remember that each partial assignment corresponds to a set of solutions where the unassigned variables can take any value. While we have written setminus as an operation on sets the algorithm will need to operate on partial assignments. Before *s* is subtracted from α, we test whether the intersection of both sets is empty. This is the case if there is a conflicting assignment $x^{\alpha} \neq x^{s}$. Whenever that happens there is nothing to subtract. The second case to consider is when $\alpha \subseteq s$. This can be detected for each assignment to be either the same in α and s or not present in s. If so no new solution has been found. The final case to

consider is when the set difference yields a smaller set for α' than α was initially. It is computed by assimilating all assignments from s which are not yet present in α in complemented form.

### 2.2.4 DPLL with Stack Redo

Conflict directed backjumping jumps back to the level where the assertion literal can be enqueued. In case of a stack redo it also starts like this. However then it tries to reestablish the previous assignment stack level by level. This may succeed for some decision literals and their levels but produce a conflict somewhere in between the *backtrack level* and the *conflict level*. It does not need to start from scratch for every level but it can put everything that was previously known to be on that level in the *assignment queue* before starting unit propagation.

Reinitializing the assignment queue for each level that is redone has the purpose that new literals can be inferenced out of the assertion literal on each level above the assertion level in addition to the old stack content. A stack redo combines the efficiency of conflict directed backjumping with the sequential search order of chronological backtracking. Since no swath of the search space is visited again stack redoing is even more efficient than simple conflict directed backjumping. The sequential search order guarantees that no duplicate solutions can be returned.

What the stack redo does is in deed a combination of chronological backtracking, conflict directed backjumping and a re-establishment of the previous stack content. At first it starts with chronological backtracking. If the assertion level is at least as high as the backtrack level for backjumping it already ends here. The difference is that the top literal is already complemented and the assertion literal can thus no more be enqueued. Already flipping the literal brings the search faster forward than a backjump search which needed to explore the unflipped state first. In case that there was no conflict but search simply continues to find the next solution after a previous solution has been yielded, it also ends here.

However in the majority of the cases the assertion level lies far deeper in the stack. Chronological backtracking would now continue from the current level and likely re-explore the same conflict in a newly built search-subtree. It would do so for one level after the other until it gets out of the conflict area. This can be very time consuming. Instead of searching for the first non-conflicting level from the top down as chronological backtracking does a stack redo jumps down to the assertion level and searches from the bottom up till the last level where no conflict is discovered. All levels above the first conflicting level are now known to be also conflicting and can thus be skipped for chronological backtracking. What the stack redo does on the first conflicting level (called the *hook level*) is to continue chronological backtracking from there.

In detail there are two possible cases to be considered when the stack is reestablished. The first case is that a new conflict is discovered at the unit propagation of the current level when it tries to inference additional literals. If a new conflict is detected the whole procedure restarts from the very beginning, now with the new conflict instead of the old one. Another case to consider is that some literal in the old stack content now already appears in complemented form on a lower level. This is clearly a contradiction and it means that the current level can not be redone. However now we do not have a classical conflict but simply two contradictory literals. As it is always legal to continue with chronological backtracking instead of conflict analysis this is right what we do then.

Some additional conditions need to be checked for when doing a stack redo, namely checking for duplicate literals in addition to contradictory literals. Duplicate literals can appear somewhere in the middle of the inferenced literals of the current level and are then simply deleted in place maintaining the order of the other literals for conflict analysis. If a decision literal happens to be a duplicate literal then the whole level can be cropped as all its literals have already been inferenced by unit propagation on a lower level. The last condition to check for is that a decision literal can become a duplicate literal after it has been flipped by a continuation of chronological backtracking. In this case the current level is also cropped and the search can continue selecting a new decision

literal.

```
proc DPLLwithStackRedo( Φ :CNF, α :assignment ):
  initialize α with level 0 and push all atomic facts onto α
propagate:
  (α,conflict) ≔ UnitPropagation( Φ, α )
  if all clauses in Φ are satisfied:
    yield α; p ≔ 0
    goto next_assignment
  if conflict = ∅: goto extend_assignment
conflict:
  learnedClause ≔ analyze(conflictClause)
  btlevel ≔ second highest level in learnedClause
  p ≔ 0
next_assignment:
  remove everything from the queue and unset the variables therein
  while level(α)≥0
    remove assignments from current level, save the decision literal in p
    if the decision literal p has not been complemented yet:
      complement the decision literal (put it back on this level)
      break;
    remove the decision literal and go one level up in α
  if level(α)<0: return
  if conflict = ∅ or btlevel ≥ cur_level:
    enqueue ¬p, remove it from tail and set complemented flag for p
    goto propagate
  prev_level ≔ cur_level   // conflict level
  cur_level ≔ btlevel
  enqueue assertion_literal(learnedClause) and set its reason
  (α,new_conflict) ≔ UnitPropagation( Φ, α )
  while new_conflict = ∅ and cur_level < prev_level:
    reenter_next_level
    while decision_variable(cur_level) is already in the trail:  // fig. 8
      delete current_level and make levels on top of it move down
    initialize the virtual queue for this level  // fig. 8
    if virtual queue for this level would conflict with α:
      break;  // ∃ hook level = cur_level+1
    cur_level≔cur_level+1;
    (α,new_conflict) = UnitPropagation( Φ, α )
    if new_conflict:
      restore assigned variable bitfield to state of reenter_level
      set all redone variable as assigned in bitfield
      set all variables in the queue as assigned in bitfield
      add the first non redone variable to the front of the queue
      unset reason and level for all non-redone variables that …
                          have not already been set on a lower level
      delete all non redone variables from the trail
  if cur_level < prev_level && !new_conflict:  // redo level < conflict l.
    p ≔ decision_literal(cur_level+1)   // literal from hook level
    if cur_level+1 = prev_level: unflip p and unset its complementation
    conflict ≔ NULL
    unset complementation flag for decision literal cur_level+2..end
  else:   // ∃ new conflict or a full redo was successful
    unset complementation flag for decision literal cur_level+1..end
  erase cur_level+1 … prev_level in trail
```

```
  if new_conflict:
    conflict = new_conflict;
    goto conflict
  if cur_level < prev_level:  // make sure we have a complemented literal
    if variable p is not already set on a lower level:   // fig. 7
      enter_new_level
      goto next_assignment
    else:
      unset the complementation flag for p   // fig. 9
      // new assignment on lower level is not conflicting with ¬p:
      goto extend_assignment
  // fig. 6
extend_assignment:
  enter_new_level
  select p ∉ α, ¬p ∉ α
  enqueue p
  goto propagate
```

Fig 5: DPLL with stack redo

The main difference between a literal being in the queue and in the *trail α* is that it has not yet been *unit propagated* as long as it is in the queue. The assignment queue is then de-queued literal by literal propagating each literal and finally placing it in the *trail* α. Unit propagation needs to be re-executed because the new literal on the assertion level may yield new propagation results.

The major advantage of stack redos is that they save previous work where the literals between the backtrack level and the *redo level* do only need to be supplemented by new literals. It does not need to re-explore that whole search space.

To make the implementation more efficient DualSat does not shuffle the data from the trail into the queue and then back into the trail but it virtually adds the literals that have been on the current redoing level previously into the queue. This means the de-queue operation first looks for previous literals on the current level and then on what really is in the queue.

The implementation becomes a bit tricky when the solver detects a conflict while it is redoing the virtual queue. It will need to jump back to conflict analysis thereafter but this code section and especially the conflict analysis there will expect everything that has been redone successfully in the trail and the literal that caused the conflict at the front of the queue. All other trail literals that pertain to the virtual queue must be deleted then. Their *level* and *reason* is unset in order to leave clean data structures. Unsetting the reason is important for unlocking clauses. As long as a clause remains the reason for a unit propagation implication, it must not be deleted when tidying up old clauses because it could be needed for conflict analysis. You can see this procedure in fig. 5 at “if new_conflict”.

Special precautions must be taken if a literal should be redone that has now already been inferenced on a lower level. Such a literal is deleted from the virtual queue. For a possible later conflict analysis the order of the literals must be maintained thereby. When the literal to be deleted happens to be a decision literal then the whole level needs to be removed, all the levels on top of it moving down one level.

Another case appears when for any literal which should be put into the virtual queue there happens to be already the same literal in negated form on a lower level. In this case we do not have a new conflict but the old stack content is simply conflicting with the newly derived literals. The current level cannot be redone in this case and the redo level will remain one level below. We say the current level becomes the *hook level*.

DualSat can now simply continue with new variable assignments falling through to the label *extend_assignment*. However it would be better if it did not make a choice on any variable next but on the same variable as before. If it does it can either complement this variable if it has not been complemented yet or go to *next_assignment* to search for any uncomplemented variable down the stack. The effect is the same as if it would have done chronological backtracking until no more conflict appears.

Chronological backtracking goes down the stack until it finds an uncomplemented variable. If the current level is still conflicting it can continue this process until it arrives at a level without conflict. This is exactly what DPLL with stack redo does in effect. It searches the last level where there is no conflict from the bottom up and on the level above it applies *next_assignment* to flip the next variable down from there. The code needs a case distinction if the level above happens to be the level where chronological backtracking has ended before to already complement the decision variable. This variable then needs to be uncomplemented temporarily because *next_assignment* wants to complement it again. The final effect may be the same as for chronological backtracking but what a stack redo does is much faster in effect.

Let us look once more on what the stack redoing code section really does. At first it applies chronological backtracking. Then, if there has been no conflict or if it already has undone the stack till the assertion level *(btlevel)* it has finished and goes over to propagate. If the assertion level lies lower it continues to do the same as on a *conflict directed backjump*. However it does not forget the stack when backjumping but keeps everything in place for the following stack redo. We have already described how the stack redo works in detail. The stack redo can break with a new conflict in which case the whole procedure loops starting at chronological backtracking first. If the old stack content does simply not fit the new one, it continues with chronological backtracking. However all the levels in between the *redo level* and the *conflict level* can be skipped as they are already known to conflict with the new assignment inserted down below at the assertion level.

There is a little extra condition we need to take account of in case of continuing chronological backtracking: If it is the decision variable that appears in complemented form on a lower level then this decision variable which is on the so called hook level must not be simply complemented. Since its non-complemented form was conflicting its complemented form would be the same as down below in the stack. Now instead of establishing a duplicate decision variable, we simply continue with *extend_assignment* selecting a new decision variable and this perfectly matches a continuation in sequential search.

One last note about the case in which no *backjump* appears because initial chronological backtracking was already successful. As the topmost literal is now complemented, we are not allowed to enqueue the assertion literal though we would be on the right level to do so. The new learned clause could only fire if the topmost literal was still in an unflipped state.

If we do chronological backtracking only, we do not need any *nogoods* as the whole solution space is searched through in sequential order. However initially DualSat did not continue with chronological backtracking but it was selecting new variables after stack redo on from the level where the redo was no more successful. If you do this you still lose some part of the execution or search state. Then it is possible that the same solution is visited again. This was initially solved by adding nogoods. That are learned clauses which explicitly prevent the solver from entering the same search space again.

### *2.2.4.1* Practical Examples for the Stack Redo

In this section we want to present examples for the stack redo algorithm described in the previous section.

| | | | |
|---|---|---|---|
| ε5 | h k r | | |
| ε4 | g | | |
| ε3 | f | ¬f | ¬f z |
| ε2 | c d e | c d e | c d e y |
| ε1 | a b | a b | a b ¬k x |
| ε0 | m n o | m n o | m n o |

```
rules on the conflict level: (¬h,k), (¬k,r), (¬k,¬n,¬b,¬r)
learnt: (¬k,¬b)
additional rules: (k,¬m,x), (¬x,¬e,y), (¬y,f,z)
```

Fig 6: full stack redo starting with chronological backtracking

Fig. 6 shows an arbitrary stack content at the left. Decision literals which have already been complemented are shown underlined. The middle column shows the same stack content after chronological backtracking. Since *h* and *g* have already been complemented, they are popped from the trail and the decision literal which is then found at the top is complemented thereupon (here: *f*).

Now a stack redo continues analyzing the conflict that has caused backtracking. If we resolve the last two clauses in ‘rules on the conflict level’ then we receive (¬k,¬n,¬b). The ¬n literal does however not become part of the learned clause because it is a level zero literal. Level zero literals are atomic facts that will always hold true for any solution. Note that the 1-UIP can be found at *k* which means that the conflict directed backjump does not need to insert ¬h on a level below. If the rule were (¬h,¬c,k) instead of (¬h,k) that would be level two rather than level one.

The assertion literal is ¬k. The second highest level in the learned clause is the level of ¬b. Going to this level (the assertion level) we enqueue ¬k. The additional rules we have listed do now cause *x*, *y* and *z* to become inserted on the remaining redo levels. In this example a full stack redo does succeed keeping the complemented literal ¬f at the top. The execution thread now falls through to the label extend_assignment.

| | | | |
|---|---|---|---|
| ε5 | h k r | | |
| ε4 | g | | |
| ε3 | f | ¬f | |
| ε2 | c d e | c d e ↯ ¬e | ¬c |
| ε1 | a b | a b ¬k x ¬e | a b ¬k x ¬e |
| ε0 | m n o | m n o | m n o |

```
additional rules: (k,¬e)
```

Fig 7: when the old stack content conflicts with the new one

Unfortunately in practice it is rather an exception if a stack redo succeeds to reestablish all previous levels. It is more likely that one of the newly established literals either conflicts with the previous stack content or that it causes a completely new conflict on an intermediate level. The former is the case in fig. 7. When literal *e* should be reestablished that causes a conflict. This means that the redo level remains at level one and level two becomes the hook level. As described in the previous section the literal on the hook level is complemented starting chronological backtracking from there.

The algorithm behaves slightly different if it is not the old stack content that is conflicting with the new one, but if an additional conflict is raised and discovered as a consequence of enqueueing the assertion literal. A valid example for this would be caused by (¬y,q) and (¬y,¬q) for level 2 as shown in fig. 6 if (k,¬e) is missing. This will invoke another conflict resolution. It would discover that the fact y is atomically true and enqueue it on level zero. As the conflict level also happens to be level two, chronological backtracking would likewise be started from there also complementing *c*.

| | | | |
|---|---|---|---|
| ε5 | h k r | | |
| ε4 | g | ¬g | |
| ε3 | f x i j | f x i j | ¬g |
| ε2 | c d e | c d e | f j |
| ε1 | a b | a b ¬k x i c d e | a b ¬k x i c d e y |
| ε0 | m n o | m n o | m n o |

`assertion literal:` ¬k, some rules: (¬f,x), (¬x,i), (¬f,j), (k,¬m,x), (k,c), (¬x,¬e,y), (¬y,f,z)

Fig 8: removal of duplicate literals and levels

Now we want to show how duplicate literals that have newly been inferenced on a lower level become deleted. The first such literal in fig. 8 is c. Since it is a decision literal the whole level can be cropped. Literal x on the contrary only causes itself and the dependent literal i to be removed on that level which now falls down from altitude three down to two. Finally the whole redo succeeds with ¬g at level 3.

| | | | |
|---|---|---|---|
| ε5 | h k r | | |
| ε4 | g | | |
| ε3 | f | ¬f z | |
| ε2 | c d e | c ↯ ¬c | w |
| ε1 | a b | a b ¬k x ¬c | a b ¬k x ¬c |
| ε0 | m n o | m n o | m n o |

`additional rule:` (k,¬c), (¬d,¬e,¬x)

Fig 9: stack redo: same literal on lower level

Finally we show a conflicting decision literal in fig. 9. Normally we would establish ¬c on level two by complementing the existing decision literal c. As ¬c does already appear on level one, it must not be reestablished as decision literal on level two. Instead the code jumps to *extend_assignment* selecting a new decision literal *w*.

## 2.3 DNNF, Extended Solution Classes and Other Known Algorithms for SAT-Solving

We have already discussed the Davis Putnam algorithm in chapter 2.1.2. We have also mentioned Stålmark's algorithm and its successor HeerHugo. They do not need a formula to be in normal form but assign a variable to each subexpression. It assumes that the whole formula is false and tries to derive a contradiction. That is why Stålmark's algorithm is called a tautology tester [BHMW08]. It has a set of inference rules which can be applied. If this does not succeed it uses the *dilemma rule*: It sets all possible values for one or more variables and tries to derive new sentences. Those sentences which are the same under all assignments are kept. Something similar to the dilemma rule is also used in modern SAT-solvers: f*ailed literal detection*. For each variable the solver tests whether it can be set true or false without causing a conflict. If a conflict occurs that value needs to be prevented and the variable must take the other value. The other value of the variable can be pushed as an already complemented decision variable on the stack as it would not have any reason otherwise. Besides these two algorithms the application of distributivity and De-Morgan is another way to solve a propositional sentence as already mentioned in section 2.1.1.

Another method that has been popular for some time for solving SAT are ordered binary decision diagrams. You start at a root node and every other variable node has a choice between zero and one. You end in one of two nodes saying the propositional sentence was true or false. Such OBDD are built using the apply method for dual operators. It starts at the root node of each diagram. If both diagrams contain the same decision variable it creates a node for choice one in both successors and another one for choice zero in both successors. If a diagram does not have a decision variable it stays there at the same node until it is the turn for this decision variable. At the leave node it can choose according to the actual result of the operator. In order not to create $O(n^2)$ new nodes it has a cache matrix indexed by node ids to see when two path join again, where it can take successor nodes from. After apply it additionally simplifies the graph by reduce [By86].

A logical expression is said to be in Decomposable Negation Normal Form (DNNF) when the negations have been pushed down to the leaves and when each and-node does not contain leaves with the same variable. It is allowed that an or-node has leaves with the same variable though. That is why you can normally not count the number of solutions in a DNNF: because of duplicate solutions. A DNF is already in DNNF but it can contain the same or similar solutions multiple times [DwDN01].

However the author has explored the properties of DNNF further. If a CNF is in DNNF then in deed you can count the number of solutions. We have already shortly introduced extended solution classes that correspond to a remaining CNF in DNNF in chapter 1.1.

For a CNF the AND-node is the root node and the OR-nodes are one level below the leaves. Consequently a CNF is in DNNF exactly when every variable appears only once. Such variables are called unentangled variables. The opposite is an entangled variable: Assigning a value to an entangled variable immediately changes other OR-clauses by either making them satisfied or by reducing the number of literals that are still available for assignment and thus for making the clause satisfied.

An example of such an extended solution class would be 11x(-2)(-2)(+2)x(+3)(-3). This extended solution class contains two clauses that need to be satisfied: Clause two and clause three. One variable in each of these clauses needs to become true which can be done by selecting a zero for a negative placeholder and a one for positive placeholder. All other variables may take arbitrary values after this initial selection step.

Let us exemplify this with the solution class 10(-2)(2)(-2). It can be decomposed into the simple solution classes 100xx, 10x1x and 10xx0. In order to specify the simple solution classes for it in a disjoint matter we need to subtract the previous classes for every newly specified class. This yields 100xx, 1011x and 10100 in our example. As only one variable needs to become true in every

remaining clause, clause 2 just effectively prevents one solution (the one where all variables are set false). It prevents 10101 from the class 10xxx.

In order to come back to our initial example we need to combine each subclause of clause two with each subclause of clause three in order to obtain simple solution classes from extended ones. An extended solution class with an n-variable, an m-variable and an o-variable clause will thus be decomposed into n*m*o simple solution classes.

How do we count the number of solutions in an extended solution class? At first we may assume that the variables of all clauses can take arbitrary values. Then we need to prevent any solution that would make all variables in a clause false. We can adjust the solution count for a clause with s variables by dividing by $2^s$ and multiplying by $2^s$-1.

By the usage of extended solution classes the author aimed to make the benefits of Darwiche's c2d compiler for d-DNNF also available for well known DPLL-solvers. Though the c2d solver could not successfully solve as many problems as a DPLL solver, it yielded success on plenty of problems where a pure DPLL did not succeed.

Now let us get to what Adnan Darwiche has initially developed when exploring his new format of logical expressions called Decomposable Negation Normal Form or DNNF. He has developed the following algorithm to decompose any logical expression into DNNF. Adnan Darwiche simply added a choice on variables that are common in at least two successor branches: $\Delta_1 \wedge \Delta_2 \Leftrightarrow \bigvee\beta\ (\ (\Delta_1|\beta) \wedge (\Delta_2|\beta)\ \wedge \beta\ )$. Written differently $F \Leftrightarrow (\ F|a \wedge a\ ) \vee (\ F|\neg a \wedge \neg a\ )$. As both branches of the or-node stand for logically disjoint solutions, a DNNF obtained with this algorithm can be counted easily by adding the solution count of both branches. Later Darwiche refined his reports and stated that such a DNNF is to be called deterministic, i.e. d-DNNF [DwCD02]. D-DNNFs are very similar to OBDD because they make a decision on a variable for zero or one below each or-node. In a fact each OBDD is also a d-DNNF with two leave nodes zero and one. In general d-DNNF is a more compact format than OBDD are because decomposition can halt as soon as the variable set of the left and right subtree are disjoint. Another advantage of d-DNNF is that it is more flexible since the variable order in the left and right subtree can be different.

OBDD and d-DNNF are both so called knowledge compilation schemes. They need a previous compilation step before they can answer queries. However then answering questions about the compiled propositional sentence is usually more fast.

## *2.4 Efficient Data Structures for SAT-Solvers*

SAT-solvers do not simply store a CNF as a set of sets as indicated in section 2.1.1. Boolean Constraint Propagation needs to act whenever a literal becomes false. In order to do so the solver cannot traverse the whole set of clauses every time in order to find such literals. That is why at least all backtrack search solvers have a data structure called adjacency list. Via the adjacency list of a literal one can quickly find all clauses where that literal appears in negated form. Whenever a new variable is assigned, it finds all clauses with literals which have become false due to the assignment. Competitive solvers spend from 85% up to 95% of their execution time with unit propagation [GuMi05]. It depends very much on the algorithms a solver implements on how these adjacency lists are organized in detail.

### 2.4.1 Literal Counting

The first solvers implemented a scheme called literal counting. Each clause contains a counter for satisfied and unsatisfied literals in addition to the literals themselves. Whenever the count of unsatisfied literals reaches N-1, Boolean Constraint Propagation on the remaining literal is executed unless it is an already satisfied literal. If the number of unsatisfied literals becomes N in the clause a conflict is detected. Conflicts are also detected when a literal that should be en-queued for

propagation meets a literal on the same variable in the trail which is a negation of the literal to be en-queued. There have been attempts to remove satisfied clauses from adjacency lists but this turned out to be not worth doing. Likewise instantaneously removing unsatisfied literals from clauses has proven to be inefficient [LyMS05]. When setting a new variable it should only need to increment the respective counters but not traverse the clause to move the given literal to the end. Consequently DualSat was basically implemented to traverse all literals once the count reaches N-1 in order to find the sole unassigned literal. The newest version implements hybrid data structures for core clauses where there only needs to be a satisfied literal counter. We need this counter to detect when all clauses are satisfied in order to prevent unnecessary variable assignments when obtaining solution classes which are as compact as possible. Early satisfiedness detection is considered an advantage if a post-processing step on all found solutions should be invoked.

### 2.4.2 Watched Literals and Lazy Data Structures

Just incrementing a counter whenever a variable gets assigned appears to be considerably fast but we can do better. Imagine a long clause with many literals. Each literal triggers several increments for all the clauses. In deed we are only interested if all but one literals get assigned false. To detect this it is sufficient to watch two unassigned literals. If one gets false we search for another unassigned literal. If one of the two watched literals is true we do not even need to do that. If the search finds no unassigned literal and no true literal then it is time for unit propagation. We know that the other literal is yet unassigned. It was not found true and the current literal has become false; otherwise the adjacency list would not have pointed us to this clause. Effectively we just need two non-false literals to detect when one of them gets false forcedly in order to invoke unit propagation. The really good thing about watched literals is that we only watch two literals and that we do not need to do anything on backtracking. For literal counters we need to decrement them. A watched literal of any value can only get unassigned during backtracking which definitively is what we want. Even better as assignments are undone in a stack, the last literal that remained false for unit propagation is the first one to get unassigned on backtracking. This scheme was first implemented in the Chaff SAT-solver [MMZZ01]. The article about Chaff also points out that if the same literal that was previously unassigned should get reassigned soon, then there is little work to do since the watching points have already moved away from the previously assigned literals. It may be another reason why the stack redo as suggested in section 2.2.4 is well efficient.

The article about the Minisat solver on the other hand points us to a very efficient implementation strategy for watched literals: Instead of creating two pointers in the clause head for what literals are watched, the two watched literals are always stored in positions zero and one and exchanged with other literals as soon as they become false [EeS03]. DualSat does even exchange with the zeroth literal for core clauses with literal counting so that the procedure to analyze conflicts knows where the propagated literal is. This is the literal that will be resolved away when the reason for this literal is considered in the conflict analysis process.

DualSat has also avoided virtual classes in order to make method invocations faster. It then only needs a bit array to detemine whether a clause is learned before bumping its usage counter. Keeping things the same for the zero-th literal is another implementation detail of this decision.

The Minisat implementation always searches for non-false literals in ascending order and thus needs to progress through an area first, where we need to expect many false literals due to previous exchange operations. It was a test with DualSat to let it start at any point in the clause and then traverse round robin so that the chance to find an unassigned literal will initially be higher. It turned out that this implementation is slightly slower than the original one. The cause for this is the caching behaviour of the CPU. The literals at the front reside in the same cache block as the two watched literals do and can thus be accessed without any penalty for main memory access.

Before Chaff implemented the watched literal scheme the SATO solver implemented a

different algorithm where watched literal scanning only moves from the outer edges inwards [Zh97], [LyMS05]. On assignment-undo it just restored the old pointers. Not surprisingly this was slower, and that apparently not just because of the stored pointer management that executed on backtracking but also because of the described caching effects. Nonetheless they were the first ones to see that only two literal references need to be kept for detecting unsatisfied clauses and for Boolean Constraint Propagation.

### 2.4.3 The Data Structures of DualSat

As we have already mentioned DualSat keeps a literal counter to detect when all clauses are satisfied and it uses watched literals otherwise. That is where its name stems from: It has two types of data structures.

Some things need to change when you start to additionally do literal counting for some clauses. That has become obvious when looking at the code samples of Minisat. Most importantly the CNF::propagate procedure of DualSat is not allowed to put a literal into the trail before it has been fully propagated. During propagation there can arise a conflict any time. In this case everything needs to be undone which is on the same level.

For literals in the implication queue they only need to be unset. For literals which have been picked up by the trail, the literal counters of all associated clauses are undone. Undoing a literal twice would be fatal so a literal does not go into the trail before the propagation of that literal has finished. If there arises a conflict intermittently then it is the propagate procedure itself which is responsible for undoing what has already been counted, before it returns the conflict to the main procedure of DPLL.

Many solvers have special provisions for dual clauses. Watching two literals and always going to a clause in memory that does not contain any more than the two literals which are watched anyway is a bit unefficient. That is why DualSat fully stores dual clauses in the adjacency list. Instead of a reference to a clause it simply stores the other literal itself. The clause is stored in two adjacency lists. This provision only holds for core clauses which are provided by the input problem. The structure for learned clauses is always the same as it additionally contains a clause activity counter so that you could hardly optimize the structure itself away except if you decided to keep learned binary clauses forever. Certain solvers do even implement special provisions for ternary clauses [GuMi05].

The adjacency list for a literal is a contiguous memory block containing all the indices of core clauses for literal counting. In its header there is a vector of variable length for the adjacency list of learned clauses. This list changes: Every time a watched literal is propagated its list is cleared and the newly watched literal needs to get inscribed into the appropriate adjacency list vector. Clearly such a vector needs a reference to a dynamically allocated memory block. It has shown that a simple vector implementation to keep this dynamic memory block contiguous and to double its extent on overflow was slightly faster than the implementation of the C++ standard library, at least for the way our solver uses it. The trail is also a dynamic vector of vectors because it needs to allot space to literals added later on a stack redo at every level. The most important benefit of the own vector implementation is however that it is much easier to debug with it, because the data array can be printed directly.

DualSat does not store pointers to core clauses. Instead it always refers to such clauses via an index. Sometimes it is not necessary to dereference that index. When testing whether a clause is already satisfied it can do so by testing the index against a special bitfield called *bsat*. Dereferencing an index is done by looking into the base clause array which then contains a pointer to the actual clause for that index. That array is rather small so it incurs no or little penalty when it comes to memory access. Memory access costs dominate the access time to learned clauses. For core clauses the memory access time is less significant but it may still be relevant. As DualSat will

usually execute on a 64bit machines a 32bit index is also just half the size of a pointer. Nonetheless DualSat also compiles well on 32bit arches. Keeping structures like adjacency lists compact also saves memory access time. The *bsat* bitfield is used in unentangled literal detection, when counting the number of solutions for a solution class and for computing the dynamic but conflict independent heuristic that DualSat uses in addition to VSIDS.

## *2.5 DualSat*

One of the most interesting features of DualSat is the stack redo feature as discussed in section 2.2.4. Most features of DualSat are discussed in this chapter. Those who are interested in the nogood handling of DualSat may refer to section 2.6.1.

### 2.5.1 Unentangled Literal Detection

We have already described in chapter 1.1 and chapter 2.3 that DualSat finishes the search for a new solution class as soon as the remaining CNF is known to be in DNNF. That is when it returns a solution class in the format as described in chapter 1.1. Such a solution class can also be counted easily for #SAT.

The remaining CNF is in DNNF as soon as there are not any entangled literals any more. An entangled literal is a literal that appears in more than one clause. Changing the value of the literal in one clause immediately affects the other clauses. We simply call this effect entanglement.

In a fact it does not check at each step whether the remaining CNF is in DNNF. It checks whether a literal is still entangled whenever it wants to select this literal as decision literal. If it is already found to be unentangled then DualSat continues to search for another literal to set. If there are only unentangled literals left over, then we know that our CNF is already in DNNF.

The procedure to test whether a literal is entangled heavily requires tests whether a clause is already satisfied. Satisfied clauses do not need to be considered for the remaining CNF. When new variables are set the remaining CNF gets smaller and smaller yielding more and more unentangled literals. The procedure notes all clauses where the literal is contained by its two adjacency lists and excludes those clauses which are already satisfied. If the clause count happens to surpass one then the literal is entangled. The satisfication test can be performed very efficiently by testing the clause index against the respective bit in the *bsat* bitfield.

However testing a variable each time for entanglement when it shall be considered by the variable selection heuristic would be too inefficient. That is why DualSat has the *vopen* bitfield which stores all variables that have already been found unentangled. Every time a decision variable is popped from the stack the *vopen* bitfield is restored to its previous state. That is each level of the assignment stack contains a saved copy of *vopen*. When moving down the stack detected unentangled literals are saved by copying *vopen*. This provision is necessary because there is no way to detect all the unentangled variables caused by setting a decision literal and then executing unit propagation. Setting the *vopen* bit true just for the decision literal on a backtrack is not sufficient.

### 2.5.2 The Variable Selection Heuristic of DualSat

DualSat was initially developed on random CNFs. Such propositional sentences expose very little or no conflicts. That is why it was important to have a conflict independent heuristic. We already know that dynamic heuristics can make the search process more flexible and efficient. Algorithms using static heuristics which are bound to the same variable order on each branch like Ordered Binary Decision Diagrams or the Davis Putnam algorithm are less flexible than an algorithm where each branch can select the variable to be assigned next based on the remaining

CNF which only contains clauses that have not yet been satisfied. Dynamic heuristics dominate the implementations of DPLL and backtrack search as well as Decomoposable Negation Normal Form, d-DNNF.

Similar to the way DualSat has assimilated old style literal counting to check for which clauses are already satisfied, it has also had a look at traditional variable selection heuristics to develop a new conflict independent dynamic heuristic. Marques-Silva presented several conflict independent heuristics in one of his papers [MqSv99].

BOHM's heuristic creates a vector with all the unresolved clauses of each length: $H_i(x) = \alpha \cdot \max(h_i(x),h_i(\neg x)) + \beta \cdot \min(h_i(x),h_i(\neg x))$ where $\alpha$ is usually 1 and $\beta = 2$. Each positive and each negative occurrence of a literal in a not-yet-satisfied clause is counted. Interestingly the heuristic prefers to make decisions on variables where the relation between occurrences of x and ¬x is more balanced. If it is not then we may not need a decision on that variable as unit propagation is still likely to assign the desired value. More important however is that we prefer to set a variable with good impact on satisfying existing clauses. Additionally short clauses are to be preferred clearly as they will faster yield a unit propagation in case that they are set false. That is why BOHM's heuristic first selects the $H_2(x)$ with maximum value and only if that does not yield a result it considers $H_3(x)$ and $H_i(x)$ for longer clauses.

Another interesting heuristic is the Jeroslow-Wang Heuristic. It considers clauses of all length , however with a decay factor of ½: $J(l) = \Sigma [\, l \in w, w \in \phi \,] \; 2^{-|w|}$. The heuristic always sums J(l) and J(¬l).

DualSat has now combined both heuristics by multiplying each $H_i(x)$ with $2^{-i}$. That is very simple and effective. Nonetheless it is possible to do better. Instead of a factor of two we have taken an initial multiplier of $\delta = 1.4$ which then decays by $\varrho$. The maximum considered clause length is N=5. That should bound the computation effort and strip irrelevant results. This means it takes: $( ( ( H_2(x) \cdot \delta ) + H_3(x) ) \cdot \delta \cdot \varrho + H_4(x) ) \cdot \delta \cdot \varrho^2 + H_5(x) = H_2(x) \cdot \delta^3 \cdot \varrho^3 + H_3(x) \cdot \delta^2 \cdot \varrho^3 + H_4(x) \cdot \delta \cdot \varrho^2 + H_5(x)$. Intuitively $\varrho$ should be selected the way that $\delta \cdot \varrho^{N-3}$ is still greater than one. Otherwise the longest clauses happen to have a greater impact than somewhat shorter clauses. However as far as we could test it out $\varrho$=0.8 gave better results than $\varrho$=0.9 on random SAT clauses as well as a subset of the examples from rutgers.edu. We suggest it means that dual clauses should be clearly favoured, that somewhat longer clauses should still contribute but that the difference in contribution for clauses of length three, four and five is negligible.

Marques-Silva stated that these heuristics would be little better than randomly selecting a variable. Sometimes randomly selecting a variable would even be better since some algorithms can be too greedy. For DualSat we got good results with our heuristic though it took quite considerable time to calculate. Maybe Marques-Silva tested on problem types that are not well suited for conflict independent heuristics. Maybe also the combined heuristic is considerably better than the used individual base heuristics.

Since well known conflict based heuristics perform better on problems with many conflicts DualSat switches the type of applied heuristic dynamically. Whenever a conflict appears the next 16 variable selections are done with the conflict dependent heuristic. Still there can be 16 decisions without any conflict and in this case the conflict independent heuristic contributes to a better result. Generally switching heuristics somehow randomizes the result and prevents the solver from getting stuck in a fixed scheme.

The rational about conflict based heuristics is that variables which recently contributed in conflicts are best to be selected as new decision variables. For Chaff conflict analysis incremented a counter for each variable in the conflict clause. Periodically all scores were divided by two. For exponential VSIDS (variable state independent decay sum) most divisions are saved by incrementing the score with a variable of value $g^i$ which is stored as a floating point number. The constant g takes a value between 1.01 and 1.2 where low values are best suitable for hard satisfiable

instances like cryptographic instances [BFS15]. DualSat currently uses g=1.05. Even if floating point numbers are used to 'bump' variable values, these values can still overflow. That is why $g^i$ must be checked against an overflow when incrementing the conflict index i. If so then all variable activity values and all positive literal activity values as well as $g^i$ need to be divided by a large enough value. Selecting relevant variables becomes an issue as soon as you have a heuristic that is better than random. It is even an issue before as it enables a better recovery on backtracking.

DualSat also uses a positive literal activity counter from which the negative literal activity can be computed with the variable activity counter. While the variable activity counter is for selecting the next decision variable the positive and negative literal activity values are for selecting the initial value for a given variable that has up to now been used least (see chapter 1.1).

It was an innovation of BerkMin over Chaff to bump all variables that get involved in conflict analysis and not just the variables of the conflict clause. Furthermore the next decision variable is only selected from variables of the last conflict clause to pay more attention to the most recent conflict. As Goldberg and Novikov state a single variable reassignment can make the result of an and-gate switch from zero to one thus enabling a whole new set of clauses that will contribute to newly generated conflicts [GoNk07]. Immediately paying respect to such a change is the best to do. DualSat also selects the next decision variable only from the last conflict clause except after a restart. Biere and Fröhlich describe many different derivatives of conflict based heuristics like VSIDS, exponential VSDIDS, ACIDS and VMTF (variable move to front) but come to the conclusion that all these heuristics perform more or less equally well [BFS15].

When developing the conflict independent heuristic for DualSat it was a tradeoff between faster execution time and the time it took to compute the heuristic. That is why the author has implemented provisions to incrementally update the heuristic rather than having to calculate it all the time from scratch. It has turned out that incremental updating is beneficial while new decision variables are selected but not as soon as the solver backtracks. This is likely because backtracking additionally requires undoing choice variable assignments in addition to new choice variable selections afterwards. The result was obtained without using an additional conflict dependent heuristic.

The count for dual variables can directly be updated after the selection of a new decision variable. The key issue about it is that at the time *update_4_dual_do* is called the new choice variable has not yet been propagated and that it can be seen from the other literal whether that clause was previously unsatisfied and will get satisfied. The $H_i(x)$ vector needs to be decremented where a clause gets newly satisfied. The p*ropagate_positive* procedure counts the newly satisfied literals in a clause on propagation and sets the *bsat* bitarray for newly satisfied clauses. If it does so it also updates $H_i(x)$ in dependence of the clause length i.

### 2.5.3 Some Other Implementation Details of DualSat

The most important implementation detail of DualSat we have not yet talked about are likely restarts and clause deletion.

Restarts can only be performed before the first solution is found because otherwise DualSat would need to remember all parts of the search space that have already been visited. Restarts in DualSat are performed by the luby series with a factor chosen to be 16 or a factor of 12 for the very hard hole9 instance [Ru00].

The following number series is the luby series: 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ... . So the series starts with one, copies the whole previous series an then adds $2^k$ with incremented k. With luby the solver makes a few explorations to learn as much as possible and then it lets the solver run respectively longer to obtain a final result. The luby series grows only linearly in the number of restarts.

Luby series are a quite popular method of implementing restarts. However dynamic strategies have been reported to outperform static ones like luby [BFR15]. Biere and Fröhlich analysed dynamic restarts based on the literal block distance (LBD). The LBD is the number of different decision levels in a learned clause. Small values indicate a more important learned clause. Whenever the short term average is substantially larger than the long term average a restart can be triggered (say 25%).

Clause deletion is best based on the clause length, the clause age and the clause activity. The longer a clause is the less likely it is to contribute in preventing further conflicts. Elder clauses are usually less relevant because any switched variable may cause a new set of clauses to become relevant (f.i. if an and-gate is enabled or disabled). The clause activity is currently bumped as soon as a learned clause is created or when it is used in conflict analysis. BerkMin now suggested to keep young clauses with length<43 and activity>7. Elder clauses which constitute just 1/16 of all learned clauses are kept with length<9 and activity>*threshold* with *threshold* being initialized to 60 and gradually being increased. That is exactly what DualSat also does. DualSat does not always increment by one but by $g^i$ with g=1.05 and i being the conflict index. *Threshold* as well as the activity constant for shorter clauses are currently simply multiplied with the clause increment. That effectively compares against the current clause increment, elder incrementations being less significant. The most important fact seems that clause deletion is performed because otherwise that would result in a significant slowdown. The actual parameter of g seemed to be not very significant.

Finally there is a third procedure which slightly contributes to the speed of DualSat: *compensate_literal_movement_of_false*. The idea behind it was that when the assignment stack reorganizes due to a stack redo then variable assignments can change their level. However if a false literal that was previously found on a higher level and which became a watched literal is now swallowed deep in the stack then it may not get unassigned first and the clause would be blocked until the literal becomes unassigned on a very low level. It would then watch a false literal while other literals are already unassigned thus not reacting on the reassignment of these newly unassigned literals and not yielding unit propagation. The issue has been found not to be true by careful verification and testing. In deed all the literals get into the assignment queue on a stack redo. The false literals get unassigned like under normal conditions and are then propagated again as if the decisions were made from scratch. It was a bit surprisingly that the procedure did still give a slight performance benefit. It should not be explained because *compensate_literal_movement-_of_false* always chooses a false literal with maximum level so that less re-searching for watched literals becomes necessary in the future with similar choice literals. At least it comes at a very low cost as this procedure only revisits very recently visited clauses. Maybe it is just because it pays off to keep these clauses in the cache for longer. Variables which change level are thereby collected in a bitfield.

### 2.5.4 Wide Integer Implementation for DualSat

DualSat has its own wide integer implementation since the author wanted to avoid a respective library dependency for counting solutions for #SAT. Another advantage of the own wide integer implementation is, that the source code does not need to be changed when switching from a 64bit integer to a wider data type. The implementation is based on C++ operators and thus the syntax for computations is mostly the same. There are only a very few differences like having to write ‘v!=0’ instead of ‘!v’. The commands for reading and writing integers to the console are also different.

When it comes to counting solutions a 64bit counter is often not enough. However for the examples from rutgers.edu a 128bit counter was fully sufficient. Basically the counter would need to have as many bits as there are variables. Fortunately this requirement does not hold in practice most times. The integers in the implementation under the *types* folder have a width that is fixed on compile time, so a wider integer width must be set in the source code. Nonetheless the definition of

a SignedDblInt or an UnsignedDblInt is recursive basing on a data type of half the width so that you can assemble the desired data type on demand, if it has not already been predefined. The definition of these types goes via a C++ template taking a half width integer and a parameter type class as parameters. The parameter type class defines whether parameters of this type should be passed by value or by reference by default. For integers longer than 128bit passing by reference is recommended on a 64bit machine.

The addition and subtraction operators test for the highest bit before and after the operation in order to detect an overflow of the lower half of the result. This bit can be extracted by converting the type into a signed type and by testing for the sign. When adding there is an overflow if both high bits are set before the addition or if exactly one of them is set before (xor) and the result yields a zero in the highest bit. Likewise for subtraction there is always a borrowing if the first operand has a zero in the highest bit and the second operand a one or if exactly one of them is set before and the result has a one in the highest bit. For incrementing and decrementing the overflow test is as simple as to test the lower half of the operand against zero.

Multiplication of two wideints is implemented by a simple principle. The two operands are partitioned into four high/low areas. Then you can perform the multiplication with variables and shift-multipliers on paper and see how each of the four areas of the two operands contributes to the solution. Finally this just needs to written with the proper bit-shifts into the source code.

In order to read and write wideints to the console it was necessary to multiply and divide by 10 so that an own multiplication and division operator with a second operand of 64bit is supplied. The division can be performed from the highest word down to the lowest word by always adding the remainder of the previous division with a shift of half the word width to the operand. Division is performed using 64bit integers and the divisor operand is allowed to take 32bits. Likewise 32bit multiplication is performed from the least significant to the most significant word in the opposite direction the overflow being added to the multiplication performed next. There are own versions for this division, multiplication and addition that also return if an overflow was detected.

The implementation of the shift operand turned out to be a bit tricky since shifting by zero digits returns a wrong result for the base data type of an int64. Consequently a case distinction for shifting a multiple of 64bit was introduced which performs the operation also faster. Left shifts process each word downwards from the most significant to the least significant word and the right shift goes into the opposite direction. Besides this a fast dual logarithm was implemented that first tests the upper and the lower half of a bitarea for being zeroed.

When creating the wideint implementation the author has discovered a bug in the Gnu C++ Compiler which has become known as Bug 95665 ‘memory access error passing additional parameters in addition to Int128’ [GCC0]. Though the error could also be avoided for GCC by replacing (thisNum>>2)*3 with thisNum>>=2 and thisNum*=3 the wideint implementation is disabled there by default. Nonetheless DualSat was written to also compile with the Clang compiler where it does also execute slightly faster. For Clang usage of int128 is the default.

## *2.6 Interesting #SAT solvers*

In this section we refer to what we know about Clasp and sharpSAT. For the c2d d-DNNF solver please refer to chapter 2.3 and the literature referenced there. Download addresses for the used solvers can be found in the bibliography [cl00], [ZC00], [sS00], [c2d00].

### 2.6.1 Clasp and Nogoods in DualSat

DualSat has been leadingly inspired by the Clasp solver. The initial idea was to write a solver that can enumerate solutions without remembering previously visited solutions, like Clasp can. As the DualSat solver seemed to still require nogoods the reason for introducing them was also inspired

by the design of Clasp. Recent improvements to the stack redo do not require nogoods any more. Nonetheless we would assume that remembering some of these nogoods as learned clauses could still improve the performance. This way the solver would act very much like a human being when searching for a solution: Remember new findings like clauses learned by conflict analysis but also remember coarsely where the solver has already searched for solutions. This may be especially beneficial for search space regions with few conflicts.

Now let us get to what Clasp does. On chronological backtracking and thereby also after a solution has been found it remembers the decision level where chronological backtracking complements the decision variable in the 'backtrack level' (for complementation see fig. 4, label next_assignment). Note that this is different from the DualSat solver where the backtrack level always becomes the assertion level. The idea now is that if a conflict occurs then it does never backtrack below the backtrack level. If the current decision level is still as high as the backtrack level then Clasp continues with chronological backtracking even on a conflict. Otherwise it undoes all decision levels down to the maximum of the assertion level and the backtrack level. This is because new decision literals that do not belong to an already found solution can be backtracked without any problem.

Let us look in greater detail on what Clasp does. When it finds another solution it complements a literal afterwards and that complemented literal assures that none of the solutions which have been visited so far can be revisited. It is in the nature of chronological backtracking that it always keeps at least on literal complemented in comparison to any previous solution. As soon as it pops a complemented decision literal from the stack, it continues to complement at least on literal on a lower level.

The elder solutions cannot be revisited simply because there is always one literal that is flipped in comparison with previous solutions. The idea of DualSat was now that it can backtrack below the 'backtrack level' of Clasp if it remembers that already complemented decision literals need to be complemented again as soon as they reappear on the stack. DualSat continues to consider all already complemented literals between the redo level and the level from which it started to backtrack. That is the window in which complemented literals may reappear in their uncomplemented form, thus traversing parts of the search space that have already been visited. This is prohibited by a nogood for each already complemented literal.

The nogood contains the top literal which is a complemented literal and all the literals below in negated form. Consequently if the literals below happen to reappear in some order then the complemented literal is forced again thus skipping to take its uncomplemented, already visited value.

As nogoods must not be forgotten arbitrarily by clause cleanup the DualSat solver stores them in an own data structure. However if chronological backtracking succeeds to complement a literal then all nogoods on higher levels can be forgotten.

Some nogoods can be forgotten after the stack redo. If there has appeared a variable on a lower level than the redo level which always makes the nogood true then it can also be deleted right after creation. The variable must not be on the same level as it could have been inferenced by the very same nogood for which deletion should be detected. Likewise it can be deleted if it is a decision literal.

Nonetheless use of nogoods to exclude duplicate solutions is a deprecated method and not recommended for future use. We can easily do better by continuing chronological backtracking from the redo level.

One last word to the Clasp solver. It is not a plain SAT-solver but an extension of a SAT-solver for conflict-driven answer set enumeration. Such solvers can exclude cyclically supported solutions where the literals in a cycle are only true because one follows out of the other without external support.

### 2.6.2 sharpSAT and Component Caching

Component caching is a feature not yet implemented with DualSat that promises high gains in speed. As soon as the residual formula falls apart into different contingency components these components can be solved independently, one each at a time. The resulting CNF is unsatisfiable as soon as only one component has been detected unsatisfiable. Furthermore performing independent computation on these components saves the solver to re-establish a sentence from component one before it can push on to solve clauses from component two. This effect alone was found to be able to increase the speed for orders of magnitude in an artificial setting with four times replicating a whole CNF before invoking the solver on it [PiDw07].

We have selected sharpSAT for comparison because of its new and innovative features. Like Cachet it does not only detect but also cache existing components. Only the core clauses are used to identify caching components. This is called *bounded component analysis* [SBB04]. Learned clauses would slow the computation of contingency components considerably down as there are many. Furthermore learned clauses can span multiple components thus reducing the expected benefit.

SharpSAT uses a variable and a clause set for identifying cache components. The variable values of variables in unsatisfied clauses can be deduced because they need to be false there. Components without solutions are not cached as it is the turn for conflict analysis to detect this. If a component has solutions then there is no binary clauses with two unsatisfied literals. Neither are there binary clauses with just one unsatisfied literal because unit propagation would immediately have been executed on them. Binary clauses are thus irrelevant for caching [Th06].

During component analysis it may be beneficial to disallow unit propagation to other components as these results are not needed [SBK05]. The Cachet solver has also settled on analyzing the sub-component with the least variables first. If one component is unsatisfiable so is the whole CNF. As there was no better mean at hand to estimate the probability of a component to be unsatisfiable, Cachet just starts with the component that is most easy to computer. This has been proven better than starting at the component with the largest clause/variable ratio.

SharpSAT uses a VSIDS like score to decide which entries to keep in the cache. It furthermore uses failed literal detection like we have already shortly described it in chapter 2.3. With failed literal detection both truth values for each variable are assumed and if a conflict arises then this value can be excluded. It thus yields more unit results.

Failed literal detection can detect everything that binary clause resolution can detect [Bc02]. Consider the example (a,b), (¬a,c), (¬b,c). It yields c which is found by a conflict on ¬c. However there are inferencing methods which can yield more than failed literal detection: With hyper resolution (h,¬a), (h,¬b), (h,¬d) and (a,b,c,d) inference (h,c). Hyper resolution takes an n-ary clause and n-1 binary clauses for resolution [Br04].

SharpSAT uses failed literal detection only for newly detected binary clauses to save time. The benefit of failed literal detection and more reasoning versus searching as a whole is that it saves from unnecessary component analysis and cache lookup steps.

There are some fallacies with component caching though. Consider two components: A: $(\neg p \vee a)$, $(\neg q \vee b)$ and B: $(q \vee r)$, $(\neg r \vee p)$. From $A \wedge B$ we can inference $a \vee b$. Now let us consider exploring the components A and B after p and q have been set false. Component *B* is unsatisfiable but for component *A,* the learned clause $a \vee b$ further infringes the number of results thus returning a too low count. The issue about it is that ‘ex falso quodlibet’. Since $A \wedge B$ is false because of B, it was possible to inference anything out of $A \wedge B$ that does no more hold now. If a component has an unsatisfiable sibling it must not be cached. Fortunately this issue has already been found by a very early work on component caching [SBB04].

Unfortunately sharpSAT returned a wrong result for two problems from rutgers.edu: 1728832 instead of 17288370 for ii32b2.cnf and 47262042 instead of 47262168 for ii32b3.cnf. Clasp and

DualSat return the correct result. The homepage of sharpSAT mentions that previous versions were already found to return wrong results, but it does not say anything about how that was resolved [sS00]. That way we can only speculate on what issue it could be. Sang et al. says that a component needs to be deleted from the cache as soon as an unsatisfiable sibling is detected. We claim that this is not enough, because the sibling can already have been redetected in another subcomponent before its deletion from the cache is scheduled. This problem may be especially imminent if small components are computed first because these may more easily reappear in a subgraph. The solution for this problem would be to not add a clause to the cache unless a solution for all of its sibilings has been found.

# 3 Conclusion and Outlook

DualSat implements new features like unentangled literal detection, a new conflict independent heuristic and backtracking with stack redo.

| | DualSat | Clasp | ZChaff | SharpSAT |
|---|---|---|---|---|
| rutgers.edu, one solution | not solved: 18 of 244 (7%) | not solved: 16-17 of 244 (7%) | not solved: 17 of 244 (7%) | |
| rutgers.edu, many solutions | not solved: 50 of 244 (20%) | not solved: 60 of 244 (25%) | | not solved[1]: 44 (18%) |
| DQMR, many solutions | not solved: 333 of 559 (60%) | not solved: 100% | | |

Table 10: overview of the benchmarking results

As you can see from table 10 these features contribute to a new and performant SAT-solver. As ZChaff can only be used to find a singleton solution and as SharpSAT does only count but not return solutions the corresponding entries in table 10 have been left out. We have not considered the results of SharpSAT for DQMR as it already returned some wrong instance counts for the Rutgers collection of problems. Not unlikely it may behave similar to Clasp because of its conflict dependent heuristic on problems that did yield only a very few conflicts with DualSat.

DualSat needs to be mainly compared with Clasp as it is a new but further going realization of the basic ideas of Clasp. Nonetheless it could profit from the component caching feature of modern #SAT-solvers like sharpSAT. The main purpose of DualSat was neither to count solutions (#SAT) nor to just find a singleton solution but to return all solutions in deed, compound to solution classes. DualSat can proficiently meet this new objective. It should leverage new applications of SAT-solvers like calculating a non-linear numeric optimization problem for a set of solutions defined by a propositional sentence or input CNF.

The stack redo feature and the unentangled literal detection are particularly beneficial when it comes to find many solutions. Nonetheless the stack redo can be seen completely independent of DualSat´s main application purpose of finding all solutions for a given propositional sentence. It simply saves a solver from re-exploring already visited search areas over and over again and it can do so simply by the being-complemented state of the decision literals which is present in every current DPLL-solver.

The new and conflict independent heuristic leverages new application areas like the DQMR benchmarks [Ca00]. It is combined with the widely used conflict dependent VSIDS heuristic in order to yield good results on a variety of different problem classes. The DQMR benchmark is similar to random SAT instances in that it evokes only a few conflicts. DualSat was initially developed on random SAT instances, a problem known to be particularly hard.

These three features, namely unentangled literal detection, the new conflict independent heuristic and backtracking with stack redo may also prove beneficial in future solvers. The unentangled literal detection is backed by DualSat´s dual data structures. Such a provision may additionally aid the development of better solvers with more reasoning. Restricted resolution, two-variable equivalence and pattern-based clause inference conflict with lazy data structures and may at least be partially leveraged by dual data structures [LyMS05].

New versions of DualSat are still bleeding edge development and there remain a lot of features we are still looking forward to implement. A version of hybrid data structures for core clauses currently needs to be combined with a version of stack redo that employs the continuation

1 two instances returned a wrong solution count (ii32b2, ii32b3)

of chronological backtracking after the stack redo. The latter saves us from additional nogoods that cannot be forgotten. Nonetheless optional nogoods for remembering already visited swath of the search space may help to yield better results.

Hybrid data structures mean that only satisfied literals are counted and two watched literals are kept to identify when the clause becomes unit resulting. Watched literal solvers have been reported to be up to eight times faster than pure literal counting solvers [GuMi05]. The speedup for half of the core clauses will of course be lower but may still be significant. There are also some smaller improvements possible like checking if a core clause has been subsumed by a clause derived by conflict analysis [HJS10]. Here is a quite comprehensive list of possible improvements:

- hybrid core clause data structures combined with full stack redo employing a continuation of chronological backtracking
- transitive reduction of binary clauses
- other preprocessing steps
- detecting equivalent literals by strongly connected components [Br04]
- component identification and caching
- failed literal detection
- subsumption checking on conflict analysis

Table 11: possible improvements for DualSat

May the new techniques of DualSat be inspiring for future solvers and may the current version of DualSat just be the beginning of a prosperous development path. We also assume that DualSat can already be successfully applied to a wide range of real world problems.

# 4 Benchmarks

Here are the results of the last benchmarks we have run with DualSat, Clasp, sharpSAT and ZChaff on an Intel Core i7-8750H 2.2 Ghz with Turbo Boost up to 4.1 Ghz. Remember that sharpSAT still returned wrong results for some problems (see chapter 2.6.2) and thus needs to be considered out of competition. It may still point to what may be possible with DualSat when it implements component caching one day. Please remember also that the primary intent of DualSat was to return solution classes for later processing and that this may justify some additional work.

For solving problems from rutgers.edu being just interested in one solution DualSat did not solve 18 problems while ZChaff and Clasp had to give up on 17 problems of 244 problems in 60 seconds. Initial test have shown Clasp to solve one more problem. Unfortunately these have been lost. Maybe one problem is just at the edge of being solved in 60s and execution times differed because of Intel Turbo Boost. For finding the solution count Clasp had to give up on 60 instances, DualSat on 50 instances and sharpSAT on just 44 instances of rutgers.edu in 60 seconds [Ru00]. Clasp could solve none of the DQMR instances for solution counting while DualSat solved about 40% of these problems [Ca00].

There may be results with DualSat and hybrid core data structures available soon.

## *4.1 rutgers.edu DualSat*

```
      rutgers.edu/samename/jnh8.cnf(0):  0.000s, unsat,    12 confl,      17 dec,      1 nd
   rutgers.edu/samename/par16-1.cnf(1):  1.082s, sat  , 24997 confl,   29811 dec,    100 nd
   rutgers.edu/aim-100-1_6-no-1.cnf(3):  0.000s, unsat,    25 confl,      70 dec,      6 nd
   rutgers.edu/aim-100-1_6-no-2.cnf(4):  0.000s, unsat,    34 confl,      90 dec,      5 nd
   rutgers.edu/aim-100-1_6-no-3.cnf(5):  0.000s, unsat,    33 confl,      84 dec,      5 nd
   rutgers.edu/aim-100-1_6-no-4.cnf(6):  0.000s, unsat,    24 confl,      81 dec,      4 nd
 rutgers.edu/aim-100-1_6-yes1-1.cnf(7):  0.000s, sat  ,    16 confl,      52 dec,      7 nd
rutgers.edu/aim-100-1_6-yes1-2.cnf(8):  0.000s, sat  ,    35 confl,     103 dec,      5 nd
rutgers.edu/aim-100-1_6-yes1-3.cnf(9):  0.000s, sat  ,    29 confl,     107 dec,      9 nd
rutgers.edu/aim-100-1_6-yes1-4.cnf(10):  0.000s, sat  ,    22 confl,      82 dec,      3 nd
  rutgers.edu/aim-100-2_0-no-1.cnf(11):  0.000s, unsat,    12 confl,      36 dec,      2 nd
  rutgers.edu/aim-100-2_0-no-2.cnf(12):  0.000s, unsat,    27 confl,      67 dec,      1 nd
  rutgers.edu/aim-100-2_0-no-3.cnf(13):  0.000s, unsat,    20 confl,      64 dec,      3 nd
  rutgers.edu/aim-100-2_0-no-4.cnf(14):  0.000s, unsat,    18 confl,      52 dec,      2 nd
rutgers.edu/aim-100-2_0-yes1-1.cnf(15):  0.000s, sat  ,    75 confl,     190 dec,      5 nd
rutgers.edu/aim-100-2_0-yes1-2.cnf(16):  0.000s, sat  ,    58 confl,     138 dec,      8 nd
rutgers.edu/aim-100-2_0-yes1-3.cnf(17):  0.000s, sat  ,    20 confl,      68 dec,      6 nd
rutgers.edu/aim-100-2_0-yes1-4.cnf(18):  0.000s, sat  ,    48 confl,     106 dec,      5 nd
rutgers.edu/aim-100-3_4-yes1-1.cnf(19):  0.000s, sat  ,    62 confl,      87 dec,      3 nd
rutgers.edu/aim-100-3_4-yes1-2.cnf(20):  0.000s, sat  ,    70 confl,     114 dec,      3 nd
rutgers.edu/aim-100-3_4-yes1-3.cnf(21):  0.000s, sat  ,    69 confl,     109 dec,      4 nd
rutgers.edu/aim-100-3_4-yes1-4.cnf(22):  0.000s, sat  ,    60 confl,     108 dec,      3 nd
rutgers.edu/aim-100-6_0-yes1-1.cnf(23):  0.000s, sat  ,    52 confl,      70 dec,      2 nd
rutgers.edu/aim-100-6_0-yes1-2.cnf(24):  0.000s, sat  ,    43 confl,      54 dec,      3 nd
rutgers.edu/aim-100-6_0-yes1-3.cnf(25):  0.000s, sat  ,    52 confl,      77 dec,      3 nd
rutgers.edu/aim-100-6_0-yes1-4.cnf(26):  0.000s, sat  ,    86 confl,     111 dec,      3 nd
  rutgers.edu/aim-200-1_6-no-1.cnf(27):  0.000s, unsat,    26 confl,     132 dec,      6 nd
  rutgers.edu/aim-200-1_6-no-2.cnf(28):  0.000s, unsat,    37 confl,     201 dec,     11 nd
  rutgers.edu/aim-200-1_6-no-3.cnf(29):  0.001s, unsat,    50 confl,     303 dec,     17 nd
  rutgers.edu/aim-200-1_6-no-4.cnf(30):  0.000s, unsat,    26 confl,     127 dec,      4 nd
rutgers.edu/aim-200-1_6-yes1-1.cnf(31):  0.000s, sat  ,    19 confl,     135 dec,      8 nd
rutgers.edu/aim-200-1_6-yes1-2.cnf(32):  0.001s, sat  ,    66 confl,     311 dec,     14 nd
rutgers.edu/aim-200-1_6-yes1-3.cnf(33):  0.000s, sat  ,    28 confl,     181 dec,     13 nd
rutgers.edu/aim-200-1_6-yes1-4.cnf(34):  0.000s, sat  ,    43 confl,     224 dec,     12 nd
  rutgers.edu/aim-200-2_0-no-1.cnf(35):  0.000s, unsat,    20 confl,     115 dec,      5 nd
  rutgers.edu/aim-200-2_0-no-2.cnf(36):  0.000s, unsat,    27 confl,     169 dec,      8 nd
  rutgers.edu/aim-200-2_0-no-3.cnf(37):  0.000s, unsat,    26 confl,     123 dec,      5 nd
  rutgers.edu/aim-200-2_0-no-4.cnf(38):  0.000s, unsat,    24 confl,     141 dec,      6 nd
rutgers.edu/aim-200-2_0-yes1-1.cnf(39):  0.001s, sat  ,   157 confl,     442 dec,      9 nd
rutgers.edu/aim-200-2_0-yes1-2.cnf(40):  0.001s, sat  ,    77 confl,     291 dec,     10 nd
rutgers.edu/aim-200-2_0-yes1-3.cnf(41):  0.000s, sat  ,    54 confl,     248 dec,     12 nd
rutgers.edu/aim-200-2_0-yes1-4.cnf(42):  0.001s, sat  ,    83 confl,     311 dec,      6 nd
rutgers.edu/aim-200-3_4-yes1-1.cnf(43):  0.001s, sat  ,    81 confl,     179 dec,      4 nd
rutgers.edu/aim-200-3_4-yes1-2.cnf(44):  0.001s, sat  ,    76 confl,     181 dec,      6 nd
rutgers.edu/aim-200-3_4-yes1-3.cnf(45):  0.010s, sat  ,  1142 confl,    1659 dec,      6 nd
rutgers.edu/aim-200-3_4-yes1-4.cnf(46):  0.000s, sat  ,    51 confl,     116 dec,      4 nd
rutgers.edu/aim-200-6_0-yes1-1.cnf(47):  0.000s, sat  ,    34 confl,      64 dec,      3 nd
rutgers.edu/aim-200-6_0-yes1-2.cnf(48):  0.004s, sat  ,   411 confl,     613 dec,      2 nd
rutgers.edu/aim-200-6_0-yes1-3.cnf(49):  0.000s, sat  ,    50 confl,      86 dec,      3 nd
rutgers.edu/aim-200-6_0-yes1-4.cnf(50):  0.000s, sat  ,    43 confl,      80 dec,      3 nd
    rutgers.edu/aim-50-1_6-no-1.cnf(51):  0.000s, unsat,    11 confl,      27 dec,      5 nd
    rutgers.edu/aim-50-1_6-no-2.cnf(52):  0.000s, unsat,    13 confl,      19 dec,      1 nd
    rutgers.edu/aim-50-1_6-no-3.cnf(53):  0.000s, unsat,    16 confl,      21 dec,      2 nd
    rutgers.edu/aim-50-1_6-no-4.cnf(54):  0.000s, unsat,     9 confl,      17 dec,      2 nd
 rutgers.edu/aim-50-1_6-yes1-1.cnf(55):  0.000s, sat  ,    14 confl,      21 dec,      4 nd
 rutgers.edu/aim-50-1_6-yes1-2.cnf(56):  0.000s, sat  ,     9 confl,      15 dec,      3 nd
 rutgers.edu/aim-50-1_6-yes1-3.cnf(57):  0.000s, sat  ,    10 confl,      15 dec,      2 nd
 rutgers.edu/aim-50-1_6-yes1-4.cnf(58):  0.000s, sat  ,     5 confl,       9 dec,      3 nd
   rutgers.edu/aim-50-2_0-no-1.cnf(59):  0.000s, unsat,     6 confl,      16 dec,      2 nd
```

```
  rutgers.edu/aim-50-2_0-no-2.cnf(60):  0.000s, unsat,    12 confl,      31 dec,      3 nd
  rutgers.edu/aim-50-2_0-no-3.cnf(61):  0.000s, unsat,    15 confl,      23 dec,      2 nd
  rutgers.edu/aim-50-2_0-no-4.cnf(62):  0.000s, unsat,    21 confl,      35 dec,      1 nd
rutgers.edu/aim-50-2_0-yes1-1.cnf(63):  0.000s, sat  ,    13 confl,      41 dec,      6 nd
rutgers.edu/aim-50-2_0-yes1-2.cnf(64):  0.000s, sat  ,    21 confl,      32 dec,      3 nd
rutgers.edu/aim-50-2_0-yes1-3.cnf(65):  0.000s, sat  ,    14 confl,      30 dec,      3 nd
rutgers.edu/aim-50-2_0-yes1-4.cnf(66):  0.000s, sat  ,    25 confl,      46 dec,      4 nd
rutgers.edu/aim-50-3_4-yes1-1.cnf(67):  0.000s, sat  ,    20 confl,      33 dec,      2 nd
rutgers.edu/aim-50-3_4-yes1-2.cnf(68):  0.000s, sat  ,    53 confl,      63 dec,      4 nd
rutgers.edu/aim-50-3_4-yes1-3.cnf(69):  0.000s, sat  ,    24 confl,      36 dec,      2 nd
rutgers.edu/aim-50-3_4-yes1-4.cnf(70):  0.000s, sat  ,    37 confl,      43 dec,      3 nd
rutgers.edu/aim-50-6_0-yes1-1.cnf(71):  0.000s, sat  ,    18 confl,      24 dec,      2 nd
rutgers.edu/aim-50-6_0-yes1-2.cnf(72):  0.000s, sat  ,    20 confl,      20 dec,      2 nd
rutgers.edu/aim-50-6_0-yes1-3.cnf(73):  0.000s, sat  ,    21 confl,      28 dec,      3 nd
rutgers.edu/aim-50-6_0-yes1-4.cnf(74):  0.000s, sat  ,    20 confl,      24 dec,      3 nd
       rutgers.edu/bf0432-007.cnf(75):  0.012s, unsat,   387 confl,     701 dec,      7 nd
       rutgers.edu/bf1355-075.cnf(76):  0.002s, unsat,    37 confl,      79 dec,      4 nd
       rutgers.edu/bf1355-638.cnf(77):  0.003s, unsat,    44 confl,     138 dec,      9 nd
       rutgers.edu/bf2670-001.cnf(78):  0.001s, unsat,    21 confl,      66 dec,      2 nd
        rutgers.edu/dubois100.cnf(79): expired
         rutgers.edu/dubois20.cnf(80):  0.000s, unsat,   115 confl,     324 dec,     13 nd
         rutgers.edu/dubois21.cnf(81):  0.000s, unsat,   150 confl,     390 dec,      5 nd
         rutgers.edu/dubois22.cnf(82):  0.000s, unsat,   142 confl,     426 dec,     15 nd
         rutgers.edu/dubois23.cnf(83):  0.000s, unsat,   153 confl,     453 dec,      7 nd
         rutgers.edu/dubois24.cnf(84):  0.000s, unsat,   203 confl,     565 dec,     15 nd
         rutgers.edu/dubois25.cnf(85):  0.001s, unsat,   221 confl,     552 dec,      7 nd
         rutgers.edu/dubois26.cnf(86):  0.001s, unsat,   201 confl,     635 dec,     18 nd
         rutgers.edu/dubois27.cnf(87):  0.001s, unsat,   247 confl,     601 dec,      8 nd
         rutgers.edu/dubois28.cnf(88):  0.001s, unsat,   184 confl,     681 dec,     18 nd
         rutgers.edu/dubois29.cnf(89):  0.001s, unsat,   216 confl,     608 dec,     11 nd
         rutgers.edu/dubois30.cnf(90):  0.001s, unsat,   208 confl,     787 dec,     22 nd
         rutgers.edu/dubois50.cnf(91):  0.002s, unsat,   463 confl,    2218 dec,     61 nd
            rutgers.edu/f1000.cnf(92): expired
            rutgers.edu/f2000.cnf(93): expired
             rutgers.edu/f600.cnf(94): expired
          rutgers.edu/g125.17.cnf(95): expired
          rutgers.edu/g125.18.cnf(96): expired
          rutgers.edu/g250.15.cnf(97): expired
          rutgers.edu/g250.29.cnf(98): expired
           rutgers.edu/hanoi4.cnf(99):  0.115s, sat  ,  3762 confl,    5876 dec,     27 nd
           rutgers.edu/hanoi5.cnf(100):  2.528s, sat  , 48256 confl,   76120 dec,    361 nd
           rutgers.edu/hole10.cnf(101): expired
            rutgers.edu/hole6.cnf(102):  0.005s, unsat,   907 confl,    1011 dec,      1 nd
            rutgers.edu/hole7.cnf(103):  0.158s, unsat,  9811 confl,   10571 dec,      1 nd
            rutgers.edu/hole8.cnf(104): 20.768s, unsat, 779658 confl,  823103 dec,      1 nd
            rutgers.edu/hole9.cnf(105): expired
           rutgers.edu/ii16a1.cnf(106): expired
           rutgers.edu/ii16a2.cnf(107): expired
           rutgers.edu/ii16b1.cnf(108): expired
           rutgers.edu/ii16b2.cnf(109):  0.034s, sat  ,  1200 confl,    1561 dec,     53 nd - solution counter overflow
           rutgers.edu/ii16c1.cnf(110): expired
           rutgers.edu/ii16c2.cnf(111):  0.056s, sat  ,  1923 confl,    2402 dec,     89 nd - solution counter overflow
           rutgers.edu/ii16d1.cnf(112): expired
           rutgers.edu/ii16d2.cnf(113):  0.086s, sat  ,  2622 confl,    3359 dec,    144 nd - solution counter overflow
           rutgers.edu/ii16e1.cnf(114): expired
           rutgers.edu/ii16e2.cnf(115):  0.064s, sat  ,  2196 confl,    2890 dec,     54 nd
           rutgers.edu/ii32a1.cnf(116):  0.043s, sat  ,  2160 confl,    2415 dec,     28 nd
           rutgers.edu/ii32b1.cnf(117): expired
           rutgers.edu/ii32b2.cnf(118):  0.333s, sat  ,  4702 confl,   76872 dec,  71866 nd
           rutgers.edu/ii32b3.cnf(119):  0.800s, sat  ,  3761 confl,  127692 dec, 123869 nd
           rutgers.edu/ii32b4.cnf(120):  0.069s, sat  ,  3048 confl,    3340 dec,     43 nd
           rutgers.edu/ii32c1.cnf(121): expired
           rutgers.edu/ii32c2.cnf(122): expired
           rutgers.edu/ii32c3.cnf(123): 13.176s, sat  , 10107 confl, 2992666 dec, 2982720 nd
           rutgers.edu/ii32c4.cnf(124):  0.147s, sat  ,  2346 confl,    2607 dec,     32 nd
           rutgers.edu/ii32d1.cnf(125): expired
           rutgers.edu/ii32d2.cnf(126): expired
           rutgers.edu/ii32d3.cnf(127): 10.924s, sat  , 189711 confl,  224330 dec,   1324 nd
           rutgers.edu/ii32e1.cnf(128): expired
           rutgers.edu/ii32e2.cnf(129): expired
           rutgers.edu/ii32e3.cnf(130):  1.254s, sat  ,  3177 confl,  215392 dec, 212226 nd
           rutgers.edu/ii32e4.cnf(131):  2.281s, sat  ,  4441 confl,  289750 dec, 285227 nd
           rutgers.edu/ii32e5.cnf(132):  3.504s, sat  ,  7575 confl,  391002 dec, 383536 nd
            rutgers.edu/ii8a1.cnf(133):  0.011s, sat  ,    24 confl,    8298 dec,   8277 nd
            rutgers.edu/ii8a2.cnf(134): expired
            rutgers.edu/ii8a3.cnf(135): expired
            rutgers.edu/ii8a4.cnf(136): expired
            rutgers.edu/ii8b1.cnf(137): expired
            rutgers.edu/ii8b2.cnf(138): expired
            rutgers.edu/ii8b3.cnf(139): expired
            rutgers.edu/ii8b4.cnf(140): expired
            rutgers.edu/ii8c1.cnf(141): expired
            rutgers.edu/ii8c2.cnf(142): expired
            rutgers.edu/ii8d1.cnf(143): expired
            rutgers.edu/ii8d2.cnf(144): expired
            rutgers.edu/ii8e1.cnf(145): expired
            rutgers.edu/ii8e2.cnf(146): expired
            rutgers.edu/jnh10.cnf(147):  0.000s, unsat,    19 confl,      18 dec,      1 nd
            rutgers.edu/jnh11.cnf(148):  0.001s, unsat,    53 confl,      69 dec,      1 nd
            rutgers.edu/jnh12.cnf(149):  0.000s, sat  ,    24 confl,      29 dec,      3 nd
            rutgers.edu/jnh13.cnf(150):  0.000s, unsat,     8 confl,      10 dec,      1 nd
            rutgers.edu/jnh14.cnf(151):  0.000s, unsat,    20 confl,      24 dec,      1 nd
            rutgers.edu/jnh15.cnf(152):  0.001s, unsat,    41 confl,      51 dec,      1 nd
            rutgers.edu/jnh16.cnf(153):  0.014s, unsat,   877 confl,    1086 dec,      1 nd
            rutgers.edu/jnh17.cnf(154):  0.001s, sat  ,    32 confl,      72 dec,     43 nd
            rutgers.edu/jnh18.cnf(155):  0.001s, unsat,    54 confl,      69 dec,      2 nd
            rutgers.edu/jnh19.cnf(156):  0.000s, unsat,    26 confl,      36 dec,      1 nd
             rutgers.edu/jnh1.cnf(157):  0.008s, sat  ,   426 confl,     761 dec,    308 nd
           rutgers.edu/jnh201.cnf(158):  1.180s, sat  ,  4602 confl,  217512 dec, 213837 nd
           rutgers.edu/jnh202.cnf(159):  0.000s, unsat,     9 confl,      10 dec,      1 nd
```

```
            rutgers.edu/jnh203.cnf(160):  0.001s, unsat,    48 confl,      54 dec,      1 nd
            rutgers.edu/jnh204.cnf(161):  0.011s, sat  ,   512 confl,     591 dec,     19 nd
            rutgers.edu/jnh205.cnf(162):  0.002s, sat  ,    87 confl,     132 dec,     17 nd
            rutgers.edu/jnh206.cnf(163):  0.001s, unsat,    83 confl,     106 dec,      1 nd
            rutgers.edu/jnh207.cnf(164):  0.002s, sat  ,   141 confl,     169 dec,      5 nd
            rutgers.edu/jnh208.cnf(165):  0.001s, unsat,    41 confl,      49 dec,      2 nd
            rutgers.edu/jnh209.cnf(166):  0.002s, sat  ,    88 confl,     186 dec,     86 nd
             rutgers.edu/jnh20.cnf(167):  0.000s, unsat,    17 confl,      24 dec,      2 nd
            rutgers.edu/jnh210.cnf(168):  0.012s, sat  ,    81 confl,    2548 dec,   2465 nd
            rutgers.edu/jnh211.cnf(169):  0.000s, unsat,     9 confl,      10 dec,      2 nd
            rutgers.edu/jnh212.cnf(170):  0.004s, sat  ,   250 confl,     309 dec,      3 nd
            rutgers.edu/jnh213.cnf(171):  0.001s, sat  ,    27 confl,      90 dec,     65 nd
            rutgers.edu/jnh214.cnf(172):  0.000s, unsat,    29 confl,      34 dec,      2 nd
            rutgers.edu/jnh215.cnf(173):  0.000s, unsat,    29 confl,      35 dec,      1 nd
            rutgers.edu/jnh216.cnf(174):  0.001s, unsat,    75 confl,      92 dec,      1 nd
            rutgers.edu/jnh217.cnf(175):  0.050s, sat  ,  1896 confl,    5959 dec,   3958 nd
            rutgers.edu/jnh218.cnf(176):  0.002s, sat  ,    81 confl,     315 dec,    238 nd
            rutgers.edu/jnh219.cnf(177):  0.001s, unsat,    80 confl,      93 dec,      1 nd
            rutgers.edu/jnh220.cnf(178):  0.006s, sat  ,   397 confl,     448 dec,     18 nd
              rutgers.edu/jnh2.cnf(179):  0.000s, unsat,     8 confl,       8 dec,      2 nd
            rutgers.edu/jnh301.cnf(180):  0.002s, sat  ,   102 confl,     148 dec,     22 nd
            rutgers.edu/jnh302.cnf(181):  0.000s, unsat,     5 confl,       5 dec,      1 nd
            rutgers.edu/jnh303.cnf(182):  0.001s, unsat,    39 confl,      49 dec,      1 nd
            rutgers.edu/jnh304.cnf(183):  0.000s, unsat,     5 confl,       4 dec,      1 nd
            rutgers.edu/jnh305.cnf(184):  0.000s, unsat,     8 confl,       7 dec,      1 nd
            rutgers.edu/jnh306.cnf(185):  0.003s, unsat,   178 confl,     213 dec,      1 nd
            rutgers.edu/jnh307.cnf(186):  0.000s, unsat,     6 confl,       7 dec,      2 nd
            rutgers.edu/jnh308.cnf(187):  0.001s, unsat,    45 confl,      57 dec,      1 nd
            rutgers.edu/jnh309.cnf(188):  0.000s, unsat,    10 confl,      12 dec,      1 nd
            rutgers.edu/jnh310.cnf(189):  0.000s, unsat,     5 confl,       5 dec,      1 nd
              rutgers.edu/jnh3.cnf(190):  0.002s, unsat,   123 confl,     155 dec,      2 nd
              rutgers.edu/jnh4.cnf(191):  0.000s, unsat,    30 confl,      32 dec,      1 nd
              rutgers.edu/jnh5.cnf(192):  0.000s, unsat,    17 confl,      19 dec,      1 nd
              rutgers.edu/jnh6.cnf(193):  0.001s, unsat,    66 confl,      81 dec,      2 nd
              rutgers.edu/jnh7.cnf(194):  0.003s, sat  ,    43 confl,     452 dec,    400 nd
              rutgers.edu/jnh8.cnf(195):  0.000s, unsat,    12 confl,      17 dec,      1 nd
              rutgers.edu/jnh9.cnf(196):  0.000s, unsat,    17 confl,      22 dec,      3 nd
          rutgers.edu/par16-1-c.cnf(197):  0.484s, sat  , 14339 confl,   16091 dec,      2 nd
            rutgers.edu/par16-1.cnf(198):  1.073s, sat  , 24997 confl,   29811 dec,    100 nd
          rutgers.edu/par16-2-c.cnf(199):  0.617s, sat  , 18943 confl,   21613 dec,      2 nd
            rutgers.edu/par16-2.cnf(200):  1.238s, sat  , 25574 confl,   29233 dec,     77 nd
          rutgers.edu/par16-3-c.cnf(201):  0.534s, sat  , 15855 confl,   18234 dec,      6 nd
            rutgers.edu/par16-3.cnf(202):  3.375s, unsat, 58802 confl,   68491 dec,     24 nd
          rutgers.edu/par16-4-c.cnf(203):  0.230s, sat  ,  8316 confl,    9707 dec,      2 nd
            rutgers.edu/par16-4.cnf(204):  0.665s, sat  , 15097 confl,   18122 dec,     30 nd
          rutgers.edu/par16-5-c.cnf(205):  0.777s, sat  , 22594 confl,   25899 dec,      2 nd
            rutgers.edu/par16-5.cnf(206):  1.062s, sat  , 23318 confl,   27704 dec,     91 nd
          rutgers.edu/par32-1-c.cnf(207): expired
            rutgers.edu/par32-1.cnf(208): expired
          rutgers.edu/par32-2-c.cnf(209): expired
            rutgers.edu/par32-2.cnf(210): expired
          rutgers.edu/par32-3-c.cnf(211): expired
            rutgers.edu/par32-3.cnf(212): expired
          rutgers.edu/par32-4-c.cnf(213): expired
            rutgers.edu/par32-4.cnf(214): expired
          rutgers.edu/par32-5-c.cnf(215): expired
            rutgers.edu/par32-5.cnf(216): expired
           rutgers.edu/par8-1-c.cnf(217):  0.000s, sat  ,    18 confl,      18 dec,      2 nd
             rutgers.edu/par8-1.cnf(218):  0.001s, sat  ,   115 confl,     150 dec,     10 nd
           rutgers.edu/par8-2-c.cnf(219):  0.000s, sat  ,    15 confl,      15 dec,      2 nd
             rutgers.edu/par8-2.cnf(220):  0.001s, sat  ,   119 confl,     172 dec,     14 nd
           rutgers.edu/par8-3-c.cnf(221):  0.000s, sat  ,    30 confl,      31 dec,      2 nd
             rutgers.edu/par8-3.cnf(222):  0.002s, sat  ,   205 confl,     299 dec,     15 nd
           rutgers.edu/par8-4-c.cnf(223):  0.000s, sat  ,    16 confl,      16 dec,      2 nd
             rutgers.edu/par8-4.cnf(224):  0.005s, sat  ,   416 confl,     573 dec,     13 nd
           rutgers.edu/par8-5-c.cnf(225):  0.000s, sat  ,    14 confl,      14 dec,      2 nd
             rutgers.edu/par8-5.cnf(226):  0.005s, sat  ,   402 confl,     559 dec,      9 nd
         rutgers.edu/pret150_25.cnf(227):  0.002s, unsat,   401 confl,    2101 dec,     60 nd
         rutgers.edu/pret150_40.cnf(228):  0.003s, unsat,   442 confl,    2347 dec,     60 nd
         rutgers.edu/pret150_60.cnf(229):  0.003s, unsat,   406 confl,    2283 dec,     63 nd
         rutgers.edu/pret150_75.cnf(230):  0.003s, unsat,   437 confl,    2344 dec,     65 nd
          rutgers.edu/pret60_25.cnf(231):  0.001s, unsat,   197 confl,     470 dec,     12 nd
          rutgers.edu/pret60_40.cnf(232):  0.000s, unsat,   168 confl,     444 dec,     11 nd
          rutgers.edu/pret60_60.cnf(233):  0.001s, unsat,   214 confl,     524 dec,     12 nd
          rutgers.edu/pret60_75.cnf(234):  0.001s, unsat,   190 confl,     449 dec,     11 nd
        rutgers.edu/ssa0432-003.cnf(235):  0.000s, unsat,    40 confl,      85 dec,      4 nd
        rutgers.edu/ssa2670-130.cnf(236):  0.003s, unsat,   126 confl,     258 dec,      4 nd
        rutgers.edu/ssa2670-141.cnf(237):  0.004s, unsat,   274 confl,     439 dec,      4 nd
        rutgers.edu/ssa6288-047.cnf(238):  0.002s, unsat,     2 confl,      21 dec,      1 nd
        rutgers.edu/ssa7552-038.cnf(239): expired
        rutgers.edu/ssa7552-158.cnf(240): expired
        rutgers.edu/ssa7552-159.cnf(241): expired
        rutgers.edu/ssa7552-160.cnf(242): expired
192 solved, 50 unsolved
time for all: 68 sec 893 msec, 1356419 conflicts
```

## 4.2 rutgers.edu Clasp

```
   rutgers.edu/samename/jnh8.cnf(0): 0.001s
   rutgers.edu/samename/par16-1.cnf(1): 0.053s
   rutgers.edu/aim-100-1_6-no-1.cnf(2): 0.000s
   rutgers.edu/aim-100-1_6-no-2.cnf(3): 0.000s
   rutgers.edu/aim-100-1_6-no-3.cnf(4): 0.000s
   rutgers.edu/aim-100-1_6-no-4.cnf(5): 0.000s
 rutgers.edu/aim-100-1_6-yes1-1.cnf(6): 0.000s
 rutgers.edu/aim-100-1_6-yes1-2.cnf(7): 0.000s
 rutgers.edu/aim-100-1_6-yes1-3.cnf(8): 0.000s
 rutgers.edu/aim-100-1_6-yes1-4.cnf(9): 0.000s
   rutgers.edu/aim-100-2_0-no-1.cnf(10): 0.000s
   rutgers.edu/aim-100-2_0-no-2.cnf(11): 0.000s
   rutgers.edu/aim-100-2_0-no-3.cnf(12): 0.000s
   rutgers.edu/aim-100-2_0-no-4.cnf(13): 0.000s
 rutgers.edu/aim-100-2_0-yes1-1.cnf(14): 0.000s
 rutgers.edu/aim-100-2_0-yes1-2.cnf(15): 0.000s
```

```
rutgers.edu/aim-100-2_0-yes1-3.cnf(16): 0.000s
rutgers.edu/aim-100-2_0-yes1-4.cnf(17): 0.000s
rutgers.edu/aim-100-3_4-yes1-1.cnf(18): 0.000s
rutgers.edu/aim-100-3_4-yes1-2.cnf(19): 0.000s
rutgers.edu/aim-100-3_4-yes1-3.cnf(20): 0.000s
rutgers.edu/aim-100-3_4-yes1-4.cnf(21): 0.000s
rutgers.edu/aim-100-6_0-yes1-1.cnf(22): 0.001s
rutgers.edu/aim-100-6_0-yes1-2.cnf(23): 0.001s
rutgers.edu/aim-100-6_0-yes1-3.cnf(24): 0.000s
rutgers.edu/aim-100-6_0-yes1-4.cnf(25): 0.000s
  rutgers.edu/aim-200-1_6-no-1.cnf(26): 0.000s
  rutgers.edu/aim-200-1_6-no-2.cnf(27): 0.000s
  rutgers.edu/aim-200-1_6-no-3.cnf(28): 0.000s
  rutgers.edu/aim-200-1_6-no-4.cnf(29): 0.000s
rutgers.edu/aim-200-1_6-yes1-1.cnf(30): 0.000s
rutgers.edu/aim-200-1_6-yes1-2.cnf(31): 0.000s
rutgers.edu/aim-200-1_6-yes1-3.cnf(32): 0.000s
rutgers.edu/aim-200-1_6-yes1-4.cnf(33): 0.000s
  rutgers.edu/aim-200-2_0-no-1.cnf(34): 0.000s
  rutgers.edu/aim-200-2_0-no-2.cnf(35): 0.000s
  rutgers.edu/aim-200-2_0-no-3.cnf(36): 0.000s
  rutgers.edu/aim-200-2_0-no-4.cnf(37): 0.000s
rutgers.edu/aim-200-2_0-yes1-1.cnf(38): 0.001s
rutgers.edu/aim-200-2_0-yes1-2.cnf(39): 0.000s
rutgers.edu/aim-200-2_0-yes1-3.cnf(40): 0.001s
rutgers.edu/aim-200-2_0-yes1-4.cnf(41): 0.001s
rutgers.edu/aim-200-3_4-yes1-1.cnf(42): 0.001s
rutgers.edu/aim-200-3_4-yes1-2.cnf(43): 0.001s
rutgers.edu/aim-200-3_4-yes1-3.cnf(44): 0.001s
rutgers.edu/aim-200-3_4-yes1-4.cnf(45): 0.001s
rutgers.edu/aim-200-6_0-yes1-1.cnf(46): 0.001s
rutgers.edu/aim-200-6_0-yes1-2.cnf(47): 0.001s
rutgers.edu/aim-200-6_0-yes1-3.cnf(48): 0.001s
rutgers.edu/aim-200-6_0-yes1-4.cnf(49): 0.001s
   rutgers.edu/aim-50-1_6-no-1.cnf(50): 0.000s
   rutgers.edu/aim-50-1_6-no-2.cnf(51): 0.000s
   rutgers.edu/aim-50-1_6-no-3.cnf(52): 0.000s
   rutgers.edu/aim-50-1_6-no-4.cnf(53): 0.000s
 rutgers.edu/aim-50-1_6-yes1-1.cnf(54): 0.000s
 rutgers.edu/aim-50-1_6-yes1-2.cnf(55): 0.000s
 rutgers.edu/aim-50-1_6-yes1-3.cnf(56): 0.000s
 rutgers.edu/aim-50-1_6-yes1-4.cnf(57): 0.000s
  rutgers.edu/aim-50-2_0-no-1.cnf(58): 0.000s
  rutgers.edu/aim-50-2_0-no-2.cnf(59): 0.000s
  rutgers.edu/aim-50-2_0-no-3.cnf(60): 0.000s
  rutgers.edu/aim-50-2_0-no-4.cnf(61): 0.000s
 rutgers.edu/aim-50-2_0-yes1-1.cnf(62): 0.000s
 rutgers.edu/aim-50-2_0-yes1-2.cnf(63): 0.000s
 rutgers.edu/aim-50-2_0-yes1-3.cnf(64): 0.000s
 rutgers.edu/aim-50-2_0-yes1-4.cnf(65): 0.000s
 rutgers.edu/aim-50-3_4-yes1-1.cnf(66): 0.000s
 rutgers.edu/aim-50-3_4-yes1-2.cnf(67): 0.000s
 rutgers.edu/aim-50-3_4-yes1-3.cnf(68): 0.000s
 rutgers.edu/aim-50-3_4-yes1-4.cnf(69): 0.000s
 rutgers.edu/aim-50-6_0-yes1-1.cnf(70): 0.000s
 rutgers.edu/aim-50-6_0-yes1-2.cnf(71): 0.000s
 rutgers.edu/aim-50-6_0-yes1-3.cnf(72): 0.000s
 rutgers.edu/aim-50-6_0-yes1-4.cnf(73): 0.000s
        rutgers.edu/bf0432-007.cnf(74): 0.005s
        rutgers.edu/bf1355-075.cnf(75): 0.006s
        rutgers.edu/bf1355-638.cnf(76): 0.006s
        rutgers.edu/bf2670-001.cnf(77): 0.003s
         rutgers.edu/dubois100.cnf(78): expired 60.000s
          rutgers.edu/dubois20.cnf(79): 0.000s
          rutgers.edu/dubois21.cnf(80): 0.000s
          rutgers.edu/dubois22.cnf(81): 0.000s
          rutgers.edu/dubois23.cnf(82): 0.000s
          rutgers.edu/dubois24.cnf(83): 0.000s
          rutgers.edu/dubois25.cnf(84): 0.000s
          rutgers.edu/dubois26.cnf(85): 0.000s
          rutgers.edu/dubois27.cnf(86): 0.000s
          rutgers.edu/dubois28.cnf(87): 0.000s
          rutgers.edu/dubois29.cnf(88): 0.000s
          rutgers.edu/dubois30.cnf(89): 0.000s
          rutgers.edu/dubois50.cnf(90): 0.001s
             rutgers.edu/f1000.cnf(91): expired 60.000s
             rutgers.edu/f2000.cnf(92): expired 60.000s
              rutgers.edu/f600.cnf(93): expired 60.000s
           rutgers.edu/g125.17.cnf(94): expired 60.000s
           rutgers.edu/g125.18.cnf(95): expired 60.000s
           rutgers.edu/g250.15.cnf(96): expired 60.000s
           rutgers.edu/g250.29.cnf(97): expired 60.000s
            rutgers.edu/hanoi4.cnf(98): 0.061s
            rutgers.edu/hanoi5.cnf(99): 0.444s
            rutgers.edu/hole10.cnf(100): expired 60.000s
             rutgers.edu/hole6.cnf(101): 0.002s
             rutgers.edu/hole7.cnf(102): 0.013s
             rutgers.edu/hole8.cnf(103): 0.125s
             rutgers.edu/hole9.cnf(104): 2.094s
            rutgers.edu/ii16a1.cnf(105): expired 60.000s
            rutgers.edu/ii16a2.cnf(106): expired 60.000s
            rutgers.edu/ii16b1.cnf(107): expired 60.000s
            rutgers.edu/ii16b2.cnf(108): expired 60.000s
            rutgers.edu/ii16c1.cnf(109): expired 60.000s
            rutgers.edu/ii16c2.cnf(110): expired 60.000s
            rutgers.edu/ii16d1.cnf(111): expired 60.000s
            rutgers.edu/ii16d2.cnf(112): expired 60.000s
            rutgers.edu/ii16e1.cnf(113): expired 60.000s
            rutgers.edu/ii16e2.cnf(114): 54.397s
            rutgers.edu/ii32a1.cnf(115): expired 60.000s
            rutgers.edu/ii32b1.cnf(116): expired 60.000s
            rutgers.edu/ii32b2.cnf(117): 35.000s
            rutgers.edu/ii32b3.cnf(118): expired 60.000s
            rutgers.edu/ii32b4.cnf(119): 0.065s
            rutgers.edu/ii32c1.cnf(120): expired 60.000s
            rutgers.edu/ii32c2.cnf(121): expired 60.000s
            rutgers.edu/ii32c3.cnf(122): expired 60.000s
            rutgers.edu/ii32c4.cnf(123): expired 60.000s
            rutgers.edu/ii32d1.cnf(124): expired 60.000s
            rutgers.edu/ii32d2.cnf(125): expired 60.000s
            rutgers.edu/ii32d3.cnf(126): expired 60.000s
            rutgers.edu/ii32e1.cnf(127): expired 60.000s
            rutgers.edu/ii32e2.cnf(128): expired 60.000s
            rutgers.edu/ii32e3.cnf(129): expired 60.000s
            rutgers.edu/ii32e4.cnf(130): expired 60.000s
            rutgers.edu/ii32e5.cnf(131): expired 60.000s
             rutgers.edu/ii8a1.cnf(132): 0.588s
             rutgers.edu/ii8a2.cnf(133): expired 60.000s
             rutgers.edu/ii8a3.cnf(134): expired 60.000s
             rutgers.edu/ii8a4.cnf(135): expired 60.000s
             rutgers.edu/ii8b1.cnf(136): expired 60.000s
             rutgers.edu/ii8b2.cnf(137): expired 60.000s
             rutgers.edu/ii8b3.cnf(138): expired 60.000s
             rutgers.edu/ii8b4.cnf(139): expired 60.000s
             rutgers.edu/ii8c1.cnf(140): expired 60.000s
             rutgers.edu/ii8c2.cnf(141): expired 60.000s
             rutgers.edu/ii8d1.cnf(142): expired 60.000s
             rutgers.edu/ii8d2.cnf(143): expired 60.000s
             rutgers.edu/ii8e1.cnf(144): expired 60.000s
             rutgers.edu/ii8e2.cnf(145): expired 60.000s
             rutgers.edu/jnh10.cnf(146): 0.001s
             rutgers.edu/jnh11.cnf(147): 0.001s
             rutgers.edu/jnh12.cnf(148): 0.001s
             rutgers.edu/jnh13.cnf(149): 0.001s
             rutgers.edu/jnh14.cnf(150): 0.001s
             rutgers.edu/jnh15.cnf(151): 0.001s
             rutgers.edu/jnh16.cnf(152): 0.002s
             rutgers.edu/jnh17.cnf(153): 0.001s
             rutgers.edu/jnh18.cnf(154): 0.001s
             rutgers.edu/jnh19.cnf(155): 0.001s
              rutgers.edu/jnh1.cnf(156): 0.007s
            rutgers.edu/jnh201.cnf(157): 27.329s
            rutgers.edu/jnh202.cnf(158): 0.001s
            rutgers.edu/jnh203.cnf(159): 0.001s
            rutgers.edu/jnh204.cnf(160): 0.002s
            rutgers.edu/jnh205.cnf(161): 0.001s
            rutgers.edu/jnh206.cnf(162): 0.001s
            rutgers.edu/jnh207.cnf(163): 0.001s
            rutgers.edu/jnh208.cnf(164): 0.001s
            rutgers.edu/jnh209.cnf(165): 0.004s
             rutgers.edu/jnh20.cnf(166): 0.001s
            rutgers.edu/jnh210.cnf(167): 0.264s
            rutgers.edu/jnh211.cnf(168): 0.001s
            rutgers.edu/jnh212.cnf(169): 0.001s
            rutgers.edu/jnh213.cnf(170): 0.003s
            rutgers.edu/jnh214.cnf(171): 0.001s
            rutgers.edu/jnh215.cnf(172): 0.001s
            rutgers.edu/jnh216.cnf(173): 0.001s
            rutgers.edu/jnh217.cnf(174): 0.125s
            rutgers.edu/jnh218.cnf(175): 0.007s
            rutgers.edu/jnh219.cnf(176): 0.001s
            rutgers.edu/jnh220.cnf(177): 0.002s
              rutgers.edu/jnh2.cnf(178): 0.001s
            rutgers.edu/jnh301.cnf(179): 0.002s
            rutgers.edu/jnh302.cnf(180): 0.001s
            rutgers.edu/jnh303.cnf(181): 0.001s
            rutgers.edu/jnh304.cnf(182): 0.001s
            rutgers.edu/jnh305.cnf(183): 0.001s
            rutgers.edu/jnh306.cnf(184): 0.001s
            rutgers.edu/jnh307.cnf(185): 0.001s
            rutgers.edu/jnh308.cnf(186): 0.001s
            rutgers.edu/jnh309.cnf(187): 0.001s
            rutgers.edu/jnh310.cnf(188): 0.001s
              rutgers.edu/jnh3.cnf(189): 0.001s
              rutgers.edu/jnh4.cnf(190): 0.001s
              rutgers.edu/jnh5.cnf(191): 0.001s
              rutgers.edu/jnh6.cnf(192): 0.001s
              rutgers.edu/jnh7.cnf(193): 0.034s
              rutgers.edu/jnh8.cnf(194): 0.001s
              rutgers.edu/jnh9.cnf(195): 0.001s
       rutgers.edu/par16-1-c.cnf(196): 0.073s
        rutgers.edu/par16-1.cnf(197): 0.052s
       rutgers.edu/par16-2-c.cnf(198): 0.080s
        rutgers.edu/par16-2.cnf(199): 0.086s
       rutgers.edu/par16-3-c.cnf(200): 0.099s
        rutgers.edu/par16-3.cnf(201): 0.076s
       rutgers.edu/par16-4-c.cnf(202): 0.029s
        rutgers.edu/par16-4.cnf(203): 0.037s
       rutgers.edu/par16-5-c.cnf(204): 0.055s
        rutgers.edu/par16-5.cnf(205): 0.051s
       rutgers.edu/par32-1-c.cnf(206): expired 60.000s
        rutgers.edu/par32-1.cnf(207): expired 60.000s
       rutgers.edu/par32-2-c.cnf(208): expired 60.000s
        rutgers.edu/par32-2.cnf(209): expired 60.000s
       rutgers.edu/par32-3-c.cnf(210): expired 60.000s
        rutgers.edu/par32-3.cnf(211): expired 60.000s
       rutgers.edu/par32-4-c.cnf(212): expired 60.000s
        rutgers.edu/par32-4.cnf(213): expired 60.000s
       rutgers.edu/par32-5-c.cnf(214): expired 60.000s
        rutgers.edu/par32-5.cnf(215): expired 60.000s
```

```
          rutgers.edu/par8-1-c.cnf(216): 0.000s
            rutgers.edu/par8-1.cnf(217): 0.001s
          rutgers.edu/par8-2-c.cnf(218): 0.000s
            rutgers.edu/par8-2.cnf(219): 0.001s
          rutgers.edu/par8-3-c.cnf(220): 0.000s
            rutgers.edu/par8-3.cnf(221): 0.001s
          rutgers.edu/par8-4-c.cnf(222): 0.000s
            rutgers.edu/par8-4.cnf(223): 0.001s
          rutgers.edu/par8-5-c.cnf(224): 0.000s
            rutgers.edu/par8-5.cnf(225): 0.001s
        rutgers.edu/pret150_25.cnf(226): 0.000s
        rutgers.edu/pret150_40.cnf(227): 0.000s
        rutgers.edu/pret150_60.cnf(228): 0.000s
         rutgers.edu/pret150_75.cnf(229): 0.000s
          rutgers.edu/pret60_25.cnf(230): 0.000s
          rutgers.edu/pret60_40.cnf(231): 0.000s
          rutgers.edu/pret60_60.cnf(232): 0.000s
          rutgers.edu/pret60_75.cnf(233): 0.000s
        rutgers.edu/ssa0432-003.cnf(234): 0.001s
        rutgers.edu/ssa2670-130.cnf(235): 0.003s
        rutgers.edu/ssa2670-141.cnf(236): 0.002s
        rutgers.edu/ssa6288-047.cnf(237): 0.010s
        rutgers.edu/ssa7552-038.cnf(238): expired 60.000s
        rutgers.edu/ssa7552-158.cnf(239): expired 60.000s
        rutgers.edu/ssa7552-159.cnf(240): expired 60.000s
        rutgers.edu/ssa7552-160.cnf(241): expired 60.000s
```

## 4.3 rutgers.edu sharpSAT

```
     rutgers.edu/samename/jnh8.cnf(0): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/samename/par16-1.cnf(1): 0.218, sat  ,  1480 confl,    1480 dec
  rutgers.edu/aim-100-1_6-no-1.cnf(2): 0.002, unsat,    21 confl,      21 dec
  rutgers.edu/aim-100-1_6-no-2.cnf(3): 0.002, unsat,     9 confl,       8 dec
  rutgers.edu/aim-100-1_6-no-3.cnf(4): 0.002, unsat,    40 confl,      39 dec
  rutgers.edu/aim-100-1_6-no-4.cnf(5): 0.003, unsat,     9 confl,       8 dec
rutgers.edu/aim-100-1_6-yes1-1.cnf(6): 0.002, sat  ,    24 confl,      24 dec
rutgers.edu/aim-100-1_6-yes1-2.cnf(7): 0.002, sat  ,     4 confl,       4 dec
rutgers.edu/aim-100-1_6-yes1-3.cnf(8): 0.002, sat  ,     1 confl,       1 dec
rutgers.edu/aim-100-1_6-yes1-4.cnf(9): 0.002, sat  ,     9 confl,       9 dec
  rutgers.edu/aim-100-2_0-no-1.cnf(10): 0.003, unsat,    21 confl,      20 dec
  rutgers.edu/aim-100-2_0-no-2.cnf(11): 0.003, unsat,    16 confl,      15 dec
  rutgers.edu/aim-100-2_0-no-3.cnf(12): 0.002, unsat,    21 confl,      20 dec
  rutgers.edu/aim-100-2_0-no-4.cnf(13): 0.003, unsat,    28 confl,      27 dec
rutgers.edu/aim-100-2_0-yes1-1.cnf(14): 0.002, sat  ,    10 confl,      10 dec
rutgers.edu/aim-100-2_0-yes1-2.cnf(15): 0.003, sat  ,    10 confl,      10 dec
rutgers.edu/aim-100-2_0-yes1-3.cnf(16): 0.003, sat  ,    16 confl,      16 dec
rutgers.edu/aim-100-2_0-yes1-4.cnf(17): 0.002, sat  ,     1 confl,       1 dec
rutgers.edu/aim-100-3_4-yes1-1.cnf(18): 0.002, sat  ,     7 confl,       7 dec
rutgers.edu/aim-100-3_4-yes1-2.cnf(19): 0.003, sat  ,     7 confl,       7 dec
rutgers.edu/aim-100-3_4-yes1-3.cnf(20): 0.003, sat  ,     5 confl,       5 dec
rutgers.edu/aim-100-3_4-yes1-4.cnf(21): 0.003, sat  ,    13 confl,      13 dec
rutgers.edu/aim-100-6_0-yes1-1.cnf(22): 0.002, sat  ,     4 confl,       4 dec
rutgers.edu/aim-100-6_0-yes1-2.cnf(23): 0.003, sat  ,     3 confl,       3 dec
rutgers.edu/aim-100-6_0-yes1-3.cnf(24): 0.003, sat  ,     3 confl,       3 dec
rutgers.edu/aim-100-6_0-yes1-4.cnf(25): 0.003, sat  ,     2 confl,       2 dec
  rutgers.edu/aim-200-1_6-no-1.cnf(26): 0.002, unsat,     8 confl,       7 dec
  rutgers.edu/aim-200-1_6-no-2.cnf(27): 0.003, unsat,    71 confl,      70 dec
  rutgers.edu/aim-200-1_6-no-3.cnf(28): 0.003, unsat,    76 confl,      75 dec
  rutgers.edu/aim-200-1_6-no-4.cnf(29): 0.003, unsat,    48 confl,      47 dec
rutgers.edu/aim-200-1_6-yes1-1.cnf(30): 0.003, sat  ,     1 confl,       1 dec
rutgers.edu/aim-200-1_6-yes1-2.cnf(31): 0.002, sat  ,     1 confl,       1 dec
rutgers.edu/aim-200-1_6-yes1-3.cnf(32): 0.003, sat  ,     5 confl,       5 dec
rutgers.edu/aim-200-1_6-yes1-4.cnf(33): 0.003, sat  ,     7 confl,       7 dec
  rutgers.edu/aim-200-2_0-no-1.cnf(34): 0.003, unsat,    53 confl,      52 dec
  rutgers.edu/aim-200-2_0-no-2.cnf(35): 0.004, unsat,    50 confl,      49 dec
  rutgers.edu/aim-200-2_0-no-3.cnf(36): 0.004, unsat,    54 confl,      53 dec
  rutgers.edu/aim-200-2_0-no-4.cnf(37): 0.004, unsat,    48 confl,      47 dec
rutgers.edu/aim-200-2_0-yes1-1.cnf(38): 0.004, sat  ,    66 confl,      66 dec
rutgers.edu/aim-200-2_0-yes1-2.cnf(39): 0.003, sat  ,    27 confl,      27 dec
rutgers.edu/aim-200-2_0-yes1-3.cnf(40): 0.002, sat  ,     1 confl,       1 dec
rutgers.edu/aim-200-2_0-yes1-4.cnf(41): 0.004, sat  ,    92 confl,      92 dec
rutgers.edu/aim-200-3_4-yes1-1.cnf(42): 0.004, sat  ,    22 confl,      22 dec
rutgers.edu/aim-200-3_4-yes1-2.cnf(43): 0.004, sat  ,    10 confl,      10 dec
rutgers.edu/aim-200-3_4-yes1-3.cnf(44): 0.008, sat  ,   101 confl,     101 dec
rutgers.edu/aim-200-3_4-yes1-4.cnf(45): 0.004, sat  ,    19 confl,      19 dec
rutgers.edu/aim-200-6_0-yes1-1.cnf(46): 0.004, sat  ,     4 confl,       4 dec
rutgers.edu/aim-200-6_0-yes1-2.cnf(47): 0.006, sat  ,    32 confl,      32 dec
rutgers.edu/aim-200-6_0-yes1-3.cnf(48): 0.005, sat  ,    14 confl,      14 dec
rutgers.edu/aim-200-6_0-yes1-4.cnf(49): 0.009, sat  ,    65 confl,      65 dec
   rutgers.edu/aim-50-1_6-no-1.cnf(50): 0.003, unsat,     5 confl,       4 dec
   rutgers.edu/aim-50-1_6-no-2.cnf(51): 0.002, unsat,     7 confl,       6 dec
   rutgers.edu/aim-50-1_6-no-3.cnf(52): 0.002, unsat,     5 confl,       4 dec
   rutgers.edu/aim-50-1_6-no-4.cnf(53): 0.002, unsat,     4 confl,       3 dec
 rutgers.edu/aim-50-1_6-yes1-1.cnf(54): 0.003, sat  ,     1 confl,       1 dec
 rutgers.edu/aim-50-1_6-yes1-2.cnf(55): 0.003, sat  ,     5 confl,       5 dec
 rutgers.edu/aim-50-1_6-yes1-3.cnf(56): 0.003, sat  ,     1 confl,       1 dec
 rutgers.edu/aim-50-1_6-yes1-4.cnf(57): 0.003, sat  ,     2 confl,       2 dec
  rutgers.edu/aim-50-2_0-no-1.cnf(58): 0.003, unsat,     6 confl,       5 dec
  rutgers.edu/aim-50-2_0-no-2.cnf(59): 0.003, unsat,    13 confl,      12 dec
  rutgers.edu/aim-50-2_0-no-3.cnf(60): 0.003, unsat,     8 confl,       7 dec
  rutgers.edu/aim-50-2_0-no-4.cnf(61): 0.003, unsat,     5 confl,       4 dec
 rutgers.edu/aim-50-2_0-yes1-1.cnf(62): 0.003, sat  ,     6 confl,       6 dec
 rutgers.edu/aim-50-2_0-yes1-2.cnf(63): 0.003, sat  ,     4 confl,       4 dec
 rutgers.edu/aim-50-2_0-yes1-3.cnf(64): 0.003, sat  ,     9 confl,       9 dec
 rutgers.edu/aim-50-2_0-yes1-4.cnf(65): 0.002, sat  ,     5 confl,       5 dec
 rutgers.edu/aim-50-3_4-yes1-1.cnf(66): 0.003, sat  ,     4 confl,       4 dec
 rutgers.edu/aim-50-3_4-yes1-2.cnf(67): 0.003, sat  ,     2 confl,       2 dec
 rutgers.edu/aim-50-3_4-yes1-3.cnf(68): 0.003, sat  ,     4 confl,       4 dec
 rutgers.edu/aim-50-3_4-yes1-4.cnf(69): 0.003, sat  ,     4 confl,       4 dec
 rutgers.edu/aim-50-6_0-yes1-1.cnf(70): 0.003, sat  ,     2 confl,       2 dec
 rutgers.edu/aim-50-6_0-yes1-2.cnf(71): 0.002, sat  ,     1 confl,       1 dec
 rutgers.edu/aim-50-6_0-yes1-3.cnf(72): 0.003, sat  ,     0 confl,       0 dec
 rutgers.edu/aim-50-6_0-yes1-4.cnf(73): 0.003, sat  ,     1 confl,       1 dec
        rutgers.edu/bf0432-007.cnf(74): 0.023, unsat,   152 confl,     392 dec
        rutgers.edu/bf1355-075.cnf(75): 0.029, unsat,    68 confl,      71 dec
        rutgers.edu/bf1355-638.cnf(76): 0.022, unsat,    56 confl,      57 dec
        rutgers.edu/bf2670-001.cnf(77): 0.007, unsat,    19 confl,      18 dec
         rutgers.edu/dubois100.cnf(78): 0.004, sat  ,     0 confl,     295 dec
          rutgers.edu/dubois20.cnf(79): 0.004, unsat,   183 confl,     410 dec
          rutgers.edu/dubois21.cnf(80): 0.005, unsat,   260 confl,     539 dec
          rutgers.edu/dubois22.cnf(81): 0.004, unsat,   251 confl,     538 dec
```

```
rutgers.edu/dubois23.cnf(82): 0.006, unsat,   363 confl,     780 dec
rutgers.edu/dubois24.cnf(83): 0.005, unsat,   270 confl,     781 dec
rutgers.edu/dubois25.cnf(84): 0.006, unsat,   365 confl,     700 dec
rutgers.edu/dubois26.cnf(85): 0.005, unsat,   267 confl,     712 dec
rutgers.edu/dubois27.cnf(86): 0.007, unsat,   552 confl,    1029 dec
rutgers.edu/dubois28.cnf(87): 0.005, unsat,   365 confl,     813 dec
rutgers.edu/dubois29.cnf(88): 0.010, unsat,   659 confl,    1583 dec
rutgers.edu/dubois30.cnf(89): 0.006, unsat,   370 confl,     927 dec
rutgers.edu/dubois50.cnf(90): 0.014, unsat,   971 confl,    2540 dec
   rutgers.edu/f1000.cnf(91): timeout
   rutgers.edu/f2000.cnf(92): timeout
    rutgers.edu/f600.cnf(93): timeout
 rutgers.edu/g125.17.cnf(94): timeout
 rutgers.edu/g125.18.cnf(95): timeout
 rutgers.edu/g250.15.cnf(96): timeout
 rutgers.edu/g250.29.cnf(97): timeout
  rutgers.edu/hanoi4.cnf(98): 0.624, sat  ,  3644 confl,    3644 dec
  rutgers.edu/hanoi5.cnf(99): timeout
  rutgers.edu/hole10.cnf(100): 5.500, unsat, 90001 confl,   90003 dec
   rutgers.edu/hole6.cnf(101): 0.004, unsat,   309 confl,     308 dec
   rutgers.edu/hole7.cnf(102): 0.019, unsat,  1362 confl,    1361 dec
   rutgers.edu/hole8.cnf(103): 0.150, unsat,  5732 confl,    5731 dec
   rutgers.edu/hole9.cnf(104): 1.046, unsat, 22370 confl,   22369 dec
  rutgers.edu/ii16a1.cnf(105): timeout
  rutgers.edu/ii16a2.cnf(106): timeout
  rutgers.edu/ii16b1.cnf(107): timeout
  rutgers.edu/ii16b2.cnf(108): 0.110, sat  ,   676 confl,     972 dec
  rutgers.edu/ii16c1.cnf(109): timeout
  rutgers.edu/ii16c2.cnf(110): 0.219, sat  ,  1480 confl,    2129 dec
  rutgers.edu/ii16d1.cnf(111): timeout
  rutgers.edu/ii16d2.cnf(112): 0.247, sat  ,  1355 confl,    2126 dec
  rutgers.edu/ii16e1.cnf(113): timeout
  rutgers.edu/ii16e2.cnf(114): 0.248, sat  ,  1070 confl,    1507 dec
  rutgers.edu/ii32a1.cnf(115): 0.305, sat  ,  1523 confl,    3287 dec
  rutgers.edu/ii32b1.cnf(116): 54.053, sat  , 28560 confl, 8015170 dec
  rutgers.edu/ii32b2.cnf(117): 0.640, sat  ,  2876 confl,   83906 dec
  rutgers.edu/ii32b3.cnf(118): 0.796, sat  ,  1978 confl,  101936 dec
  rutgers.edu/ii32b4.cnf(119): 0.548, sat  ,  1886 confl,    7567 dec
  rutgers.edu/ii32c1.cnf(120): timeout
  rutgers.edu/ii32c2.cnf(121): timeout
  rutgers.edu/ii32c3.cnf(122): 32.524, sat  ,  9319 confl, 4960298 dec
  rutgers.edu/ii32c4.cnf(123): timeout
  rutgers.edu/ii32d1.cnf(124): timeout
  rutgers.edu/ii32d2.cnf(125): timeout
  rutgers.edu/ii32d3.cnf(126): 11.567, sat  , 28084 confl,  102257 dec
  rutgers.edu/ii32e1.cnf(127): timeout
  rutgers.edu/ii32e2.cnf(128): timeout
  rutgers.edu/ii32e3.cnf(129): 1.084, sat  ,  3742 confl,  156363 dec
  rutgers.edu/ii32e4.cnf(130): 1.716, sat  ,  4680 confl,  189308 dec
  rutgers.edu/ii32e5.cnf(131): 2.105, sat  ,  3988 confl,  252364 dec
   rutgers.edu/ii8a1.cnf(132): 0.017, sat  ,    82 confl,    5415 dec
   rutgers.edu/ii8a2.cnf(133): timeout
   rutgers.edu/ii8a3.cnf(134): timeout
   rutgers.edu/ii8a4.cnf(135): timeout
   rutgers.edu/ii8b1.cnf(136): timeout
   rutgers.edu/ii8b2.cnf(137): timeout
   rutgers.edu/ii8b3.cnf(138): timeout
   rutgers.edu/ii8b4.cnf(139): timeout
   rutgers.edu/ii8c1.cnf(140): timeout
   rutgers.edu/ii8c2.cnf(141): timeout
   rutgers.edu/ii8d1.cnf(142): timeout
   rutgers.edu/ii8d2.cnf(143): timeout
   rutgers.edu/ii8e1.cnf(144): timeout
   rutgers.edu/ii8e2.cnf(145): timeout
   rutgers.edu/jnh10.cnf(146): 0.001, unsat,     0 confl,       0 dec
   rutgers.edu/jnh11.cnf(147): 0.001, unsat,     0 confl,       0 dec
   rutgers.edu/jnh12.cnf(148): 0.003, sat  ,     0 confl,       1 dec
   rutgers.edu/jnh13.cnf(149): 0.001, unsat,     0 confl,       0 dec
   rutgers.edu/jnh14.cnf(150): 0.001, unsat,     0 confl,       0 dec
   rutgers.edu/jnh15.cnf(151): 0.001, unsat,     0 confl,       0 dec
   rutgers.edu/jnh16.cnf(152): 0.006, unsat,    39 confl,      38 dec
   rutgers.edu/jnh17.cnf(153): 0.003, sat  ,     6 confl,      35 dec
   rutgers.edu/jnh18.cnf(154): 0.003, unsat,     5 confl,       4 dec
   rutgers.edu/jnh19.cnf(155): 0.003, unsat,     7 confl,       6 dec
    rutgers.edu/jnh1.cnf(156): 0.007, sat  ,    25 confl,     269 dec
  rutgers.edu/jnh201.cnf(157): 1.086, sat  ,   581 confl,  158958 dec
  rutgers.edu/jnh202.cnf(158): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/jnh203.cnf(159): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/jnh204.cnf(160): 0.006, sat  ,    38 confl,      59 dec
  rutgers.edu/jnh205.cnf(161): 0.004, sat  ,     6 confl,      35 dec
  rutgers.edu/jnh206.cnf(162): 0.003, unsat,     4 confl,       3 dec
  rutgers.edu/jnh207.cnf(163): 0.004, sat  ,    13 confl,      15 dec
  rutgers.edu/jnh208.cnf(164): 0.003, unsat,     5 confl,       4 dec
  rutgers.edu/jnh209.cnf(165): 0.004, sat  ,    12 confl,      85 dec
   rutgers.edu/jnh20.cnf(166): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/jnh210.cnf(167): 0.010, sat  ,     7 confl,    1996 dec
  rutgers.edu/jnh211.cnf(168): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/jnh212.cnf(169): 0.004, sat  ,    16 confl,      18 dec
  rutgers.edu/jnh213.cnf(170): 0.003, sat  ,     1 confl,      82 dec
  rutgers.edu/jnh214.cnf(171): 0.003, unsat,     3 confl,       2 dec
  rutgers.edu/jnh215.cnf(172): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/jnh216.cnf(173): 0.003, unsat,     4 confl,       3 dec
  rutgers.edu/jnh217.cnf(174): 0.037, sat  ,   134 confl,    3847 dec
  rutgers.edu/jnh218.cnf(175): 0.004, sat  ,     5 confl,     182 dec
  rutgers.edu/jnh219.cnf(176): 0.003, unsat,     4 confl,       3 dec
  rutgers.edu/jnh220.cnf(177): 0.006, sat  ,    36 confl,      51 dec
    rutgers.edu/jnh2.cnf(178): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/jnh301.cnf(179): 0.004, sat  ,    15 confl,      30 dec
  rutgers.edu/jnh302.cnf(180): 0.001, unsat,     0 confl,       0 dec
  rutgers.edu/jnh303.cnf(181): 0.003, unsat,     6 confl,       5 dec
```

```
        rutgers.edu/jnh304.cnf(182): 0.001, unsat,     0 confl,       0 dec
        rutgers.edu/jnh305.cnf(183): 0.001, unsat,     0 confl,       0 dec
        rutgers.edu/jnh306.cnf(184): 0.004, unsat,    16 confl,      15 dec
        rutgers.edu/jnh307.cnf(185): 0.001, unsat,     0 confl,       0 dec
        rutgers.edu/jnh308.cnf(186): 0.001, unsat,     0 confl,       0 dec
        rutgers.edu/jnh309.cnf(187): 0.001, unsat,     0 confl,       0 dec
        rutgers.edu/jnh310.cnf(188): 0.001, unsat,     0 confl,       0 dec
          rutgers.edu/jnh3.cnf(189): 0.003, unsat,    11 confl,      10 dec
          rutgers.edu/jnh4.cnf(190): 0.001, unsat,     0 confl,       0 dec
          rutgers.edu/jnh5.cnf(191): 0.001, unsat,     0 confl,       0 dec
          rutgers.edu/jnh6.cnf(192): 0.003, unsat,     6 confl,       5 dec
          rutgers.edu/jnh7.cnf(193): 0.003, sat  ,     0 confl,     133 dec
          rutgers.edu/jnh8.cnf(194): 0.001, unsat,     0 confl,       0 dec
          rutgers.edu/jnh9.cnf(195): 0.001, unsat,     0 confl,       0 dec
      rutgers.edu/par16-1-c.cnf(196): 0.137, sat  ,  1169 confl,    1169 dec
        rutgers.edu/par16-1.cnf(197): 0.212, sat  ,  1480 confl,    1480 dec
      rutgers.edu/par16-2-c.cnf(198): 0.173, sat  ,  1613 confl,    1613 dec
        rutgers.edu/par16-2.cnf(199): 0.276, sat  ,  1649 confl,    1649 dec
      rutgers.edu/par16-3-c.cnf(200): 0.135, sat  ,  1366 confl,    1366 dec
        rutgers.edu/par16-3.cnf(201): 0.190, sat  ,  1513 confl,    1513 dec
      rutgers.edu/par16-4-c.cnf(202): 0.050, sat  ,   571 confl,     571 dec
        rutgers.edu/par16-4.cnf(203): 0.240, sat  ,  1864 confl,    1864 dec
      rutgers.edu/par16-5-c.cnf(204): 0.115, sat  ,  1263 confl,    1263 dec
        rutgers.edu/par16-5.cnf(205): 0.319, sat  ,  2127 confl,    2127 dec
      rutgers.edu/par32-1-c.cnf(206): timeout
        rutgers.edu/par32-1.cnf(207): timeout
      rutgers.edu/par32-2-c.cnf(208): timeout
        rutgers.edu/par32-2.cnf(209): timeout
      rutgers.edu/par32-3-c.cnf(210): timeout
        rutgers.edu/par32-3.cnf(211): timeout
      rutgers.edu/par32-4-c.cnf(212): timeout
        rutgers.edu/par32-4.cnf(213): timeout
      rutgers.edu/par32-5-c.cnf(214): timeout
        rutgers.edu/par32-5.cnf(215): timeout
       rutgers.edu/par8-1-c.cnf(216): 0.003, sat  ,     3 confl,       3 dec
         rutgers.edu/par8-1.cnf(217): 0.003, sat  ,     3 confl,       3 dec
       rutgers.edu/par8-2-c.cnf(218): 0.002, sat  ,     1 confl,       1 dec
         rutgers.edu/par8-2.cnf(219): 0.003, sat  ,     3 confl,       3 dec
       rutgers.edu/par8-3-c.cnf(220): 0.003, sat  ,     7 confl,       7 dec
         rutgers.edu/par8-3.cnf(221): 0.003, sat  ,     5 confl,       5 dec
       rutgers.edu/par8-4-c.cnf(222): 0.003, sat  ,     1 confl,       1 dec
         rutgers.edu/par8-4.cnf(223): 0.003, sat  ,     1 confl,       1 dec
       rutgers.edu/par8-5-c.cnf(224): 0.003, sat  ,     3 confl,       3 dec
         rutgers.edu/par8-5.cnf(225): 0.003, sat  ,     2 confl,       2 dec
     rutgers.edu/pret150_25.cnf(226): 0.013, unsat,   705 confl,    3191 dec
     rutgers.edu/pret150_40.cnf(227): 0.013, unsat,   685 confl,    3088 dec
     rutgers.edu/pret150_60.cnf(228): 0.019, unsat,   931 confl,    4802 dec
     rutgers.edu/pret150_75.cnf(229): 0.011, unsat,   572 confl,    2680 dec
      rutgers.edu/pret60_25.cnf(230): 0.003, unsat,   183 confl,     449 dec
      rutgers.edu/pret60_40.cnf(231): 0.004, unsat,   212 confl,     529 dec
      rutgers.edu/pret60_60.cnf(232): 0.004, unsat,   201 confl,     480 dec
      rutgers.edu/pret60_75.cnf(233): 0.004, unsat,   203 confl,     585 dec
    rutgers.edu/ssa0432-003.cnf(234): 0.001, unsat,     0 confl,       0 dec
    rutgers.edu/ssa2670-130.cnf(235): 0.015, unsat,    96 confl,      95 dec
    rutgers.edu/ssa2670-141.cnf(236): 0.014, unsat,   152 confl,     306 dec
    rutgers.edu/ssa6288-047.cnf(237): 0.013, unsat,     0 confl,       0 dec
    rutgers.edu/ssa7552-038.cnf(238): 0.027, sat  ,    31 confl,     863 dec
    rutgers.edu/ssa7552-158.cnf(239): 0.013, sat  ,     1 confl,     528 dec
    rutgers.edu/ssa7552-159.cnf(240): 0.021, sat  ,    13 confl,     827 dec
    rutgers.edu/ssa7552-160.cnf(241): 0.015, sat  ,     9 confl,     603 dec
```

## 4.4 rutgers.edu one solution DualSat

```
  rutgers.edu/samename/jnh8.cnf(0):  0.000s, unsat,    12 confl,      17 dec,      1 nd, 767 basecl, 8 learnts
    rutgers.edu/samename/par16-1.cnf(1):  0.774s, sat  , 18286 confl,   22849 dec,     98 nd, 2244 basecl, 7261 learnts
    rutgers.edu/aim-100-1_6-no-1.cnf(2):  0.000s, unsat,    25 confl,      70 dec,      6 nd, 160 basecl, 22 learnts
    rutgers.edu/aim-100-1_6-no-2.cnf(3):  0.000s, unsat,    34 confl,      90 dec,      5 nd, 158 basecl, 32 learnts
    rutgers.edu/aim-100-1_6-no-3.cnf(4):  0.000s, unsat,    33 confl,      84 dec,      5 nd, 156 basecl, 27 learnts
    rutgers.edu/aim-100-1_6-no-4.cnf(5):  0.000s, unsat,    24 confl,      81 dec,      4 nd, 158 basecl, 20 learnts
  rutgers.edu/aim-100-1_6-yes1-1.cnf(6):  0.000s, sat  ,    12 confl,      39 dec,      6 nd, 159 basecl, 12 learnts
  rutgers.edu/aim-100-1_6-yes1-2.cnf(7):  0.000s, sat  ,    31 confl,     103 dec,      5 nd, 159 basecl, 27 learnts
  rutgers.edu/aim-100-1_6-yes1-3.cnf(8):  0.000s, sat  ,    28 confl,     107 dec,      9 nd, 160 basecl, 23 learnts
  rutgers.edu/aim-100-1_6-yes1-4.cnf(9):  0.000s, sat  ,    22 confl,      82 dec,      3 nd, 160 basecl, 19 learnts
    rutgers.edu/aim-100-2_0-no-1.cnf(10):  0.000s, unsat,    12 confl,      36 dec,      2 nd, 200 basecl, 9 learnts
    rutgers.edu/aim-100-2_0-no-2.cnf(11):  0.000s, unsat,    27 confl,      67 dec,      1 nd, 199 basecl, 24 learnts
    rutgers.edu/aim-100-2_0-no-3.cnf(12):  0.000s, unsat,    20 confl,      64 dec,      3 nd, 196 basecl, 17 learnts
    rutgers.edu/aim-100-2_0-no-4.cnf(13):  0.000s, unsat,    18 confl,      52 dec,      2 nd, 200 basecl, 14 learnts
  rutgers.edu/aim-100-2_0-yes1-1.cnf(14):  0.000s, sat  ,    75 confl,     190 dec,      5 nd, 197 basecl, 68 learnts
  rutgers.edu/aim-100-2_0-yes1-2.cnf(15):  0.000s, sat  ,    58 confl,     138 dec,      8 nd, 198 basecl, 52 learnts
  rutgers.edu/aim-100-2_0-yes1-3.cnf(16):  0.000s, sat  ,    20 confl,      68 dec,      6 nd, 199 basecl, 13 learnts
  rutgers.edu/aim-100-2_0-yes1-4.cnf(17):  0.000s, sat  ,    42 confl,     102 dec,      4 nd, 199 basecl, 38 learnts
  rutgers.edu/aim-100-3_4-yes1-1.cnf(18):  0.000s, sat  ,    41 confl,      69 dec,      2 nd, 339 basecl, 41 learnts
  rutgers.edu/aim-100-3_4-yes1-2.cnf(19):  0.000s, sat  ,    53 confl,     100 dec,      3 nd, 337 basecl, 50 learnts
  rutgers.edu/aim-100-3_4-yes1-3.cnf(20):  0.000s, sat  ,    55 confl,      99 dec,      3 nd, 339 basecl, 51 learnts
  rutgers.edu/aim-100-3_4-yes1-4.cnf(21):  0.000s, sat  ,    56 confl,     101 dec,      2 nd, 338 basecl, 56 learnts
  rutgers.edu/aim-100-6_0-yes1-1.cnf(22):  0.000s, sat  ,    27 confl,      46 dec,      2 nd, 596 basecl, 27 learnts
  rutgers.edu/aim-100-6_0-yes1-2.cnf(23):  0.000s, sat  ,    31 confl,      43 dec,      2 nd, 599 basecl, 31 learnts
  rutgers.edu/aim-100-6_0-yes1-3.cnf(24):  0.000s, sat  ,    36 confl,      61 dec,      2 nd, 598 basecl, 35 learnts
  rutgers.edu/aim-100-6_0-yes1-4.cnf(25):  0.000s, sat  ,    41 confl,      58 dec,      2 nd, 594 basecl, 41 learnts
    rutgers.edu/aim-200-1_6-no-1.cnf(26):  0.000s, unsat,    26 confl,     132 dec,      6 nd, 319 basecl, 23 learnts
    rutgers.edu/aim-200-1_6-no-2.cnf(27):  0.000s, unsat,    37 confl,     201 dec,     11 nd, 316 basecl, 29 learnts
    rutgers.edu/aim-200-1_6-no-3.cnf(28):  0.001s, unsat,    50 confl,     303 dec,     17 nd, 318 basecl, 46 learnts
    rutgers.edu/aim-200-1_6-no-4.cnf(29):  0.000s, unsat,    26 confl,     127 dec,      4 nd, 320 basecl, 24 learnts
  rutgers.edu/aim-200-1_6-yes1-1.cnf(30):  0.000s, sat  ,    19 confl,     135 dec,      8 nd, 319 basecl, 12 learnts
  rutgers.edu/aim-200-1_6-yes1-2.cnf(31):  0.001s, sat  ,    66 confl,     311 dec,     14 nd, 319 basecl, 56 learnts
  rutgers.edu/aim-200-1_6-yes1-3.cnf(32):  0.000s, sat  ,    28 confl,     181 dec,     13 nd, 318 basecl, 24 learnts
  rutgers.edu/aim-200-1_6-yes1-4.cnf(33):  0.000s, sat  ,    43 confl,     224 dec,     12 nd, 316 basecl, 34 learnts
    rutgers.edu/aim-200-2_0-no-1.cnf(34):  0.000s, unsat,    20 confl,     115 dec,      5 nd, 398 basecl, 17 learnts
```

```
    rutgers.edu/aim-200-2_0-no-2.cnf(35):  0.000s, unsat,    27 confl,     169 dec,      8 nd, 398 basecl, 23 learnts
    rutgers.edu/aim-200-2_0-no-3.cnf(36):  0.000s, unsat,    26 confl,     123 dec,      5 nd, 397 basecl, 23 learnts
    rutgers.edu/aim-200-2_0-no-4.cnf(37):  0.000s, unsat,    24 confl,     141 dec,      6 nd, 397 basecl, 21 learnts
  rutgers.edu/aim-200-2_0-yes1-1.cnf(38):  0.001s, sat  ,   153 confl,     441 dec,      9 nd, 396 basecl, 140 learnts
  rutgers.edu/aim-200-2_0-yes1-2.cnf(39):  0.001s, sat  ,    75 confl,     291 dec,     10 nd, 398 basecl, 59 learnts
  rutgers.edu/aim-200-2_0-yes1-3.cnf(40):  0.000s, sat  ,    54 confl,     248 dec,     12 nd, 398 basecl, 45 learnts
  rutgers.edu/aim-200-2_0-yes1-4.cnf(41):  0.001s, sat  ,    77 confl,     304 dec,      6 nd, 397 basecl, 71 learnts
  rutgers.edu/aim-200-3_4-yes1-1.cnf(42):  0.000s, sat  ,    56 confl,     150 dec,      3 nd, 677 basecl, 53 learnts
  rutgers.edu/aim-200-3_4-yes1-2.cnf(43):  0.000s, sat  ,    67 confl,     175 dec,      5 nd, 679 basecl, 62 learnts
  rutgers.edu/aim-200-3_4-yes1-3.cnf(44):  0.008s, sat  ,   870 confl,    1353 dec,      6 nd, 677 basecl, 855 learnts
  rutgers.edu/aim-200-3_4-yes1-4.cnf(45):  0.000s, sat  ,    51 confl,     116 dec,      4 nd, 675 basecl, 42 learnts
  rutgers.edu/aim-200-6_0-yes1-1.cnf(46):  0.000s, sat  ,    28 confl,      60 dec,      2 nd, 1170 basecl, 25 learnts
  rutgers.edu/aim-200-6_0-yes1-2.cnf(47):  0.003s, sat  ,   405 confl,     611 dec,      2 nd, 1183 basecl, 403 learnts
  rutgers.edu/aim-200-6_0-yes1-3.cnf(48):  0.000s, sat  ,    45 confl,      77 dec,      2 nd, 1180 basecl, 43 learnts
  rutgers.edu/aim-200-6_0-yes1-4.cnf(49):  0.000s, sat  ,    43 confl,      80 dec,      3 nd, 1179 basecl, 41 learnts
     rutgers.edu/aim-50-1_6-no-1.cnf(50):  0.000s, unsat,    11 confl,      27 dec,      5 nd, 75 basecl, 8 learnts
     rutgers.edu/aim-50-1_6-no-2.cnf(51):  0.000s, unsat,    13 confl,      19 dec,      1 nd, 80 basecl, 11 learnts
     rutgers.edu/aim-50-1_6-no-3.cnf(52):  0.000s, unsat,    16 confl,      21 dec,      2 nd, 79 basecl, 12 learnts
     rutgers.edu/aim-50-1_6-no-4.cnf(53):  0.000s, unsat,     9 confl,      17 dec,      2 nd, 80 basecl, 6 learnts
   rutgers.edu/aim-50-1_6-yes1-1.cnf(54):  0.000s, sat  ,    12 confl,      18 dec,      3 nd, 80 basecl, 12 learnts
   rutgers.edu/aim-50-1_6-yes1-2.cnf(55):  0.000s, sat  ,     5 confl,      12 dec,      2 nd, 79 basecl, 5 learnts
   rutgers.edu/aim-50-1_6-yes1-3.cnf(56):  0.000s, sat  ,     8 confl,      15 dec,      2 nd, 78 basecl, 7 learnts
   rutgers.edu/aim-50-1_6-yes1-4.cnf(57):  0.000s, sat  ,     3 confl,       5 dec,      2 nd, 78 basecl, 2 learnts
     rutgers.edu/aim-50-2_0-no-1.cnf(58):  0.000s, unsat,     6 confl,      16 dec,      2 nd, 100 basecl, 3 learnts
     rutgers.edu/aim-50-2_0-no-2.cnf(59):  0.000s, unsat,    12 confl,      31 dec,      3 nd, 100 basecl, 7 learnts
     rutgers.edu/aim-50-2_0-no-3.cnf(60):  0.000s, unsat,    15 confl,      23 dec,      2 nd, 100 basecl, 11 learnts
     rutgers.edu/aim-50-2_0-no-4.cnf(61):  0.000s, unsat,    21 confl,      35 dec,      1 nd, 96 basecl, 18 learnts
   rutgers.edu/aim-50-2_0-yes1-1.cnf(62):  0.000s, sat  ,    13 confl,      41 dec,      6 nd, 99 basecl, 7 learnts
   rutgers.edu/aim-50-2_0-yes1-2.cnf(63):  0.000s, sat  ,    21 confl,      32 dec,      3 nd, 99 basecl, 18 learnts
   rutgers.edu/aim-50-2_0-yes1-3.cnf(64):  0.000s, sat  ,    12 confl,      30 dec,      3 nd, 100 basecl, 10 learnts
   rutgers.edu/aim-50-2_0-yes1-4.cnf(65):  0.000s, sat  ,     5 confl,      11 dec,      3 nd, 99 basecl, 5 learnts
   rutgers.edu/aim-50-3_4-yes1-1.cnf(66):  0.000s, sat  ,    20 confl,      33 dec,      2 nd, 169 basecl, 15 learnts
   rutgers.edu/aim-50-3_4-yes1-2.cnf(67):  0.000s, sat  ,    10 confl,      14 dec,      3 nd, 170 basecl, 10 learnts
   rutgers.edu/aim-50-3_4-yes1-3.cnf(68):  0.000s, sat  ,    16 confl,      30 dec,      2 nd, 170 basecl, 16 learnts
   rutgers.edu/aim-50-3_4-yes1-4.cnf(69):  0.000s, sat  ,     6 confl,      11 dec,      2 nd, 168 basecl, 6 learnts
   rutgers.edu/aim-50-6_0-yes1-1.cnf(70):  0.000s, sat  ,    11 confl,      18 dec,      2 nd, 299 basecl, 10 learnts
   rutgers.edu/aim-50-6_0-yes1-2.cnf(71):  0.000s, sat  ,     1 confl,       5 dec,      2 nd, 296 basecl, 1 learnts
   rutgers.edu/aim-50-6_0-yes1-3.cnf(72):  0.000s, sat  ,     7 confl,      15 dec,      2 nd, 295 basecl, 7 learnts
   rutgers.edu/aim-50-6_0-yes1-4.cnf(73):  0.000s, sat  ,    13 confl,      15 dec,      2 nd, 298 basecl, 13 learnts
          rutgers.edu/bf0432-007.cnf(74):  0.012s, unsat,   387 confl,     701 dec,      7 nd, 1867 basecl, 375 learnts
          rutgers.edu/bf1355-075.cnf(75):  0.002s, unsat,    37 confl,      79 dec,      4 nd, 3033 basecl, 29 learnts
          rutgers.edu/bf1355-638.cnf(76):  0.003s, unsat,    44 confl,     138 dec,      9 nd, 3026 basecl, 31 learnts
          rutgers.edu/bf2670-001.cnf(77):  0.001s, unsat,    21 confl,      66 dec,      2 nd, 917 basecl, 18 learnts
           rutgers.edu/dubois100.cnf(78):  0.000s, sat  ,     0 confl,     100 dec,      1 nd, 396 basecl, 0 learnts - solution
counter overflow
            rutgers.edu/dubois20.cnf(79):  0.000s, unsat,   115 confl,     324 dec,     13 nd, 160 basecl, 110 learnts
            rutgers.edu/dubois21.cnf(80):  0.000s, unsat,   150 confl,     390 dec,      5 nd, 168 basecl, 147 learnts
            rutgers.edu/dubois22.cnf(81):  0.000s, unsat,   142 confl,     426 dec,     15 nd, 176 basecl, 138 learnts
            rutgers.edu/dubois23.cnf(82):  0.000s, unsat,   153 confl,     453 dec,      7 nd, 184 basecl, 151 learnts
            rutgers.edu/dubois24.cnf(83):  0.001s, unsat,   203 confl,     565 dec,     15 nd, 192 basecl, 199 learnts
            rutgers.edu/dubois25.cnf(84):  0.001s, unsat,   221 confl,     552 dec,      7 nd, 200 basecl, 216 learnts
            rutgers.edu/dubois26.cnf(85):  0.001s, unsat,   201 confl,     635 dec,     18 nd, 208 basecl, 197 learnts
            rutgers.edu/dubois27.cnf(86):  0.001s, unsat,   247 confl,     601 dec,      8 nd, 216 basecl, 241 learnts
            rutgers.edu/dubois28.cnf(87):  0.001s, unsat,   184 confl,     681 dec,     18 nd, 224 basecl, 181 learnts
            rutgers.edu/dubois29.cnf(88):  0.001s, unsat,   216 confl,     608 dec,     11 nd, 232 basecl, 213 learnts
            rutgers.edu/dubois30.cnf(89):  0.001s, unsat,   208 confl,     787 dec,     22 nd, 240 basecl, 203 learnts
            rutgers.edu/dubois50.cnf(90):  0.002s, unsat,   463 confl,    2218 dec,     61 nd, 400 basecl, 456 learnts
               rutgers.edu/f1000.cnf(91): expired
               rutgers.edu/f2000.cnf(92): expired
                rutgers.edu/f600.cnf(93): expired
             rutgers.edu/g125.17.cnf(94): expired
             rutgers.edu/g125.18.cnf(95): expired
             rutgers.edu/g250.15.cnf(96): expired
             rutgers.edu/g250.29.cnf(97): expired
              rutgers.edu/hanoi4.cnf(98):  0.089s, sat  ,  3613 confl,    5708 dec,     26 nd, 2325 basecl, 2978 learnts
              rutgers.edu/hanoi5.cnf(99):  2.179s, sat  , 58735 confl,   91711 dec,    388 nd, 6913 basecl, 18408 learnts
             rutgers.edu/hole10.cnf(100): expired
              rutgers.edu/hole6.cnf(101):  0.004s, unsat,   907 confl,    1011 dec,      1 nd, 7 basecl, 867 learnts
              rutgers.edu/hole7.cnf(102):  0.154s, unsat,  9942 confl,   10698 dec,      1 nd, 8 basecl, 5765 learnts
              rutgers.edu/hole8.cnf(103): 21.194s, unsat, 827195 confl,  871077 dec,      1 nd, 9 basecl, 7911 learnts
              rutgers.edu/hole9.cnf(104): 193.267s, unsat, 8250618 confl, 8696421 dec,      1 nd, 10 basecl, 8377 learnts
             rutgers.edu/ii16a1.cnf(105):  0.012s, sat  ,   470 confl,    1280 dec,     20 nd, 408 basecl, 470 learnts
             rutgers.edu/ii16a2.cnf(106):  0.000s, sat  ,     1 confl,      35 dec,      2 nd, 625 basecl, 1 learnts
             rutgers.edu/ii16b1.cnf(107):  0.006s, sat  ,   169 confl,     429 dec,     10 nd, 1112 basecl, 169 learnts - solution
counter overflow
             rutgers.edu/ii16b2.cnf(108):  0.002s, sat  ,    83 confl,     146 dec,      7 nd, 889 basecl, 83 learnts - solution
counter overflow
             rutgers.edu/ii16c1.cnf(109):  0.013s, sat  ,   398 confl,    1502 dec,     23 nd, 1107 basecl, 398 learnts - solution
counter overflow
             rutgers.edu/ii16c2.cnf(110):  0.007s, sat  ,   316 confl,     503 dec,     12 nd, 1003 basecl, 316 learnts - solution
counter overflow
             rutgers.edu/ii16d1.cnf(111):  0.003s, sat  ,    90 confl,     264 dec,     10 nd, 1181 basecl, 90 learnts - solution
counter overflow
             rutgers.edu/ii16d2.cnf(112):  0.021s, sat  ,   943 confl,    1383 dec,     19 nd, 1069 basecl, 906 learnts - solution
counter overflow
             rutgers.edu/ii16e1.cnf(113):  0.112s, unsat,  3686 confl,    8535 dec,    107 nd, 2286 basecl, 1361 learnts - solution
counter overflow
             rutgers.edu/ii16e2.cnf(114):  0.047s, sat  ,  1697 confl,    2309 dec,     19 nd, 1297 basecl, 1467 learnts
             rutgers.edu/ii32a1.cnf(115):  0.013s, sat  ,   598 confl,     764 dec,     16 nd, 572 basecl, 504 learnts
             rutgers.edu/ii32b1.cnf(116):  0.000s, sat  ,     5 confl,      20 dec,      2 nd, 126 basecl, 5 learnts
             rutgers.edu/ii32b2.cnf(117):  0.008s, sat  ,   686 confl,     862 dec,     20 nd, 254 basecl, 550 learnts
             rutgers.edu/ii32b3.cnf(118):  0.001s, sat  ,    32 confl,      52 dec,      3 nd, 646 basecl, 32 learnts
             rutgers.edu/ii32b4.cnf(119):  0.004s, sat  ,   169 confl,     235 dec,      8 nd, 774 basecl, 169 learnts
             rutgers.edu/ii32c1.cnf(120):  0.000s, unsat,     0 confl,      32 dec,      1 nd, 128 basecl, 0 learnts - solution
counter overflow
             rutgers.edu/ii32c2.cnf(121):  0.000s, sat  ,     0 confl,      48 dec,      1 nd, 262 basecl, 0 learnts
             rutgers.edu/ii32c3.cnf(122):  0.000s, sat  ,     1 confl,      54 dec,      2 nd, 392 basecl, 1 learnts
             rutgers.edu/ii32c4.cnf(123):  0.018s, sat  ,   208 confl,     290 dec,      9 nd, 2622 basecl, 207 learnts
             rutgers.edu/ii32d1.cnf(124):  0.000s, sat  ,     3 confl,      31 dec,      2 nd, 143 basecl, 3 learnts
             rutgers.edu/ii32d2.cnf(125):  0.052s, sat  ,  3205 confl,    4486 dec,     80 nd, 289 basecl, 1708 learnts
```

```
rutgers.edu/ii32d3.cnf(126):  1.766s, sat  , 35692 confl,   45470 dec,    435 nd, 1174 basecl, 6910 learnts
rutgers.edu/ii32e1.cnf(127):  0.000s, sat  ,     0 confl,      20 dec,      1 nd, 130 basecl, 0 learnts
rutgers.edu/ii32e2.cnf(128):  0.000s, sat  ,     0 confl,      13 dec,      1 nd, 250 basecl, 0 learnts
rutgers.edu/ii32e3.cnf(129):  0.000s, sat  ,     4 confl,      19 dec,      2 nd, 508 basecl, 4 learnts
rutgers.edu/ii32e4.cnf(130):  0.012s, sat  ,   614 confl,     785 dec,     15 nd, 770 basecl, 494 learnts
rutgers.edu/ii32e5.cnf(131):  0.005s, sat  ,   175 confl,     228 dec,      7 nd, 980 basecl, 175 learnts
 rutgers.edu/ii8a1.cnf(132):  0.000s, sat  ,     0 confl,      15 dec,      1 nd, 18 basecl, 0 learnts
 rutgers.edu/ii8a2.cnf(133):  0.000s, sat  ,     6 confl,      32 dec,      3 nd, 80 basecl, 6 learnts
 rutgers.edu/ii8a3.cnf(134):  0.000s, sat  ,     5 confl,      39 dec,      2 nd, 160 basecl, 5 learnts
 rutgers.edu/ii8a4.cnf(135):  0.000s, sat  ,     5 confl,      30 dec,      2 nd, 350 basecl, 5 learnts
 rutgers.edu/ii8b1.cnf(136):  0.000s, sat  ,     0 confl,      21 dec,      1 nd, 100 basecl, 0 learnts
 rutgers.edu/ii8b2.cnf(137):  0.000s, sat  ,     0 confl,      20 dec,      1 nd, 200 basecl, 0 learnts
 rutgers.edu/ii8b3.cnf(138):  0.000s, sat  ,     0 confl,      26 dec,      1 nd, 300 basecl, 0 learnts
 rutgers.edu/ii8b4.cnf(139):  0.000s, sat  ,     0 confl,      25 dec,      1 nd, 390 basecl, 0 learnts
 rutgers.edu/ii8c1.cnf(140):  0.000s, sat  ,     0 confl,      41 dec,      1 nd, 185 basecl, 0 learnts
 rutgers.edu/ii8c2.cnf(141):  0.000s, sat  ,     0 confl,      39 dec,      1 nd, 289 basecl, 0 learnts
 rutgers.edu/ii8d1.cnf(142):  0.000s, sat  ,     0 confl,      52 dec,      1 nd, 167 basecl, 0 learnts
 rutgers.edu/ii8d2.cnf(143):  0.000s, sat  ,     3 confl,      70 dec,      2 nd, 307 basecl, 3 learnts
 rutgers.edu/ii8e1.cnf(144):  0.000s, sat  ,    34 confl,     111 dec,      4 nd, 176 basecl, 34 learnts - solution
counter overflow
 rutgers.edu/ii8e2.cnf(145):  0.001s, sat  ,    72 confl,     217 dec,      5 nd, 361 basecl, 72 learnts
 rutgers.edu/jnh10.cnf(146):  0.000s, unsat,    19 confl,      18 dec,      1 nd, 768 basecl, 9 learnts
 rutgers.edu/jnh11.cnf(147):  0.001s, unsat,    53 confl,      69 dec,      1 nd, 783 basecl, 46 learnts
 rutgers.edu/jnh12.cnf(148):  0.000s, sat  ,    17 confl,      26 dec,      2 nd, 774 basecl, 14 learnts
 rutgers.edu/jnh13.cnf(149):  0.000s, unsat,     8 confl,      10 dec,      1 nd, 769 basecl, 4 learnts
 rutgers.edu/jnh14.cnf(150):  0.000s, unsat,    20 confl,      24 dec,      1 nd, 783 basecl, 16 learnts
 rutgers.edu/jnh15.cnf(151):  0.000s, unsat,    41 confl,      51 dec,      1 nd, 767 basecl, 31 learnts
 rutgers.edu/jnh16.cnf(152):  0.013s, unsat,   877 confl,    1086 dec,      1 nd, 795 basecl, 842 learnts
 rutgers.edu/jnh17.cnf(153):  0.000s, sat  ,    13 confl,      24 dec,      2 nd, 768 basecl, 8 learnts
 rutgers.edu/jnh18.cnf(154):  0.001s, unsat,    54 confl,      69 dec,      2 nd, 775 basecl, 45 learnts
 rutgers.edu/jnh19.cnf(155):  0.000s, unsat,    26 confl,      36 dec,      1 nd, 789 basecl, 19 learnts
  rutgers.edu/jnh1.cnf(156):  0.000s, sat  ,    19 confl,      36 dec,      2 nd, 787 basecl, 19 learnts
rutgers.edu/jnh201.cnf(157):  0.000s, sat  ,     2 confl,      17 dec,      3 nd, 744 basecl, 2 learnts
rutgers.edu/jnh202.cnf(158):  0.000s, unsat,     9 confl,      10 dec,      1 nd, 729 basecl, 3 learnts
rutgers.edu/jnh203.cnf(159):  0.001s, unsat,    48 confl,      54 dec,      1 nd, 723 basecl, 42 learnts
rutgers.edu/jnh204.cnf(160):  0.001s, sat  ,    79 confl,     112 dec,      2 nd, 743 basecl, 79 learnts
rutgers.edu/jnh205.cnf(161):  0.001s, sat  ,    77 confl,     113 dec,      2 nd, 726 basecl, 74 learnts
rutgers.edu/jnh206.cnf(162):  0.001s, unsat,    83 confl,     106 dec,      1 nd, 727 basecl, 71 learnts
rutgers.edu/jnh207.cnf(163):  0.001s, sat  ,    86 confl,     109 dec,      2 nd, 742 basecl, 85 learnts
rutgers.edu/jnh208.cnf(164):  0.001s, unsat,    41 confl,      49 dec,      2 nd, 722 basecl, 34 learnts
rutgers.edu/jnh209.cnf(165):  0.000s, sat  ,    28 confl,      50 dec,      2 nd, 740 basecl, 28 learnts
 rutgers.edu/jnh20.cnf(166):  0.000s, unsat,    17 confl,      24 dec,      2 nd, 768 basecl, 12 learnts
rutgers.edu/jnh210.cnf(167):  0.001s, sat  ,    44 confl,      58 dec,      2 nd, 734 basecl, 40 learnts
rutgers.edu/jnh211.cnf(168):  0.000s, unsat,     9 confl,      10 dec,      2 nd, 716 basecl, 5 learnts
rutgers.edu/jnh212.cnf(169):  0.003s, sat  ,   217 confl,     274 dec,      2 nd, 746 basecl, 213 learnts
rutgers.edu/jnh213.cnf(170):  0.000s, sat  ,     8 confl,      14 dec,      3 nd, 724 basecl, 7 learnts
rutgers.edu/jnh214.cnf(171):  0.000s, unsat,    29 confl,      34 dec,      2 nd, 729 basecl, 22 learnts
rutgers.edu/jnh215.cnf(172):  0.000s, unsat,    29 confl,      35 dec,      1 nd, 733 basecl, 22 learnts
rutgers.edu/jnh216.cnf(173):  0.001s, unsat,    75 confl,      92 dec,      1 nd, 719 basecl, 67 learnts
rutgers.edu/jnh217.cnf(174):  0.000s, sat  ,    17 confl,      42 dec,      2 nd, 752 basecl, 17 learnts
rutgers.edu/jnh218.cnf(175):  0.000s, sat  ,    25 confl,      38 dec,      2 nd, 724 basecl, 21 learnts
rutgers.edu/jnh219.cnf(176):  0.001s, unsat,    80 confl,      93 dec,      1 nd, 724 basecl, 74 learnts
rutgers.edu/jnh220.cnf(177):  0.000s, sat  ,    19 confl,      36 dec,      3 nd, 748 basecl, 19 learnts
  rutgers.edu/jnh2.cnf(178):  0.000s, unsat,     8 confl,       8 dec,      2 nd, 773 basecl, 4 learnts
rutgers.edu/jnh301.cnf(179):  0.001s, sat  ,    79 confl,     115 dec,      3 nd, 835 basecl, 78 learnts
rutgers.edu/jnh302.cnf(180):  0.000s, unsat,     5 confl,       5 dec,      1 nd, 817 basecl, 3 learnts
rutgers.edu/jnh303.cnf(181):  0.001s, unsat,    39 confl,      49 dec,      1 nd, 821 basecl, 31 learnts
rutgers.edu/jnh304.cnf(182):  0.000s, unsat,     5 confl,       4 dec,      1 nd, 830 basecl, 0 learnts
rutgers.edu/jnh305.cnf(183):  0.000s, unsat,     8 confl,       7 dec,      1 nd, 811 basecl, 3 learnts
rutgers.edu/jnh306.cnf(184):  0.002s, unsat,   178 confl,     213 dec,      1 nd, 838 basecl, 164 learnts
rutgers.edu/jnh307.cnf(185):  0.000s, unsat,     6 confl,       7 dec,      2 nd, 811 basecl, 1 learnts
rutgers.edu/jnh308.cnf(186):  0.001s, unsat,    45 confl,      57 dec,      1 nd, 833 basecl, 39 learnts
rutgers.edu/jnh309.cnf(187):  0.000s, unsat,    10 confl,      12 dec,      1 nd, 823 basecl, 5 learnts
rutgers.edu/jnh310.cnf(188):  0.000s, unsat,     5 confl,       5 dec,      1 nd, 815 basecl, 2 learnts
  rutgers.edu/jnh3.cnf(189):  0.002s, unsat,   123 confl,     155 dec,      2 nd, 771 basecl, 109 learnts
  rutgers.edu/jnh4.cnf(190):  0.000s, unsat,    30 confl,      32 dec,      1 nd, 790 basecl, 21 learnts
  rutgers.edu/jnh5.cnf(191):  0.000s, unsat,    17 confl,      19 dec,      1 nd, 767 basecl, 11 learnts
  rutgers.edu/jnh6.cnf(192):  0.001s, unsat,    66 confl,      81 dec,      2 nd, 785 basecl, 58 learnts
  rutgers.edu/jnh7.cnf(193):  0.000s, sat  ,     7 confl,      25 dec,      2 nd, 780 basecl, 7 learnts
  rutgers.edu/jnh8.cnf(194):  0.000s, unsat,    12 confl,      17 dec,      1 nd, 767 basecl, 8 learnts
  rutgers.edu/jnh9.cnf(195):  0.000s, unsat,    17 confl,      22 dec,      3 nd, 790 basecl, 14 learnts
rutgers.edu/par16-1-c.cnf(196):  0.141s, sat  ,  5761 confl,    7168 dec,      2 nd, 1142 basecl, 4166 learnts
  rutgers.edu/par16-1.cnf(197):  0.769s, sat  , 18286 confl,   22849 dec,     98 nd, 2244 basecl, 7261 learnts
rutgers.edu/par16-2-c.cnf(198):  0.296s, sat  , 10362 confl,   12667 dec,      2 nd, 1270 basecl, 6051 learnts
  rutgers.edu/par16-2.cnf(199):  0.478s, sat  , 10319 confl,   13349 dec,     75 nd, 2372 basecl, 5542 learnts
rutgers.edu/par16-3-c.cnf(200):  0.306s, sat  ,  9982 confl,   12125 dec,      6 nd, 1210 basecl, 5961 learnts
  rutgers.edu/par16-3.cnf(201):  4.289s, unsat, 72351 confl,   83710 dec,     24 nd, 2312 basecl, 10012 learnts
rutgers.edu/par16-4-c.cnf(202):  0.139s, sat  ,  5561 confl,    6819 dec,      2 nd, 1170 basecl, 3916 learnts
  rutgers.edu/par16-4.cnf(203):  0.546s, sat  , 12470 confl,   15402 dec,     28 nd, 2272 basecl, 6613 learnts
rutgers.edu/par16-5-c.cnf(204):  0.375s, sat  , 12468 confl,   15182 dec,      2 nd, 1238 basecl, 6685 learnts
  rutgers.edu/par16-5.cnf(205):  0.904s, sat  , 19808 confl,   24777 dec,     88 nd, 2340 basecl, 7570 learnts
rutgers.edu/par32-1-c.cnf(206): expired
  rutgers.edu/par32-1.cnf(207): expired
rutgers.edu/par32-2-c.cnf(208): expired
  rutgers.edu/par32-2.cnf(209): expired
rutgers.edu/par32-3-c.cnf(210): expired
  rutgers.edu/par32-3.cnf(211): expired
rutgers.edu/par32-4-c.cnf(212): expired
  rutgers.edu/par32-4.cnf(213): expired
rutgers.edu/par32-5-c.cnf(214): expired
  rutgers.edu/par32-5.cnf(215): expired
 rutgers.edu/par8-1-c.cnf(216):  0.000s, sat  ,    13 confl,      16 dec,      2 nd, 224 basecl, 11 learnts
   rutgers.edu/par8-1.cnf(217):  0.001s, sat  ,    69 confl,     125 dec,      9 nd, 772 basecl, 50 learnts
 rutgers.edu/par8-2-c.cnf(218):  0.000s, sat  ,     8 confl,      12 dec,      2 nd, 240 basecl, 8 learnts
   rutgers.edu/par8-2.cnf(219):  0.001s, sat  ,    53 confl,     115 dec,     13 nd, 788 basecl, 40 learnts
 rutgers.edu/par8-3-c.cnf(220):  0.000s, sat  ,    30 confl,      31 dec,      2 nd, 268 basecl, 27 learnts
   rutgers.edu/par8-3.cnf(221):  0.001s, sat  ,   122 confl,     225 dec,     14 nd, 816 basecl, 108 learnts
 rutgers.edu/par8-4-c.cnf(222):  0.000s, sat  ,     6 confl,       8 dec,      2 nd, 236 basecl, 6 learnts
   rutgers.edu/par8-4.cnf(223):  0.005s, sat  ,   354 confl,     529 dec,     12 nd, 784 basecl, 340 learnts
 rutgers.edu/par8-5-c.cnf(224):  0.000s, sat  ,     6 confl,      11 dec,      2 nd, 268 basecl, 6 learnts
```

```
          rutgers.edu/par8-5.cnf(225):  0.005s, sat  ,   352 confl,     529 dec,      8 nd, 816 basecl, 332 learnts
       rutgers.edu/pret150_25.cnf(226):  0.002s, unsat,   401 confl,    2101 dec,     60 nd, 400 basecl, 397 learnts
       rutgers.edu/pret150_40.cnf(227):  0.003s, unsat,   442 confl,    2347 dec,     60 nd, 400 basecl, 438 learnts
       rutgers.edu/pret150_60.cnf(228):  0.003s, unsat,   406 confl,    2283 dec,     63 nd, 400 basecl, 401 learnts
       rutgers.edu/pret150_75.cnf(229):  0.003s, unsat,   437 confl,    2344 dec,     65 nd, 400 basecl, 431 learnts
        rutgers.edu/pret60_25.cnf(230):  0.001s, unsat,   197 confl,     470 dec,     12 nd, 160 basecl, 192 learnts
        rutgers.edu/pret60_40.cnf(231):  0.000s, unsat,   168 confl,     444 dec,     11 nd, 160 basecl, 162 learnts
        rutgers.edu/pret60_60.cnf(232):  0.001s, unsat,   214 confl,     524 dec,     12 nd, 160 basecl, 211 learnts
        rutgers.edu/pret60_75.cnf(233):  0.001s, unsat,   190 confl,     449 dec,     11 nd, 160 basecl, 190 learnts
      rutgers.edu/ssa0432-003.cnf(234):  0.000s, unsat,    40 confl,      85 dec,      4 nd, 245 basecl, 31 learnts
      rutgers.edu/ssa2670-130.cnf(235):  0.003s, unsat,   126 confl,     258 dec,      4 nd, 863 basecl, 115 learnts
      rutgers.edu/ssa2670-141.cnf(236):  0.004s, unsat,   274 confl,     439 dec,      4 nd, 469 basecl, 259 learnts
      rutgers.edu/ssa6288-047.cnf(237):  0.002s, unsat,     2 confl,      21 dec,      1 nd, 16516 basecl, 0 learnts
      rutgers.edu/ssa7552-038.cnf(238):  0.001s, sat  ,    15 confl,     161 dec,      5 nd, 947 basecl, 14 learnts
      rutgers.edu/ssa7552-158.cnf(239):  0.002s, sat  ,    19 confl,     196 dec,     11 nd, 681 basecl, 4 learnts
      rutgers.edu/ssa7552-159.cnf(240):  0.001s, sat  ,    11 confl,     110 dec,      7 nd, 681 basecl, 3 learnts
      rutgers.edu/ssa7552-160.cnf(241):  0.002s, sat  ,    23 confl,     217 dec,      8 nd, 683 basecl, 20 learnts
224 solved, 18 unsolved
time for all: 228 sec 127 msec, 9409248 conflicts
```

## 4.5 rutgers.edu one solution Clasp

```
rutgers.edu/samename/jnh8.cnf(0): 0.003s
rutgers.edu/samename/par16-1.cnf(1): 0.014s
rutgers.edu/aim-100-1_6-no-1.cnf(2): 0.000s
rutgers.edu/aim-100-1_6-no-2.cnf(3): 0.000s
rutgers.edu/aim-100-1_6-no-3.cnf(4): 0.000s
rutgers.edu/aim-100-1_6-no-4.cnf(5): 0.000s
rutgers.edu/aim-100-1_6-yes1-1.cnf(6): 0.000s
rutgers.edu/aim-100-1_6-yes1-2.cnf(7): 0.000s
rutgers.edu/aim-100-1_6-yes1-3.cnf(8): 0.000s
rutgers.edu/aim-100-1_6-yes1-4.cnf(9): 0.000s
rutgers.edu/aim-100-2_0-no-1.cnf(10): 0.000s
rutgers.edu/aim-100-2_0-no-2.cnf(11): 0.000s
rutgers.edu/aim-100-2_0-no-3.cnf(12): 0.000s
rutgers.edu/aim-100-2_0-no-4.cnf(13): 0.000s
rutgers.edu/aim-100-2_0-yes1-1.cnf(14): 0.000s
rutgers.edu/aim-100-2_0-yes1-2.cnf(15): 0.000s
rutgers.edu/aim-100-2_0-yes1-3.cnf(16): 0.000s
rutgers.edu/aim-100-2_0-yes1-4.cnf(17): 0.000s
rutgers.edu/aim-100-3_4-yes1-1.cnf(18): 0.000s
rutgers.edu/aim-100-3_4-yes1-2.cnf(19): 0.000s
rutgers.edu/aim-100-3_4-yes1-3.cnf(20): 0.000s
rutgers.edu/aim-100-3_4-yes1-4.cnf(21): 0.000s
rutgers.edu/aim-100-6_0-yes1-1.cnf(22): 0.000s
rutgers.edu/aim-100-6_0-yes1-2.cnf(23): 0.000s
rutgers.edu/aim-100-6_0-yes1-3.cnf(24): 0.000s
rutgers.edu/aim-100-6_0-yes1-4.cnf(25): 0.000s
rutgers.edu/aim-200-1_6-no-1.cnf(26): 0.000s
rutgers.edu/aim-200-1_6-no-2.cnf(27): 0.000s
rutgers.edu/aim-200-1_6-no-3.cnf(28): 0.000s
rutgers.edu/aim-200-1_6-no-4.cnf(29): 0.000s
rutgers.edu/aim-200-1_6-yes1-1.cnf(30): 0.000s
rutgers.edu/aim-200-1_6-yes1-2.cnf(31): 0.000s
rutgers.edu/aim-200-1_6-yes1-3.cnf(32): 0.000s
rutgers.edu/aim-200-1_6-yes1-4.cnf(33): 0.000s
rutgers.edu/aim-200-2_0-no-1.cnf(34): 0.000s
rutgers.edu/aim-200-2_0-no-2.cnf(35): 0.000s
rutgers.edu/aim-200-2_0-no-3.cnf(36): 0.000s
rutgers.edu/aim-200-2_0-no-4.cnf(37): 0.000s
rutgers.edu/aim-200-2_0-yes1-1.cnf(38): 0.001s
rutgers.edu/aim-200-2_0-yes1-2.cnf(39): 0.000s
rutgers.edu/aim-200-2_0-yes1-3.cnf(40): 0.000s
rutgers.edu/aim-200-2_0-yes1-4.cnf(41): 0.001s
rutgers.edu/aim-200-3_4-yes1-1.cnf(42): 0.001s
rutgers.edu/aim-200-3_4-yes1-2.cnf(43): 0.001s
rutgers.edu/aim-200-3_4-yes1-3.cnf(44): 0.001s
rutgers.edu/aim-200-3_4-yes1-4.cnf(45): 0.001s
rutgers.edu/aim-200-6_0-yes1-1.cnf(46): 0.001s
rutgers.edu/aim-200-6_0-yes1-2.cnf(47): 0.001s
rutgers.edu/aim-200-6_0-yes1-3.cnf(48): 0.001s
rutgers.edu/aim-200-6_0-yes1-4.cnf(49): 0.001s
rutgers.edu/aim-50-1_6-no-1.cnf(50): 0.000s
rutgers.edu/aim-50-1_6-no-2.cnf(51): 0.000s
rutgers.edu/aim-50-1_6-no-3.cnf(52): 0.000s
rutgers.edu/aim-50-1_6-no-4.cnf(53): 0.000s
rutgers.edu/aim-50-1_6-yes1-1.cnf(54): 0.000s
rutgers.edu/aim-50-1_6-yes1-2.cnf(55): 0.000s
rutgers.edu/aim-50-1_6-yes1-3.cnf(56): 0.000s
rutgers.edu/aim-50-1_6-yes1-4.cnf(57): 0.000s
rutgers.edu/aim-50-2_0-no-1.cnf(58): 0.000s
rutgers.edu/aim-50-2_0-no-2.cnf(59): 0.000s
rutgers.edu/aim-50-2_0-no-3.cnf(60): 0.000s
rutgers.edu/aim-50-2_0-no-4.cnf(61): 0.000s
rutgers.edu/aim-50-2_0-yes1-1.cnf(62): 0.000s
rutgers.edu/aim-50-2_0-yes1-2.cnf(63): 0.000s
rutgers.edu/aim-50-2_0-yes1-3.cnf(64): 0.000s
rutgers.edu/aim-50-2_0-yes1-4.cnf(65): 0.000s
rutgers.edu/aim-50-3_4-yes1-1.cnf(66): 0.000s
rutgers.edu/aim-50-3_4-yes1-2.cnf(67): 0.000s
rutgers.edu/aim-50-3_4-yes1-3.cnf(68): 0.000s
rutgers.edu/aim-50-3_4-yes1-4.cnf(69): 0.000s
rutgers.edu/aim-50-6_0-yes1-1.cnf(70): 0.000s
rutgers.edu/aim-50-6_0-yes1-2.cnf(71): 0.000s
rutgers.edu/aim-50-6_0-yes1-3.cnf(72): 0.000s
rutgers.edu/aim-50-6_0-yes1-4.cnf(73): 0.000s
rutgers.edu/bf0432-007.cnf(74): 0.005s
rutgers.edu/bf1355-075.cnf(75): 0.004s
rutgers.edu/bf1355-638.cnf(76): 0.005s
rutgers.edu/bf2670-001.cnf(77): 0.002s
rutgers.edu/dubois100.cnf(78): 0.000s
rutgers.edu/dubois20.cnf(79): 0.000s
rutgers.edu/dubois21.cnf(80): 0.000s
rutgers.edu/dubois22.cnf(81): 0.000s
rutgers.edu/dubois23.cnf(82): 0.000s
rutgers.edu/dubois24.cnf(83): 0.000s
rutgers.edu/dubois25.cnf(84): 0.000s
rutgers.edu/dubois26.cnf(85): 0.000s
rutgers.edu/dubois27.cnf(86): 0.000s
rutgers.edu/dubois28.cnf(87): 0.000s
rutgers.edu/dubois29.cnf(88): 0.000s
rutgers.edu/dubois30.cnf(89): 0.000s
rutgers.edu/dubois50.cnf(90): 0.001s
rutgers.edu/f1000.cnf(91): expired
rutgers.edu/f2000.cnf(92): expired
rutgers.edu/f600.cnf(93): expired
rutgers.edu/g125.17.cnf(94): expired
rutgers.edu/g125.18.cnf(95): expired
rutgers.edu/g250.15.cnf(96): 24.935s
rutgers.edu/g250.29.cnf(97): expired
rutgers.edu/hanoi4.cnf(98): 0.036s
rutgers.edu/hanoi5.cnf(99): 2.183s
rutgers.edu/hole10.cnf(100): expired
rutgers.edu/hole6.cnf(101): 0.002s
rutgers.edu/hole7.cnf(102): 0.013s
rutgers.edu/hole8.cnf(103): 0.126s
rutgers.edu/hole9.cnf(104): 2.099s
rutgers.edu/ii16a1.cnf(105): 0.011s
rutgers.edu/ii16a2.cnf(106): 0.035s
rutgers.edu/ii16b1.cnf(107): 0.023s
rutgers.edu/ii16b2.cnf(108): 0.023s
rutgers.edu/ii16c1.cnf(109): 0.023s
rutgers.edu/ii16c2.cnf(110): 0.032s
rutgers.edu/ii16d1.cnf(111): 0.009s
rutgers.edu/ii16d2.cnf(112): 0.023s
rutgers.edu/ii16e1.cnf(113): 0.009s
rutgers.edu/ii16e2.cnf(114): 0.015s
rutgers.edu/ii32a1.cnf(115): 0.007s
rutgers.edu/ii32b1.cnf(116): 0.002s
rutgers.edu/ii32b2.cnf(117): 0.004s
rutgers.edu/ii32b3.cnf(118): 0.007s
rutgers.edu/ii32b4.cnf(119): 0.007s
rutgers.edu/ii32c1.cnf(120): 0.002s
rutgers.edu/ii32c2.cnf(121): 0.003s
rutgers.edu/ii32c3.cnf(122): 0.004s
rutgers.edu/ii32c4.cnf(123): 0.036s
rutgers.edu/ii32d1.cnf(124): 0.003s
rutgers.edu/ii32d2.cnf(125): 0.008s
rutgers.edu/ii32d3.cnf(126): 0.143s
rutgers.edu/ii32e1.cnf(127): 0.002s
rutgers.edu/ii32e2.cnf(128): 0.002s
rutgers.edu/ii32e3.cnf(129): 0.004s
rutgers.edu/ii32e4.cnf(130): 0.009s
rutgers.edu/ii32e5.cnf(131): 0.011s
rutgers.edu/ii8a1.cnf(132): 0.000s
rutgers.edu/ii8a2.cnf(133): 0.000s
rutgers.edu/ii8a3.cnf(134): 0.001s
rutgers.edu/ii8a4.cnf(135): 0.002s
rutgers.edu/ii8b1.cnf(136): 0.001s
rutgers.edu/ii8b2.cnf(137): 0.002s
rutgers.edu/ii8b3.cnf(138): 0.003s
rutgers.edu/ii8b4.cnf(139): 0.004s
rutgers.edu/ii8c1.cnf(140): 0.002s
rutgers.edu/ii8c2.cnf(141): 0.003s
rutgers.edu/ii8d1.cnf(142): 0.002s
rutgers.edu/ii8d2.cnf(143): 0.004s
rutgers.edu/ii8e1.cnf(144): 0.002s
rutgers.edu/ii8e2.cnf(145): 0.003s
rutgers.edu/jnh10.cnf(146): 0.001s
rutgers.edu/jnh11.cnf(147): 0.001s
rutgers.edu/jnh12.cnf(148): 0.001s
rutgers.edu/jnh13.cnf(149): 0.001s
rutgers.edu/jnh14.cnf(150): 0.001s
rutgers.edu/jnh15.cnf(151): 0.001s
rutgers.edu/jnh16.cnf(152): 0.003s
rutgers.edu/jnh17.cnf(153): 0.001s
rutgers.edu/jnh18.cnf(154): 0.001s
rutgers.edu/jnh19.cnf(155): 0.001s
```

rutgers.edu/jnh1.cnf(156): 0.001s
rutgers.edu/jnh201.cnf(157): 0.001s
rutgers.edu/jnh202.cnf(158): 0.001s
rutgers.edu/jnh203.cnf(159): 0.001s
rutgers.edu/jnh204.cnf(160): 0.001s
rutgers.edu/jnh205.cnf(161): 0.001s
rutgers.edu/jnh206.cnf(162): 0.001s
rutgers.edu/jnh207.cnf(163): 0.001s
rutgers.edu/jnh208.cnf(164): 0.001s
rutgers.edu/jnh209.cnf(165): 0.001s
rutgers.edu/jnh20.cnf(166): 0.001s
rutgers.edu/jnh210.cnf(167): 0.001s
rutgers.edu/jnh211.cnf(168): 0.001s
rutgers.edu/jnh212.cnf(169): 0.001s
rutgers.edu/jnh213.cnf(170): 0.001s
rutgers.edu/jnh214.cnf(171): 0.001s
rutgers.edu/jnh215.cnf(172): 0.001s
rutgers.edu/jnh216.cnf(173): 0.001s
rutgers.edu/jnh217.cnf(174): 0.001s
rutgers.edu/jnh218.cnf(175): 0.001s
rutgers.edu/jnh219.cnf(176): 0.001s
rutgers.edu/jnh220.cnf(177): 0.001s
rutgers.edu/jnh2.cnf(178): 0.001s
rutgers.edu/jnh301.cnf(179): 0.001s
rutgers.edu/jnh302.cnf(180): 0.001s
rutgers.edu/jnh303.cnf(181): 0.001s
rutgers.edu/jnh304.cnf(182): 0.001s
rutgers.edu/jnh305.cnf(183): 0.001s
rutgers.edu/jnh306.cnf(184): 0.001s
rutgers.edu/jnh307.cnf(185): 0.001s
rutgers.edu/jnh308.cnf(186): 0.001s
rutgers.edu/jnh309.cnf(187): 0.001s
rutgers.edu/jnh310.cnf(188): 0.001s
rutgers.edu/jnh3.cnf(189): 0.001s
rutgers.edu/jnh4.cnf(190): 0.001s
rutgers.edu/jnh5.cnf(191): 0.001s
rutgers.edu/jnh6.cnf(192): 0.001s
rutgers.edu/jnh7.cnf(193): 0.001s
rutgers.edu/jnh8.cnf(194): 0.001s
rutgers.edu/jnh9.cnf(195): 0.001s
rutgers.edu/par16-1-c.cnf(196): 0.015s
rutgers.edu/par16-1.cnf(197): 0.014s
rutgers.edu/par16-2-c.cnf(198): 0.055s
rutgers.edu/par16-2.cnf(199): 0.045s
rutgers.edu/par16-3-c.cnf(200): 0.079s
rutgers.edu/par16-3.cnf(201): 0.047s
rutgers.edu/par16-4-c.cnf(202): 0.010s
rutgers.edu/par16-4.cnf(203): 0.017s
rutgers.edu/par16-5-c.cnf(204): 0.038s
rutgers.edu/par16-5.cnf(205): 0.040s
rutgers.edu/par32-1-c.cnf(206): expired
rutgers.edu/par32-1.cnf(207): expired
rutgers.edu/par32-2-c.cnf(208): expired
rutgers.edu/par32-2.cnf(209): expired
rutgers.edu/par32-3-c.cnf(210): expired
rutgers.edu/par32-3.cnf(211): expired
rutgers.edu/par32-4-c.cnf(212): expired
rutgers.edu/par32-4.cnf(213): expired
rutgers.edu/par32-5-c.cnf(214): expired
rutgers.edu/par32-5.cnf(215): expired
rutgers.edu/par8-1-c.cnf(216): 0.000s
rutgers.edu/par8-1.cnf(217): 0.001s
rutgers.edu/par8-2-c.cnf(218): 0.000s
rutgers.edu/par8-2.cnf(219): 0.001s
rutgers.edu/par8-3-c.cnf(220): 0.000s
rutgers.edu/par8-3.cnf(221): 0.001s
rutgers.edu/par8-4-c.cnf(222): 0.000s
rutgers.edu/par8-4.cnf(223): 0.001s
rutgers.edu/par8-5-c.cnf(224): 0.000s
rutgers.edu/par8-5.cnf(225): 0.001s
rutgers.edu/pret150_25.cnf(226): 0.000s
rutgers.edu/pret150_40.cnf(227): 0.000s
rutgers.edu/pret150_60.cnf(228): 0.000s
rutgers.edu/pret150_75.cnf(229): 0.000s
rutgers.edu/pret60_25.cnf(230): 0.000s
rutgers.edu/pret60_40.cnf(231): 0.000s
rutgers.edu/pret60_60.cnf(232): 0.000s
rutgers.edu/pret60_75.cnf(233): 0.000s
rutgers.edu/ssa0432-003.cnf(234): 0.001s
rutgers.edu/ssa2670-130.cnf(235): 0.003s
rutgers.edu/ssa2670-141.cnf(236): 0.002s
rutgers.edu/ssa6288-047.cnf(237): 0.010s
rutgers.edu/ssa7552-038.cnf(238): 0.002s
rutgers.edu/ssa7552-158.cnf(239): 0.002s
rutgers.edu/ssa7552-159.cnf(240): 0.002s
rutgers.edu/ssa7552-160.cnf(241): 0.002s

## 4.6 rutgers.edu one solution ZChaff

```
rutgers.edu/samename/jnh8.cnf(0): 0.000957
   rutgers.edu/samename/par16-1.cnf(1): 0.147459
   rutgers.edu/aim-100-1_6-no-1.cnf(2): 0.000141
   rutgers.edu/aim-100-1_6-no-2.cnf(3): 0.000175
   rutgers.edu/aim-100-1_6-no-3.cnf(4): 0.000165
   rutgers.edu/aim-100-1_6-no-4.cnf(5): 0.00012
 rutgers.edu/aim-100-1_6-yes1-1.cnf(6): 0.000165
 rutgers.edu/aim-100-1_6-yes1-2.cnf(7): 0.000123
 rutgers.edu/aim-100-1_6-yes1-3.cnf(8): 0.000181
 rutgers.edu/aim-100-1_6-yes1-4.cnf(9): 0.0001
   rutgers.edu/aim-100-2_0-no-1.cnf(10): 9.9e-05
   rutgers.edu/aim-100-2_0-no-2.cnf(11): 0.000119
   rutgers.edu/aim-100-2_0-no-3.cnf(12): 6.6e-05
   rutgers.edu/aim-100-2_0-no-4.cnf(13): 7.6e-05
 rutgers.edu/aim-100-2_0-yes1-1.cnf(14): 0.000248
 rutgers.edu/aim-100-2_0-yes1-2.cnf(15): 0.000183
 rutgers.edu/aim-100-2_0-yes1-3.cnf(16): 0.000218
 rutgers.edu/aim-100-2_0-yes1-4.cnf(17): 0.000169
 rutgers.edu/aim-100-3_4-yes1-1.cnf(18): 0.000666
 rutgers.edu/aim-100-3_4-yes1-2.cnf(19): 0.00051
 rutgers.edu/aim-100-3_4-yes1-3.cnf(20): 0.000319
 rutgers.edu/aim-100-3_4-yes1-4.cnf(21): 0.000514
 rutgers.edu/aim-100-6_0-yes1-1.cnf(22): 0.000194
 rutgers.edu/aim-100-6_0-yes1-2.cnf(23): 8.8e-05
 rutgers.edu/aim-100-6_0-yes1-3.cnf(24): 0.000182
 rutgers.edu/aim-100-6_0-yes1-4.cnf(25): 8.3e-05
   rutgers.edu/aim-200-1_6-no-1.cnf(26): 0.000279
   rutgers.edu/aim-200-1_6-no-2.cnf(27): 0.000412
   rutgers.edu/aim-200-1_6-no-3.cnf(28): 0.000558
   rutgers.edu/aim-200-1_6-no-4.cnf(29): 0.000203
 rutgers.edu/aim-200-1_6-yes1-1.cnf(30): 0.000217
 rutgers.edu/aim-200-1_6-yes1-2.cnf(31): 0.00054
 rutgers.edu/aim-200-1_6-yes1-3.cnf(32): 0.000224
 rutgers.edu/aim-200-1_6-yes1-4.cnf(33): 0.000305
   rutgers.edu/aim-200-2_0-no-1.cnf(34): 0.000191
   rutgers.edu/aim-200-2_0-no-2.cnf(35): 0.000294
   rutgers.edu/aim-200-2_0-no-3.cnf(36): 0.000363
   rutgers.edu/aim-200-2_0-no-4.cnf(37): 0.000206
 rutgers.edu/aim-200-2_0-yes1-1.cnf(38): 0.000768
 rutgers.edu/aim-200-2_0-yes1-2.cnf(39): 0.000868
 rutgers.edu/aim-200-2_0-yes1-3.cnf(40): 0.000409
 rutgers.edu/aim-200-2_0-yes1-4.cnf(41): 0.000664
 rutgers.edu/aim-200-3_4-yes1-1.cnf(42): 0.001706
 rutgers.edu/aim-200-3_4-yes1-2.cnf(43): 0.001011
 rutgers.edu/aim-200-3_4-yes1-3.cnf(44): 0.000765
 rutgers.edu/aim-200-3_4-yes1-4.cnf(45): 0.000612
 rutgers.edu/aim-200-6_0-yes1-1.cnf(46): 0.000432
 rutgers.edu/aim-200-6_0-yes1-2.cnf(47): 0.000902
 rutgers.edu/aim-200-6_0-yes1-3.cnf(48): 0.002085
 rutgers.edu/aim-200-6_0-yes1-4.cnf(49): 0.002686
    rutgers.edu/aim-50-1_6-no-1.cnf(50): 8.9e-05
   rutgers.edu/aim-50-1_6-no-2.cnf(51): 7.9e-05
   rutgers.edu/aim-50-1_6-no-3.cnf(52): 9.4e-05
   rutgers.edu/aim-50-1_6-no-4.cnf(53): 5.5e-05
 rutgers.edu/aim-50-1_6-yes1-1.cnf(54): 8.4e-05
 rutgers.edu/aim-50-1_6-yes1-2.cnf(55): 6.2e-05
 rutgers.edu/aim-50-1_6-yes1-3.cnf(56): 6.9e-05
 rutgers.edu/aim-50-1_6-yes1-4.cnf(57): 6e-05
   rutgers.edu/aim-50-2_0-no-1.cnf(58): 7.7e-05
   rutgers.edu/aim-50-2_0-no-2.cnf(59): 0.000167
   rutgers.edu/aim-50-2_0-no-3.cnf(60): 6.7e-05
   rutgers.edu/aim-50-2_0-no-4.cnf(61): 5.9e-05
 rutgers.edu/aim-50-2_0-yes1-1.cnf(62): 8.9e-05
 rutgers.edu/aim-50-2_0-yes1-2.cnf(63): 0.00015
 rutgers.edu/aim-50-2_0-yes1-3.cnf(64): 9.7e-05
 rutgers.edu/aim-50-2_0-yes1-4.cnf(65): 5.7e-05
 rutgers.edu/aim-50-3_4-yes1-1.cnf(66): 0.000168
 rutgers.edu/aim-50-3_4-yes1-2.cnf(67): 7.5e-05
 rutgers.edu/aim-50-3_4-yes1-3.cnf(68): 7.5e-05
 rutgers.edu/aim-50-3_4-yes1-4.cnf(69): 5.8e-05
 rutgers.edu/aim-50-6_0-yes1-1.cnf(70): 7.9e-05
 rutgers.edu/aim-50-6_0-yes1-2.cnf(71): 2.8e-05
 rutgers.edu/aim-50-6_0-yes1-3.cnf(72): 3.2e-05
 rutgers.edu/aim-50-6_0-yes1-4.cnf(73): 8e-05
        rutgers.edu/bf0432-007.cnf(74): 0.012037
        rutgers.edu/bf1355-075.cnf(75): 0.000569
        rutgers.edu/bf1355-638.cnf(76): 0.000496
        rutgers.edu/bf2670-001.cnf(77): 0.000282
         rutgers.edu/dubois100.cnf(78): 8.6e-05
          rutgers.edu/dubois20.cnf(79): 0.000383
          rutgers.edu/dubois21.cnf(80): 0.00045
          rutgers.edu/dubois22.cnf(81): 0.000492
          rutgers.edu/dubois23.cnf(82): 0.00051
          rutgers.edu/dubois24.cnf(83): 0.000611
          rutgers.edu/dubois25.cnf(84): 0.000511
          rutgers.edu/dubois26.cnf(85): 0.000509
          rutgers.edu/dubois27.cnf(86): 0.000571
          rutgers.edu/dubois28.cnf(87): 0.000436
          rutgers.edu/dubois29.cnf(88): 0.000514
          rutgers.edu/dubois30.cnf(89): 0.000575
          rutgers.edu/dubois50.cnf(90): 0.00099
             rutgers.edu/f1000.cnf(91): expired
             rutgers.edu/f2000.cnf(92): expired
              rutgers.edu/f600.cnf(93): expired
           rutgers.edu/g125.17.cnf(94): expired
           rutgers.edu/g125.18.cnf(95): expired
           rutgers.edu/g250.15.cnf(96): expired
           rutgers.edu/g250.29.cnf(97): expired
            rutgers.edu/hanoi4.cnf(98): 0.108652
            rutgers.edu/hanoi5.cnf(99): 18.8458
           rutgers.edu/hole10.cnf(100): 1.59025
            rutgers.edu/hole6.cnf(101): 0.002563
```

```
 rutgers.edu/hole7.cnf(102): 0.010485
 rutgers.edu/hole8.cnf(103): 0.055815
 rutgers.edu/hole9.cnf(104): 0.336355
rutgers.edu/ii16a1.cnf(105): 0.005567
rutgers.edu/ii16a2.cnf(106): 0.001509
rutgers.edu/ii16b1.cnf(107): 0.013599
rutgers.edu/ii16b2.cnf(108): 0.134517
rutgers.edu/ii16c1.cnf(109): 0.004926
rutgers.edu/ii16c2.cnf(110): 0.008139
rutgers.edu/ii16d1.cnf(111): 0.003713
rutgers.edu/ii16d2.cnf(112): 0.007424
rutgers.edu/ii16e1.cnf(113): 0.028829
rutgers.edu/ii16e2.cnf(114): 0.004378
rutgers.edu/ii32a1.cnf(115): 0.000545
rutgers.edu/ii32b1.cnf(116): 9.2e-05
rutgers.edu/ii32b2.cnf(117): 0.000159
rutgers.edu/ii32b3.cnf(118): 0.000277
rutgers.edu/ii32b4.cnf(119): 0.04098
rutgers.edu/ii32c1.cnf(120): 6.9e-05
rutgers.edu/ii32c2.cnf(121): 0.000147
rutgers.edu/ii32c3.cnf(122): 0.00244
rutgers.edu/ii32c4.cnf(123): 0.001601
rutgers.edu/ii32d1.cnf(124): 0.001302
rutgers.edu/ii32d2.cnf(125): 0.000311
rutgers.edu/ii32d3.cnf(126): 0.021972
rutgers.edu/ii32e1.cnf(127): 8.7e-05
rutgers.edu/ii32e2.cnf(128): 8.7e-05
rutgers.edu/ii32e3.cnf(129): 0.000314
rutgers.edu/ii32e4.cnf(130): 0.000616
rutgers.edu/ii32e5.cnf(131): 0.000771
 rutgers.edu/ii8a1.cnf(132): 3.6e-05
 rutgers.edu/ii8a2.cnf(133): 0.000152
 rutgers.edu/ii8a3.cnf(134): 0.00018
 rutgers.edu/ii8a4.cnf(135): 0.000129
 rutgers.edu/ii8b1.cnf(136): 0.000155
 rutgers.edu/ii8b2.cnf(137): 0.00028
 rutgers.edu/ii8b3.cnf(138): 0.000369
 rutgers.edu/ii8b4.cnf(139): 0.00054
 rutgers.edu/ii8c1.cnf(140): 0.000261
 rutgers.edu/ii8c2.cnf(141): 0.00081
 rutgers.edu/ii8d1.cnf(142): 0.000645
 rutgers.edu/ii8d2.cnf(143): 0.000786
 rutgers.edu/ii8e1.cnf(144): 0.000564
 rutgers.edu/ii8e2.cnf(145): 0.000811
 rutgers.edu/jnh10.cnf(146): 0.000212
 rutgers.edu/jnh11.cnf(147): 0.000556
 rutgers.edu/jnh12.cnf(148): 0.000184
 rutgers.edu/jnh13.cnf(149): 0.000118
 rutgers.edu/jnh14.cnf(150): 0.000166
 rutgers.edu/jnh15.cnf(151): 0.000535
 rutgers.edu/jnh16.cnf(152): 0.008676
 rutgers.edu/jnh17.cnf(153): 0.000387
 rutgers.edu/jnh18.cnf(154): 0.000451
 rutgers.edu/jnh19.cnf(155): 0.00038
  rutgers.edu/jnh1.cnf(156): 0.000338
rutgers.edu/jnh201.cnf(157): 6.2e-05
rutgers.edu/jnh202.cnf(158): 9.3e-05
rutgers.edu/jnh203.cnf(159): 0.000352
rutgers.edu/jnh204.cnf(160): 0.000689
rutgers.edu/jnh205.cnf(161): 0.000166
rutgers.edu/jnh206.cnf(162): 0.000923
rutgers.edu/jnh207.cnf(163): 0.000797
rutgers.edu/jnh208.cnf(164): 0.0007
rutgers.edu/jnh209.cnf(165): 0.00054
 rutgers.edu/jnh20.cnf(166): 0.000213
rutgers.edu/jnh210.cnf(167): 0.000199
rutgers.edu/jnh211.cnf(168): 0.000192
rutgers.edu/jnh212.cnf(169): 0.00018
rutgers.edu/jnh213.cnf(170): 0.000151
rutgers.edu/jnh214.cnf(171): 0.000258
rutgers.edu/jnh215.cnf(172): 0.000288
rutgers.edu/jnh216.cnf(173): 0.000482
rutgers.edu/jnh217.cnf(174): 0.000333
rutgers.edu/jnh218.cnf(175): 0.000239
rutgers.edu/jnh219.cnf(176): 0.000517
rutgers.edu/jnh220.cnf(177): 0.001144
  rutgers.edu/jnh2.cnf(178): 0.000189
rutgers.edu/jnh301.cnf(179): 0.000364
rutgers.edu/jnh302.cnf(180): 0.000205
rutgers.edu/jnh303.cnf(181): 0.000464
rutgers.edu/jnh304.cnf(182): 9.9e-05
rutgers.edu/jnh305.cnf(183): 0.00026
rutgers.edu/jnh306.cnf(184): 0.00142
rutgers.edu/jnh307.cnf(185): 0.000152
rutgers.edu/jnh308.cnf(186): 0.000422
rutgers.edu/jnh309.cnf(187): 0.000225
rutgers.edu/jnh310.cnf(188): 4.5e-05
  rutgers.edu/jnh3.cnf(189): 0.001054
  rutgers.edu/jnh4.cnf(190): 0.000395
  rutgers.edu/jnh5.cnf(191): 0.000336
  rutgers.edu/jnh6.cnf(192): 0.000655
  rutgers.edu/jnh7.cnf(193): 0.000256
  rutgers.edu/jnh8.cnf(194): 0.000243
  rutgers.edu/jnh9.cnf(195): 0.00025
rutgers.edu/par16-1-c.cnf(196): 0.109648
  rutgers.edu/par16-1.cnf(197): 0.146728
rutgers.edu/par16-2-c.cnf(198): 0.166556
  rutgers.edu/par16-2.cnf(199): 0.234596
rutgers.edu/par16-3-c.cnf(200): 0.038175
  rutgers.edu/par16-3.cnf(201): 0.085406
rutgers.edu/par16-4-c.cnf(202): 0.001629
  rutgers.edu/par16-4.cnf(203): 0.141601
rutgers.edu/par16-5-c.cnf(204): 0.135668
  rutgers.edu/par16-5.cnf(205): 0.280133
rutgers.edu/par32-1-c.cnf(206): expired
  rutgers.edu/par32-1.cnf(207): expired
rutgers.edu/par32-2-c.cnf(208): expired
  rutgers.edu/par32-2.cnf(209): expired
rutgers.edu/par32-3-c.cnf(210): expired
  rutgers.edu/par32-3.cnf(211): expired
rutgers.edu/par32-4-c.cnf(212): expired
  rutgers.edu/par32-4.cnf(213): expired
rutgers.edu/par32-5-c.cnf(214): expired
  rutgers.edu/par32-5.cnf(215): expired
 rutgers.edu/par8-1-c.cnf(216): 4.6e-05
   rutgers.edu/par8-1.cnf(217): 8.1e-05
 rutgers.edu/par8-2-c.cnf(218): 0.0001
   rutgers.edu/par8-2.cnf(219): 0.000154
 rutgers.edu/par8-3-c.cnf(220): 6.1e-05
   rutgers.edu/par8-3.cnf(221): 0.000158
 rutgers.edu/par8-4-c.cnf(222): 5.1e-05
   rutgers.edu/par8-4.cnf(223): 0.000141
 rutgers.edu/par8-5-c.cnf(224): 5.7e-05
   rutgers.edu/par8-5.cnf(225): 8.9e-05
rutgers.edu/pret150_25.cnf(226): 0.014175
rutgers.edu/pret150_40.cnf(227): 0.017529
rutgers.edu/pret150_60.cnf(228): 0.014205
rutgers.edu/pret150_75.cnf(229): 0.01294
 rutgers.edu/pret60_25.cnf(230): 0.002779
 rutgers.edu/pret60_40.cnf(231): 0.003113
 rutgers.edu/pret60_60.cnf(232): 0.00261
 rutgers.edu/pret60_75.cnf(233): 0.004504
rutgers.edu/ssa0432-003.cnf(234): 0.000332
rutgers.edu/ssa2670-130.cnf(235): 0.001641
rutgers.edu/ssa2670-141.cnf(236): 0.002331
rutgers.edu/ssa6288-047.cnf(237): 0.000616
rutgers.edu/ssa7552-038.cnf(238): 0.000411
rutgers.edu/ssa7552-158.cnf(239): 0.000209
rutgers.edu/ssa7552-159.cnf(240): 0.000211
rutgers.edu/ssa7552-160.cnf(241): 0.000288
```

## 4.7 DQMR DualSat

```
> ./run --bench dualsat.c --timeout 60 cachet-all.lis -1
cachet/DQMR/qmr-50/or-50-20-1-UC-20.cnf(0): 28.389s, sat  ,    23 confl, 14716982 dec, 14716942 nd
cachet/DQMR/qmr-50/or-50-5-8-UC-30.cnf(1):  0.023s, sat  ,    33 confl,   18838 dec,  18763 nd
  cachet/DQMR/qmr-50/or-50-20-10.cnf(2): expired
cachet/DQMR/qmr-50/or-50-5-5-UC-10.cnf(3): expired
cachet/DQMR/qmr-50/or-50-10-8-UC-40.cnf(4):  0.000s, sat  ,    40 confl,     195 dec,    130 nd
   cachet/DQMR/qmr-50/or-50-20-7.cnf(5): expired
cachet/DQMR/qmr-50/or-50-20-10-UC-10.cnf(6): expired
cachet/DQMR/qmr-50/or-50-10-4-UC-30.cnf(7):  1.135s, sat  ,    26 confl,  975782 dec, 975748 nd
cachet/DQMR/qmr-50/or-50-10-8-UC-30.cnf(8):  0.000s, sat  ,    39 confl,     246 dec,    193 nd
cachet/DQMR/qmr-50/or-50-10-10-UC-20.cnf(9):  0.540s, sat  ,    29 confl,  614284 dec, 614257 nd
cachet/DQMR/qmr-50/or-50-20-10-UC-40.cnf(10):  0.037s, sat  ,    40 confl,   17847 dec,  17794 nd
   cachet/DQMR/qmr-50/or-50-20-3.cnf(11): expired
cachet/DQMR/qmr-50/or-50-5-10-UC-30.cnf(12):  0.004s, sat  ,    34 confl,    3324 dec,   3266 nd
cachet/DQMR/qmr-50/or-50-10-7-UC-30.cnf(13):  0.043s, sat  ,    32 confl,   29571 dec,  29509 nd
cachet/DQMR/qmr-50/or-50-5-1-UC-30.cnf(14):  0.013s, sat  ,    33 confl,   11568 dec,  11523 nd
cachet/DQMR/qmr-50/or-50-5-6-UC-40.cnf(15):  0.000s, sat  ,    38 confl,     285 dec,    225 nd
cachet/DQMR/qmr-50/or-50-20-3-UC-30.cnf(16):  0.164s, sat  ,    26 confl,   63742 dec,  63678 nd
cachet/DQMR/qmr-50/or-50-5-2-UC-30.cnf(17):  0.024s, sat  ,    33 confl,   23114 dec,  23074 nd
   cachet/DQMR/qmr-50/or-50-20-5.cnf(18): expired
cachet/DQMR/qmr-50/or-50-20-5-UC-40.cnf(19):  0.239s, sat  ,    35 confl,   97945 dec,  97912 nd
cachet/DQMR/qmr-50/or-50-5-1-UC-10.cnf(20): expired
cachet/DQMR/qmr-50/or-50-5-9-UC-40.cnf(21):  0.002s, sat  ,    35 confl,    1913 dec,   1873 nd
   cachet/DQMR/qmr-50/or-50-20-8.cnf(22): expired
```

```
cachet/DQMR/qmr-50/or-50-10-6-UC-20.cnf(23):  4.015s, sat  ,    23 confl, 2976733 dec, 2976700 nd
cachet/DQMR/qmr-50/or-50-5-8-UC-40.cnf(24):  0.001s, sat  ,    37 confl,     837 dec,    775 nd
cachet/DQMR/qmr-50/or-50-10-8-UC-10.cnf(25): 27.081s, sat  ,    20 confl, 23152523 dec, 23152504 nd
cachet/DQMR/qmr-50/or-50-5-2-UC-40.cnf(26):  0.005s, sat  ,    36 confl,    4524 dec,   4460 nd
cachet/DQMR/qmr-50/or-50-20-3-UC-10.cnf(27): expired
cachet/DQMR/qmr-50/or-50-20-5-UC-10.cnf(28): expired
cachet/DQMR/qmr-50/or-50-20-9-UC-10.cnf(29): expired
cachet/DQMR/qmr-50/or-50-10-1-UC-30.cnf(30):  0.018s, sat  ,    36 confl,   16194 dec,  16156 nd
    cachet/DQMR/qmr-50/or-50-5-7.cnf(31): expired
cachet/DQMR/qmr-50/or-50-20-4-UC-10.cnf(32): expired
cachet/DQMR/qmr-50/or-50-5-4-UC-30.cnf(33):  4.916s, sat  ,    26 confl, 3463169 dec, 3463128 nd
cachet/DQMR/qmr-50/or-50-5-3-UC-10.cnf(34):  4.274s, sat  ,    23 confl, 3276634 dec, 3276614 nd
cachet/DQMR/qmr-50/or-50-10-5-UC-30.cnf(35): 38.490s, sat  ,    21 confl, 22541660 dec, 22541645 nd
   cachet/DQMR/qmr-50/or-50-10-9.cnf(36): expired
cachet/DQMR/qmr-50/or-50-10-9-UC-10.cnf(37): expired
cachet/DQMR/qmr-50/or-50-10-5-UC-20.cnf(38): expired
cachet/DQMR/qmr-50/or-50-5-3-UC-40.cnf(39):  0.008s, sat  ,    35 confl,    1907 dec,   1875 nd
cachet/DQMR/qmr-50/or-50-20-4-UC-20.cnf(40): expired
cachet/DQMR/qmr-50/or-50-20-8-UC-30.cnf(41):  0.613s, sat  ,    29 confl,  399058 dec, 399016 nd
cachet/DQMR/qmr-50/or-50-10-6-UC-30.cnf(42):  5.832s, sat  ,    24 confl, 3516679 dec, 3516638 nd
   cachet/DQMR/qmr-50/or-50-20-2.cnf(43): expired
cachet/DQMR/qmr-50/or-50-5-8-UC-20.cnf(44): 10.114s, sat  ,    21 confl, 6541176 dec, 6541141 nd
cachet/DQMR/qmr-50/or-50-10-9-UC-30.cnf(45):  1.481s, sat  ,    24 confl, 1019510 dec, 1019461 nd
    cachet/DQMR/qmr-50/or-50-5-6.cnf(46): expired
cachet/DQMR/qmr-50/or-50-5-7-UC-10.cnf(47): expired
cachet/DQMR/qmr-50/or-50-10-2-UC-20.cnf(48):  0.459s, sat  ,    23 confl,  242189 dec, 242150 nd
  cachet/DQMR/qmr-50/or-50-10-10.cnf(49): expired
cachet/DQMR/qmr-50/or-50-20-9-UC-20.cnf(50):  5.802s, sat  ,    23 confl, 3381185 dec, 3381162 nd
   cachet/DQMR/qmr-50/or-50-20-9.cnf(51): expired
cachet/DQMR/qmr-50/or-50-5-5-UC-30.cnf(52):  0.070s, sat  ,    31 confl,   69246 dec,  69218 nd
cachet/DQMR/qmr-50/or-50-20-3-UC-20.cnf(53):  2.034s, sat  ,    23 confl,  974538 dec, 974497 nd
cachet/DQMR/qmr-50/or-50-10-4-UC-20.cnf(54): 29.496s, sat  ,    20 confl, 22219175 dec, 22219159 nd
cachet/DQMR/qmr-50/or-50-5-2-UC-20.cnf(55): 40.118s, sat  ,    22 confl, 33713759 dec, 33713740 nd
cachet/DQMR/qmr-50/or-50-20-5-UC-20.cnf(56): expired
cachet/DQMR/qmr-50/or-50-20-1-UC-30.cnf(57):  2.043s, sat  ,    28 confl, 1120504 dec, 1120451 nd
cachet/DQMR/qmr-50/or-50-10-1-UC-20.cnf(58):  0.481s, sat  ,    30 confl,  424968 dec, 424924 nd
cachet/DQMR/qmr-50/or-50-20-6-UC-10.cnf(59): expired
cachet/DQMR/qmr-50/or-50-10-7-UC-20.cnf(60):  8.267s, sat  ,    24 confl, 5564847 dec, 5564818 nd
cachet/DQMR/qmr-50/or-50-10-10-UC-30.cnf(61):  0.034s, sat  ,    34 confl,   43044 dec,  43010 nd
    cachet/DQMR/qmr-50/or-50-5-9.cnf(62): expired
cachet/DQMR/qmr-50/or-50-20-7-UC-30.cnf(63):  2.947s, sat  ,    29 confl, 2640294 dec, 2640265 nd
   cachet/DQMR/qmr-50/or-50-10-3.cnf(64): expired
cachet/DQMR/qmr-50/or-50-10-6-UC-40.cnf(65):  0.007s, sat  ,    38 confl,    6402 dec,   6341 nd
    cachet/DQMR/qmr-50/or-50-5-3.cnf(66): expired
   cachet/DQMR/qmr-50/or-50-10-8.cnf(67): expired
cachet/DQMR/qmr-50/or-50-5-7-UC-30.cnf(68):  0.390s, sat  ,    31 confl,  381605 dec, 381578 nd
   cachet/DQMR/qmr-50/or-50-10-7.cnf(69): expired
cachet/DQMR/qmr-50/or-50-10-2-UC-30.cnf(70):  0.040s, sat  ,    26 confl,   17835 dec,  17781 nd
cachet/DQMR/qmr-50/or-50-10-3-UC-10.cnf(71): expired
cachet/DQMR/qmr-50/or-50-5-4-UC-20.cnf(72): expired
cachet/DQMR/qmr-50/or-50-10-10-UC-40.cnf(73):  0.001s, sat  ,    39 confl,    1194 dec,   1154 nd
cachet/DQMR/qmr-50/or-50-10-2-UC-40.cnf(74):  0.007s, sat  ,    32 confl,    4571 dec,   4496 nd
cachet/DQMR/qmr-50/or-50-5-1-UC-20.cnf(75):  7.320s, sat  ,    26 confl, 6430847 dec, 6430826 nd
cachet/DQMR/qmr-50/or-50-20-5-UC-30.cnf(76):  1.074s, sat  ,    28 confl,  559757 dec, 559713 nd
cachet/DQMR/qmr-50/or-50-20-2-UC-10.cnf(77): expired
cachet/DQMR/qmr-50/or-50-5-5-UC-40.cnf(78):  0.000s, sat  ,    40 confl,     263 dec,    226 nd
cachet/DQMR/qmr-50/or-50-20-2-UC-40.cnf(79):  5.118s, sat  ,    29 confl, 3456042 dec, 3456005 nd
    cachet/DQMR/qmr-50/or-50-5-4.cnf(80): expired
cachet/DQMR/qmr-50/or-50-5-10-UC-10.cnf(81): expired
cachet/DQMR/qmr-50/or-50-10-4-UC-10.cnf(82): expired
cachet/DQMR/qmr-50/or-50-5-9-UC-30.cnf(83):  0.057s, sat  ,    31 confl,   50439 dec,  50401 nd
cachet/DQMR/qmr-50/or-50-20-8-UC-20.cnf(84): 34.863s, sat  ,    19 confl, 23037080 dec, 23037065 nd
   cachet/DQMR/qmr-50/or-50-20-4.cnf(85): expired
cachet/DQMR/qmr-50/or-50-10-10-UC-10.cnf(86): expired
    cachet/DQMR/qmr-50/or-50-5-8.cnf(87): expired
cachet/DQMR/qmr-50/or-50-20-6-UC-30.cnf(88):  0.378s, sat  ,    32 confl,  188166 dec, 188121 nd
   cachet/DQMR/qmr-50/or-50-5-10.cnf(89): expired
cachet/DQMR/qmr-50/or-50-5-10-UC-20.cnf(90):  1.458s, sat  ,    25 confl, 1383096 dec, 1383072 nd
cachet/DQMR/qmr-50/or-50-10-8-UC-20.cnf(91):  0.030s, sat  ,    32 confl,   28735 dec,  28704 nd
cachet/DQMR/qmr-50/or-50-10-5-UC-40.cnf(92):  1.244s, sat  ,    22 confl,  639263 dec, 639236 nd
cachet/DQMR/qmr-50/or-50-20-2-UC-20.cnf(93): expired
cachet/DQMR/qmr-50/or-50-10-9-UC-20.cnf(94): expired
cachet/DQMR/qmr-50/or-50-5-10-UC-40.cnf(95):  0.001s, sat  ,    36 confl,     877 dec,    818 nd
cachet/DQMR/qmr-50/or-50-20-10-UC-20.cnf(96): 23.919s, sat  ,    26 confl, 12993082 dec, 12993052 nd
cachet/DQMR/qmr-50/or-50-20-2-UC-30.cnf(97): 24.369s, sat  ,    24 confl, 17644052 dec, 17644016 nd
cachet/DQMR/qmr-50/or-50-20-7-UC-20.cnf(98): 14.141s, sat  ,    26 confl, 12736226 dec, 12736200 nd
cachet/DQMR/qmr-50/or-50-10-3-UC-20.cnf(99): 39.268s, sat  ,    22 confl, 23368758 dec, 23368745 nd
cachet/DQMR/qmr-50/or-50-10-3-UC-30.cnf(100):  0.105s, sat  ,    31 confl,   69494 dec,  69452 nd
cachet/DQMR/qmr-50/or-50-5-6-UC-20.cnf(101):  0.111s, sat  ,    29 confl,   99101 dec,  99074 nd
   cachet/DQMR/qmr-50/or-50-10-6.cnf(102): expired
cachet/DQMR/qmr-50/or-50-5-8-UC-10.cnf(103): expired
cachet/DQMR/qmr-50/or-50-20-8-UC-40.cnf(104):  0.172s, sat  ,    32 confl,  104544 dec, 104504 nd
cachet/DQMR/qmr-50/or-50-5-9-UC-20.cnf(105): 20.873s, sat  ,    18 confl, 13911868 dec, 13911845 nd
cachet/DQMR/qmr-50/or-50-10-7-UC-10.cnf(106): expired
cachet/DQMR/qmr-50/or-50-5-4-UC-40.cnf(107):  0.265s, sat  ,    34 confl,  193292 dec, 193237 nd
cachet/DQMR/qmr-50/or-50-20-8-UC-10.cnf(108): expired
cachet/DQMR/qmr-50/or-50-20-1-UC-40.cnf(109):  0.070s, sat  ,    33 confl,   38888 dec,  38828 nd
   cachet/DQMR/qmr-50/or-50-20-6.cnf(110): expired
cachet/DQMR/qmr-50/or-50-10-6-UC-10.cnf(111): expired
cachet/DQMR/qmr-50/or-50-20-6-UC-20.cnf(112): 20.391s, sat  ,    25 confl, 10768891 dec, 10768875 nd
    cachet/DQMR/qmr-50/or-50-5-1.cnf(113): expired
cachet/DQMR/qmr-50/or-50-10-3-UC-40.cnf(114):  0.003s, sat  ,    33 confl,    1952 dec,   1898 nd
cachet/DQMR/qmr-50/or-50-20-7-UC-40.cnf(115):  0.060s, sat  ,    34 confl,   53039 dec,  53001 nd
cachet/DQMR/qmr-50/or-50-10-9-UC-40.cnf(116):  0.268s, sat  ,    28 confl,  162799 dec, 162753 nd
cachet/DQMR/qmr-50/or-50-5-3-UC-20.cnf(117):  0.063s, sat  ,    31 confl,   52062 dec,  52034 nd
   cachet/DQMR/qmr-50/or-50-10-2.cnf(118): expired
   cachet/DQMR/qmr-50/or-50-10-5.cnf(119): expired
cachet/DQMR/qmr-50/or-50-20-10-UC-30.cnf(120):  1.688s, sat  ,    30 confl,  960452 dec, 960396 nd
cachet/DQMR/qmr-50/or-50-20-4-UC-30.cnf(121): 42.776s, sat  ,    22 confl, 21215509 dec, 21215499 nd
cachet/DQMR/qmr-50/or-50-20-9-UC-30.cnf(122):  0.149s, sat  ,    27 confl,   77904 dec,  77870 nd
```

```
   cachet/DQMR/qmr-50/or-50-10-1.cnf(123): expired
cachet/DQMR/qmr-50/or-50-5-6-UC-10.cnf(124):  7.345s, sat  ,    21 confl, 5674247 dec, 5674228 nd
   cachet/DQMR/qmr-50/or-50-10-4.cnf(125): expired
cachet/DQMR/qmr-50/or-50-5-2-UC-10.cnf(126): expired
cachet/DQMR/qmr-50/or-50-5-6-UC-30.cnf(127):  0.003s, sat  ,    35 confl,    3122 dec,   3074 nd
cachet/DQMR/qmr-50/or-50-20-7-UC-10.cnf(128): expired
cachet/DQMR/qmr-50/or-50-5-7-UC-20.cnf(129):  6.025s, sat  ,    26 confl, 5691020 dec, 5690998 nd
   cachet/DQMR/qmr-50/or-50-20-1.cnf(130): expired
cachet/DQMR/qmr-50/or-50-10-1-UC-10.cnf(131): expired
cachet/DQMR/qmr-50/or-50-5-4-UC-10.cnf(132): expired
cachet/DQMR/qmr-50/or-50-5-1-UC-40.cnf(133):  0.001s, sat  ,    35 confl,     482 dec,    434 nd
cachet/DQMR/qmr-50/or-50-5-3-UC-30.cnf(134):  0.005s, sat  ,    34 confl,    3779 dec,   3748 nd
    cachet/DQMR/qmr-50/or-50-5-2.cnf(135): expired
cachet/DQMR/qmr-50/or-50-20-9-UC-40.cnf(136):  0.054s, sat  ,    29 confl,   24442 dec,  24373 nd
cachet/DQMR/qmr-50/or-50-10-4-UC-40.cnf(137):  0.624s, sat  ,    28 confl,  542395 dec, 542361 nd
cachet/DQMR/qmr-50/or-50-20-6-UC-40.cnf(138):  0.023s, sat  ,    35 confl,   10905 dec,  10846 nd
    cachet/DQMR/qmr-50/or-50-5-5.cnf(139): expired
cachet/DQMR/qmr-50/or-50-20-1-UC-10.cnf(140): expired
cachet/DQMR/qmr-50/or-50-5-7-UC-40.cnf(141):  0.009s, sat  ,    36 confl,   10596 dec,  10563 nd
cachet/DQMR/qmr-50/or-50-10-7-UC-40.cnf(142):  0.012s, sat  ,    37 confl,   13668 dec,  13635 nd
cachet/DQMR/qmr-50/or-50-10-1-UC-40.cnf(143):  0.001s, sat  ,    35 confl,     223 dec,    172 nd
cachet/DQMR/qmr-50/or-50-10-5-UC-10.cnf(144): expired
cachet/DQMR/qmr-50/or-50-20-3-UC-40.cnf(145):  0.110s, sat  ,    24 confl,   42548 dec,  42475 nd
cachet/DQMR/qmr-50/or-50-10-2-UC-10.cnf(146): 34.651s, sat  ,    19 confl, 22053839 dec, 22053824 nd
cachet/DQMR/qmr-50/or-50-5-5-UC-20.cnf(147): 53.841s, sat  ,    19 confl, 38629853 dec, 38629837 nd
cachet/DQMR/qmr-50/or-50-5-9-UC-10.cnf(148): expired
cachet/DQMR/qmr-50/or-50-20-4-UC-40.cnf(149):  1.910s, sat  ,    25 confl,  917508 dec, 917478 nd
cachet/DQMR/qmr-60/or-60-10-7-UC-30.cnf(150):  0.768s, sat  ,    36 confl,  534643 dec, 534610 nd
cachet/DQMR/qmr-60/or-60-20-9-UC-40.cnf(151):  0.627s, sat  ,    41 confl,  412689 dec, 412583 nd
cachet/DQMR/qmr-60/or-60-10-7-UC-40.cnf(152):  0.015s, sat  ,    43 confl,   11398 dec,  11330 nd
cachet/DQMR/qmr-60/or-60-20-4-UC-40.cnf(153): expired
cachet/DQMR/qmr-60/or-60-10-1-UC-20.cnf(154): expired
cachet/DQMR/qmr-60/or-60-20-3-UC-20.cnf(155): expired
cachet/DQMR/qmr-60/or-60-5-5-UC-30.cnf(156):  0.802s, sat  ,    37 confl,  645828 dec, 645793 nd
cachet/DQMR/qmr-60/or-60-5-6-UC-10.cnf(157): expired
cachet/DQMR/qmr-60/or-60-10-9-UC-40.cnf(158):  0.030s, sat  ,    41 confl,   10678 dec,  10631 nd
cachet/DQMR/qmr-60/or-60-10-6-UC-30.cnf(159):  1.989s, sat  ,    37 confl, 1385264 dec, 1385222 nd
cachet/DQMR/qmr-60/or-60-20-8-UC-40.cnf(160):  0.011s, sat  ,    38 confl,    5105 dec,   5044 nd
cachet/DQMR/qmr-60/or-60-10-10-UC-20.cnf(161): expired
cachet/DQMR/qmr-60/or-60-10-1-UC-30.cnf(162): 14.302s, sat  ,    32 confl, 8008405 dec, 8008360 nd
   cachet/DQMR/qmr-60/or-60-20-8.cnf(163): expired
cachet/DQMR/qmr-60/or-60-5-3-UC-10.cnf(164): expired
    cachet/DQMR/qmr-60/or-60-5-2.cnf(165): expired
   cachet/DQMR/qmr-60/or-60-20-9.cnf(166): expired
   cachet/DQMR/qmr-60/or-60-10-7.cnf(167): expired
cachet/DQMR/qmr-60/or-60-10-3-UC-20.cnf(168): expired
cachet/DQMR/qmr-60/or-60-10-6-UC-20.cnf(169): 12.570s, sat  ,    32 confl, 8077358 dec, 8077302 nd
cachet/DQMR/qmr-60/or-60-5-8-UC-40.cnf(170):  0.002s, sat  ,    46 confl,    1302 dec,   1251 nd
cachet/DQMR/qmr-60/or-60-20-9-UC-20.cnf(171): 22.974s, sat  ,    32 confl, 15896450 dec, 15896378 nd
   cachet/DQMR/qmr-60/or-60-20-2.cnf(172): expired
cachet/DQMR/qmr-60/or-60-10-5-UC-20.cnf(173): 23.680s, sat  ,    30 confl, 12138090 dec, 12138061 nd
cachet/DQMR/qmr-60/or-60-20-1-UC-40.cnf(174):  4.947s, sat  ,    35 confl, 2601847 dec, 2601808 nd
cachet/DQMR/qmr-60/or-60-10-9-UC-20.cnf(175): 13.057s, sat  ,    30 confl, 7803716 dec, 7803682 nd
cachet/DQMR/qmr-60/or-60-20-1-UC-20.cnf(176): expired
cachet/DQMR/qmr-60/or-60-20-9-UC-10.cnf(177): expired
cachet/DQMR/qmr-60/or-60-10-1-UC-10.cnf(178): expired
cachet/DQMR/qmr-60/or-60-10-8-UC-40.cnf(179):  0.002s, sat  ,    43 confl,    1196 dec,   1155 nd
cachet/DQMR/qmr-60/or-60-5-1-UC-10.cnf(180): expired
cachet/DQMR/qmr-60/or-60-10-5-UC-10.cnf(181): expired
cachet/DQMR/qmr-60/or-60-5-2-UC-30.cnf(182): expired
   cachet/DQMR/qmr-60/or-60-10-5.cnf(183): expired
cachet/DQMR/qmr-60/or-60-5-7-UC-30.cnf(184):  1.712s, sat  ,    37 confl, 1311459 dec, 1311403 nd
cachet/DQMR/qmr-60/or-60-5-9-UC-20.cnf(185):  1.598s, sat  ,    34 confl, 1090717 dec, 1090685 nd
cachet/DQMR/qmr-60/or-60-10-3-UC-40.cnf(186):  4.741s, sat  ,    40 confl, 2435067 dec, 2435014 nd
cachet/DQMR/qmr-60/or-60-10-5-UC-40.cnf(187):  0.088s, sat  ,    37 confl,   38338 dec,  38286 nd
cachet/DQMR/qmr-60/or-60-5-9-UC-30.cnf(188):  0.054s, sat  ,    39 confl,   30080 dec,  30034 nd
cachet/DQMR/qmr-60/or-60-10-10-UC-10.cnf(189): expired
cachet/DQMR/qmr-60/or-60-10-2-UC-10.cnf(190): expired
cachet/DQMR/qmr-60/or-60-20-4-UC-20.cnf(191): expired
cachet/DQMR/qmr-60/or-60-20-5-UC-30.cnf(192): expired
cachet/DQMR/qmr-60/or-60-5-8-UC-10.cnf(193): expired
   cachet/DQMR/qmr-60/or-60-20-3.cnf(194): expired
cachet/DQMR/qmr-60/or-60-20-5-UC-10.cnf(195): expired
cachet/DQMR/qmr-60/or-60-10-5-UC-30.cnf(196):  1.561s, sat  ,    36 confl,  881903 dec, 881869 nd
cachet/DQMR/qmr-60/or-60-10-10-UC-40.cnf(197):  0.076s, sat  ,    41 confl,   40079 dec,  39991 nd
cachet/DQMR/qmr-60/or-60-5-9-UC-40.cnf(198):  0.001s, sat  ,    45 confl,     595 dec,    532 nd
cachet/DQMR/qmr-60/or-60-20-2-UC-40.cnf(199):  0.097s, sat  ,    41 confl,   50368 dec,  50306 nd
  cachet/DQMR/qmr-60/or-60-10-10.cnf(200): expired
    cachet/DQMR/qmr-60/or-60-5-9.cnf(201): expired
    cachet/DQMR/qmr-60/or-60-5-3.cnf(202): expired
   cachet/DQMR/qmr-60/or-60-10-6.cnf(203): expired
cachet/DQMR/qmr-60/or-60-20-4-UC-10.cnf(204): expired
cachet/DQMR/qmr-60/or-60-20-6-UC-10.cnf(205): expired
cachet/DQMR/qmr-60/or-60-5-10-UC-10.cnf(206): expired
cachet/DQMR/qmr-60/or-60-20-5-UC-40.cnf(207): expired
cachet/DQMR/qmr-60/or-60-20-10-UC-10.cnf(208): expired
cachet/DQMR/qmr-60/or-60-10-8-UC-30.cnf(209):  2.354s, sat  ,    35 confl, 1420643 dec, 1420612 nd
   cachet/DQMR/qmr-60/or-60-10-1.cnf(210): expired
cachet/DQMR/qmr-60/or-60-20-6-UC-30.cnf(211): expired
cachet/DQMR/qmr-60/or-60-5-10-UC-40.cnf(212):  0.004s, sat  ,    45 confl,    2018 dec,   1929 nd
cachet/DQMR/qmr-60/or-60-20-8-UC-20.cnf(213): 18.595s, sat  ,    31 confl, 11162589 dec, 11162550 nd
cachet/DQMR/qmr-60/or-60-5-4-UC-40.cnf(214):  0.000s, sat  ,    46 confl,     173 dec,    131 nd
cachet/DQMR/qmr-60/or-60-10-9-UC-10.cnf(215): expired
cachet/DQMR/qmr-60/or-60-10-7-UC-20.cnf(216): 22.105s, sat  ,    29 confl, 13806468 dec, 13806441 nd
cachet/DQMR/qmr-60/or-60-10-8-UC-20.cnf(217): expired
   cachet/DQMR/qmr-60/or-60-20-1.cnf(218): expired
cachet/DQMR/qmr-60/or-60-20-7-UC-30.cnf(219): expired
cachet/DQMR/qmr-60/or-60-20-2-UC-20.cnf(220): expired
cachet/DQMR/qmr-60/or-60-5-6-UC-20.cnf(221):  0.697s, sat  ,    38 confl,  593097 dec, 593058 nd
cachet/DQMR/qmr-60/or-60-5-6-UC-40.cnf(222):  0.006s, sat  ,    45 confl,    4846 dec,   4802 nd
```

```
cachet/DQMR/qmr-60/or-60-10-4-UC-30.cnf(223):  0.180s, sat  ,    41 confl,  212787 dec, 212740 nd
cachet/DQMR/qmr-60/or-60-10-2-UC-40.cnf(224):  0.093s, sat  ,    42 confl,   50342 dec,  50266 nd
cachet/DQMR/qmr-60/or-60-5-1-UC-40.cnf(225):  0.015s, sat  ,    47 confl,   14316 dec,  14250 nd
cachet/DQMR/qmr-60/or-60-5-2-UC-40.cnf(226):  0.409s, sat  ,    38 confl,  328363 dec, 328322 nd
   cachet/DQMR/qmr-60/or-60-10-8.cnf(227): expired
    cachet/DQMR/qmr-60/or-60-5-4.cnf(228): expired
    cachet/DQMR/qmr-60/or-60-5-5.cnf(229): expired
cachet/DQMR/qmr-60/or-60-5-10-UC-20.cnf(230): expired
   cachet/DQMR/qmr-60/or-60-5-10.cnf(231): expired
    cachet/DQMR/qmr-60/or-60-5-8.cnf(232): expired
cachet/DQMR/qmr-60/or-60-20-1-UC-10.cnf(233): expired
cachet/DQMR/qmr-60/or-60-5-2-UC-10.cnf(234): expired
cachet/DQMR/qmr-60/or-60-10-2-UC-30.cnf(235):  0.210s, sat  ,    42 confl,  117416 dec, 117330 nd
   cachet/DQMR/qmr-60/or-60-20-7.cnf(236): expired
cachet/DQMR/qmr-60/or-60-10-2-UC-20.cnf(237): expired
cachet/DQMR/qmr-60/or-60-20-2-UC-30.cnf(238):  0.226s, sat  ,    39 confl,  126537 dec, 126500 nd
cachet/DQMR/qmr-60/or-60-10-3-UC-30.cnf(239): expired
cachet/DQMR/qmr-60/or-60-5-4-UC-10.cnf(240): expired
   cachet/DQMR/qmr-60/or-60-10-9.cnf(241): expired
   cachet/DQMR/qmr-60/or-60-10-3.cnf(242): expired
cachet/DQMR/qmr-60/or-60-20-3-UC-30.cnf(243): 23.608s, sat  ,    36 confl, 11946058 dec, 11946006 nd
cachet/DQMR/qmr-60/or-60-10-8-UC-10.cnf(244): expired
cachet/DQMR/qmr-60/or-60-5-7-UC-20.cnf(245): 20.219s, sat  ,    31 confl, 14552294 dec, 14552266 nd
cachet/DQMR/qmr-60/or-60-5-5-UC-40.cnf(246):  0.007s, sat  ,    40 confl,    3466 dec,   3397 nd
cachet/DQMR/qmr-60/or-60-5-6-UC-30.cnf(247):  0.030s, sat  ,    42 confl,   23467 dec,  23426 nd
cachet/DQMR/qmr-60/or-60-10-6-UC-10.cnf(248): expired
cachet/DQMR/qmr-60/or-60-5-3-UC-20.cnf(249): expired
cachet/DQMR/qmr-60/or-60-10-1-UC-40.cnf(250):  1.372s, sat  ,    37 confl,  664873 dec, 664817 nd
cachet/DQMR/qmr-60/or-60-20-3-UC-10.cnf(251): expired
cachet/DQMR/qmr-60/or-60-20-7-UC-40.cnf(252):  0.475s, sat  ,    38 confl,  204192 dec, 204115 nd
cachet/DQMR/qmr-60/or-60-5-5-UC-10.cnf(253): expired
cachet/DQMR/qmr-60/or-60-5-4-UC-20.cnf(254):  0.305s, sat  ,    34 confl,  167929 dec, 167893 nd
cachet/DQMR/qmr-60/or-60-10-7-UC-10.cnf(255): expired
cachet/DQMR/qmr-60/or-60-5-2-UC-20.cnf(256): expired
cachet/DQMR/qmr-60/or-60-5-5-UC-20.cnf(257):  2.772s, sat  ,    35 confl, 2184802 dec, 2184769 nd
cachet/DQMR/qmr-60/or-60-5-4-UC-30.cnf(258):  0.002s, sat  ,    43 confl,     940 dec,    899 nd
cachet/DQMR/qmr-60/or-60-20-10-UC-20.cnf(259): expired
cachet/DQMR/qmr-60/or-60-5-10-UC-30.cnf(260):  0.553s, sat  ,    38 confl,  357846 dec, 357797 nd
    cachet/DQMR/qmr-60/or-60-5-7.cnf(261): expired
cachet/DQMR/qmr-60/or-60-10-10-UC-30.cnf(262):  2.064s, sat  ,    36 confl, 1265115 dec, 1265085 nd
cachet/DQMR/qmr-60/or-60-20-3-UC-40.cnf(263):  1.718s, sat  ,    40 confl,  875704 dec, 875634 nd
   cachet/DQMR/qmr-60/or-60-20-4.cnf(264): expired
cachet/DQMR/qmr-60/or-60-20-1-UC-30.cnf(265): 26.375s, sat  ,    34 confl, 16707813 dec, 16707783 nd
cachet/DQMR/qmr-60/or-60-20-6-UC-20.cnf(266): expired
cachet/DQMR/qmr-60/or-60-5-8-UC-20.cnf(267):  1.376s, sat  ,    32 confl,  729573 dec, 729515 nd
   cachet/DQMR/qmr-60/or-60-20-6.cnf(268): expired
cachet/DQMR/qmr-60/or-60-5-7-UC-10.cnf(269): expired
cachet/DQMR/qmr-60/or-60-10-4-UC-10.cnf(270): expired
cachet/DQMR/qmr-60/or-60-5-7-UC-40.cnf(271):  0.062s, sat  ,    43 confl,   47003 dec,  46956 nd
cachet/DQMR/qmr-60/or-60-5-3-UC-30.cnf(272):  0.038s, sat  ,    41 confl,   26969 dec,  26884 nd
    cachet/DQMR/qmr-60/or-60-5-1.cnf(273): expired
  cachet/DQMR/qmr-60/or-60-20-10.cnf(274): expired
cachet/DQMR/qmr-60/or-60-20-8-UC-10.cnf(275): expired
cachet/DQMR/qmr-60/or-60-20-5-UC-20.cnf(276): expired
cachet/DQMR/qmr-60/or-60-20-8-UC-30.cnf(277):  0.556s, sat  ,    36 confl,  330065 dec, 330001 nd
cachet/DQMR/qmr-60/or-60-5-8-UC-30.cnf(278):  0.012s, sat  ,    40 confl,    6085 dec,   6027 nd
cachet/DQMR/qmr-60/or-60-20-7-UC-20.cnf(279): expired
cachet/DQMR/qmr-60/or-60-10-3-UC-10.cnf(280): expired
cachet/DQMR/qmr-60/or-60-20-6-UC-40.cnf(281): expired
cachet/DQMR/qmr-60/or-60-5-1-UC-30.cnf(282):  8.874s, sat  ,    36 confl, 7618025 dec, 7617974 nd
cachet/DQMR/qmr-60/or-60-10-4-UC-20.cnf(283): expired
cachet/DQMR/qmr-60/or-60-10-4-UC-40.cnf(284):  0.016s, sat  ,    44 confl,   16563 dec,  16515 nd
cachet/DQMR/qmr-60/or-60-20-10-UC-30.cnf(285): expired
   cachet/DQMR/qmr-60/or-60-10-2.cnf(286): expired
cachet/DQMR/qmr-60/or-60-5-9-UC-10.cnf(287): expired
cachet/DQMR/qmr-60/or-60-20-4-UC-30.cnf(288): expired
cachet/DQMR/qmr-60/or-60-20-7-UC-10.cnf(289): expired
    cachet/DQMR/qmr-60/or-60-5-6.cnf(290): expired
cachet/DQMR/qmr-60/or-60-5-3-UC-40.cnf(291):  0.001s, sat  ,    46 confl,     807 dec,    746 nd
cachet/DQMR/qmr-60/or-60-20-2-UC-10.cnf(292): expired
cachet/DQMR/qmr-60/or-60-5-1-UC-20.cnf(293): expired
cachet/DQMR/qmr-60/or-60-10-6-UC-40.cnf(294):  0.118s, sat  ,    40 confl,   80998 dec,  80946 nd
   cachet/DQMR/qmr-60/or-60-10-4.cnf(295): expired
   cachet/DQMR/qmr-60/or-60-20-5.cnf(296): expired
cachet/DQMR/qmr-60/or-60-10-9-UC-30.cnf(297):  0.048s, sat  ,    41 confl,   32160 dec,  32107 nd
cachet/DQMR/qmr-60/or-60-20-10-UC-40.cnf(298):  2.921s, sat  ,    42 confl, 1400140 dec, 1400110 nd
cachet/DQMR/qmr-60/or-60-20-9-UC-30.cnf(299):  1.200s, sat  ,    37 confl,  794463 dec, 794350 nd
cachet/DQMR/qmr-100/or-100-10-2-UC-50.cnf(300): 34.450s, sat  ,    71 confl, 10807446 dec, 10807304 nd
cachet/DQMR/qmr-100/or-100-10-9-UC-50.cnf(301): 27.207s, sat  ,    82 confl, 6710014 dec, 6709831 nd
 cachet/DQMR/qmr-100/or-100-20-4.cnf(302): expired
cachet/DQMR/qmr-100/or-100-5-2-UC-40.cnf(303): expired
cachet/DQMR/qmr-100/or-100-20-1-UC-10.cnf(304): expired
cachet/DQMR/qmr-100/or-100-5-8-UC-60.cnf(305):  0.001s, sat  ,    85 confl,     259 dec,    118 nd
cachet/DQMR/qmr-100/or-100-5-3-UC-20.cnf(306): expired
cachet/DQMR/qmr-100/or-100-20-6-UC-60.cnf(307):  0.058s, sat  ,    65 confl,    7239 dec,   7055 nd
cachet/DQMR/qmr-100/or-100-10-5-UC-20.cnf(308): expired
cachet/DQMR/qmr-100/or-100-20-8-UC-20.cnf(309): expired
cachet/DQMR/qmr-100/or-100-10-3-UC-20.cnf(310): expired
cachet/DQMR/qmr-100/or-100-5-3-UC-30.cnf(311): expired
cachet/DQMR/qmr-100/or-100-5-7-UC-20.cnf(312): expired
cachet/DQMR/qmr-100/or-100-5-4-UC-40.cnf(313): expired
cachet/DQMR/qmr-100/or-100-20-9-UC-30.cnf(314): expired
 cachet/DQMR/qmr-100/or-100-20-5.cnf(315): expired
 cachet/DQMR/qmr-100/or-100-10-6.cnf(316): expired
cachet/DQMR/qmr-100/or-100-20-4-UC-10.cnf(317): expired
cachet/DQMR/qmr-100/or-100-5-5-UC-30.cnf(318): expired
cachet/DQMR/qmr-100/or-100-10-9-UC-60.cnf(319):  2.601s, sat  ,    77 confl,  544211 dec, 544031 nd
cachet/DQMR/qmr-100/or-100-10-7-UC-40.cnf(320): expired
cachet/DQMR/qmr-100/or-100-10-6-UC-20.cnf(321): expired
cachet/DQMR/qmr-100/or-100-10-4-UC-20.cnf(322): expired
```

```
cachet/DQMR/qmr-100/or-100-10-4-UC-50.cnf(323):  0.076s, sat  ,    83 confl,   31939 dec,  31756 nd
cachet/DQMR/qmr-100/or-100-10-2-UC-60.cnf(324):  0.062s, sat  ,    77 confl,   16579 dec,  16398 nd
cachet/DQMR/qmr-100/or-100-5-10-UC-50.cnf(325):  0.028s, sat  ,    79 confl,    9328 dec,   9223 nd
cachet/DQMR/qmr-100/or-100-10-4-UC-40.cnf(326): expired
cachet/DQMR/qmr-100/or-100-10-7-UC-60.cnf(327):  0.060s, sat  ,    75 confl,   13401 dec,  13229 nd
 cachet/DQMR/qmr-100/or-100-10-9.cnf(328): expired
cachet/DQMR/qmr-100/or-100-10-10-UC-30.cnf(329): expired
cachet/DQMR/qmr-100/or-100-20-10-UC-30.cnf(330): expired
 cachet/DQMR/qmr-100/or-100-20-1.cnf(331): expired
cachet/DQMR/qmr-100/or-100-10-1-UC-10.cnf(332): expired
cachet/DQMR/qmr-100/or-100-20-2-UC-10.cnf(333): expired
cachet/DQMR/qmr-100/or-100-20-2-UC-30.cnf(334): expired
cachet/DQMR/qmr-100/or-100-5-8-UC-50.cnf(335):  0.009s, sat  ,    81 confl,    3206 dec,   3078 nd
cachet/DQMR/qmr-100/or-100-20-2-UC-60.cnf(336): expired
cachet/DQMR/qmr-100/or-100-10-9-UC-40.cnf(337): expired
cachet/DQMR/qmr-100/or-100-20-7-UC-20.cnf(338): expired
cachet/DQMR/qmr-100/or-100-20-5-UC-30.cnf(339): expired
cachet/DQMR/qmr-100/or-100-5-4-UC-60.cnf(340):  0.008s, sat  ,    82 confl,    2416 dec,   2283 nd
cachet/DQMR/qmr-100/or-100-10-6-UC-60.cnf(341):  0.020s, sat  ,    76 confl,    3829 dec,   3645 nd
cachet/DQMR/qmr-100/or-100-20-9-UC-40.cnf(342): expired
cachet/DQMR/qmr-100/or-100-5-6-UC-30.cnf(343): expired
cachet/DQMR/qmr-100/or-100-5-9-UC-30.cnf(344): expired
cachet/DQMR/qmr-100/or-100-10-5-UC-50.cnf(345): expired
cachet/DQMR/qmr-100/or-100-10-8-UC-20.cnf(346): expired
cachet/DQMR/qmr-100/or-100-5-5-UC-40.cnf(347): expired
 cachet/DQMR/qmr-100/or-100-10-1.cnf(348): expired
cachet/DQMR/qmr-100/or-100-10-7-UC-10.cnf(349): expired
 cachet/DQMR/qmr-100/or-100-10-7.cnf(350): expired
cachet/DQMR/qmr-100/or-100-20-7-UC-60.cnf(351):  6.305s, sat  ,    74 confl, 1248177 dec, 1247970 nd
 cachet/DQMR/qmr-100/or-100-20-9.cnf(352): expired
 cachet/DQMR/qmr-100/or-100-20-6.cnf(353): expired
cachet/DQMR/qmr-100/or-100-20-8-UC-60.cnf(354): 36.624s, sat  ,    79 confl, 6739660 dec, 6739472 nd
cachet/DQMR/qmr-100/or-100-20-4-UC-50.cnf(355): expired
cachet/DQMR/qmr-100/or-100-5-9-UC-10.cnf(356): expired
cachet/DQMR/qmr-100/or-100-10-7-UC-30.cnf(357): expired
  cachet/DQMR/qmr-100/or-100-5-1.cnf(358): expired
cachet/DQMR/qmr-100/or-100-20-8-UC-50.cnf(359): expired
cachet/DQMR/qmr-100/or-100-5-2-UC-60.cnf(360):  0.071s, sat  ,    72 confl,   13478 dec,  13259 nd
cachet/DQMR/qmr-100/or-100-20-6-UC-30.cnf(361): expired
cachet/DQMR/qmr-100/or-100-5-9-UC-50.cnf(362):  0.049s, sat  ,    79 confl,   14866 dec,  14726 nd
cachet/DQMR/qmr-100/or-100-20-1-UC-30.cnf(363): expired
cachet/DQMR/qmr-100/or-100-20-3-UC-10.cnf(364): expired
cachet/DQMR/qmr-100/or-100-5-10-UC-40.cnf(365):  0.430s, sat  ,    75 confl,  147580 dec, 147463 nd
cachet/DQMR/qmr-100/or-100-10-2-UC-10.cnf(366): expired
cachet/DQMR/qmr-100/or-100-20-9-UC-60.cnf(367):  0.184s, sat  ,    78 confl,   39542 dec,  39317 nd
cachet/DQMR/qmr-100/or-100-20-4-UC-40.cnf(368): expired
cachet/DQMR/qmr-100/or-100-10-2-UC-40.cnf(369): expired
cachet/DQMR/qmr-100/or-100-5-5-UC-20.cnf(370): expired
cachet/DQMR/qmr-100/or-100-20-2-UC-50.cnf(371): expired
 cachet/DQMR/qmr-100/or-100-20-7.cnf(372): expired
cachet/DQMR/qmr-100/or-100-20-4-UC-60.cnf(373): expired
cachet/DQMR/qmr-100/or-100-5-1-UC-20.cnf(374): expired
cachet/DQMR/qmr-100/or-100-10-8-UC-40.cnf(375): expired
cachet/DQMR/qmr-100/or-100-5-1-UC-10.cnf(376): expired
cachet/DQMR/qmr-100/or-100-20-10-UC-40.cnf(377): expired
cachet/DQMR/qmr-100/or-100-10-1-UC-30.cnf(378): expired
cachet/DQMR/qmr-100/or-100-20-5-UC-60.cnf(379):  0.426s, sat  ,    71 confl,   71627 dec,  71409 nd
cachet/DQMR/qmr-100/or-100-10-10-UC-50.cnf(380):  2.034s, sat  ,    77 confl,  522758 dec, 522543 nd
cachet/DQMR/qmr-100/or-100-10-5-UC-30.cnf(381): expired
cachet/DQMR/qmr-100/or-100-10-9-UC-30.cnf(382): expired
cachet/DQMR/qmr-100/or-100-5-1-UC-60.cnf(383):  0.001s, sat  ,    84 confl,     147 dec,     38 nd
cachet/DQMR/qmr-100/or-100-20-6-UC-40.cnf(384): expired
cachet/DQMR/qmr-100/or-100-5-7-UC-50.cnf(385):  2.391s, sat  ,    76 confl,  663689 dec, 663563 nd
cachet/DQMR/qmr-100/or-100-5-10-UC-20.cnf(386): expired
cachet/DQMR/qmr-100/or-100-20-5-UC-50.cnf(387): expired
cachet/DQMR/qmr-100/or-100-10-5-UC-40.cnf(388): expired
cachet/DQMR/qmr-100/or-100-5-5-UC-10.cnf(389): expired
cachet/DQMR/qmr-100/or-100-10-10-UC-40.cnf(390): 45.700s, sat  ,    65 confl, 12514960 dec, 12514762 nd
cachet/DQMR/qmr-100/or-100-20-6-UC-10.cnf(391): expired
cachet/DQMR/qmr-100/or-100-10-6-UC-40.cnf(392): expired
cachet/DQMR/qmr-100/or-100-10-9-UC-10.cnf(393): expired
cachet/DQMR/qmr-100/or-100-5-10-UC-60.cnf(394):  0.001s, sat  ,    83 confl,     198 dec,    102 nd
cachet/DQMR/qmr-100/or-100-20-5-UC-10.cnf(395): expired
cachet/DQMR/qmr-100/or-100-5-8-UC-30.cnf(396): expired
cachet/DQMR/qmr-100/or-100-5-3-UC-60.cnf(397):  0.070s, sat  ,    76 confl,   23120 dec,  23042 nd
cachet/DQMR/qmr-100/or-100-10-8-UC-30.cnf(398): expired
cachet/DQMR/qmr-100/or-100-20-1-UC-20.cnf(399): expired
 cachet/DQMR/qmr-100/or-100-5-10.cnf(400): expired
cachet/DQMR/qmr-100/or-100-5-5-UC-60.cnf(401):  3.006s, sat  ,    75 confl,  861949 dec, 861741 nd
cachet/DQMR/qmr-100/or-100-10-3-UC-40.cnf(402): expired
cachet/DQMR/qmr-100/or-100-5-8-UC-20.cnf(403): expired
cachet/DQMR/qmr-100/or-100-20-6-UC-50.cnf(404): expired
cachet/DQMR/qmr-100/or-100-5-2-UC-20.cnf(405): expired
 cachet/DQMR/qmr-100/or-100-10-5.cnf(406): expired
cachet/DQMR/qmr-100/or-100-5-6-UC-50.cnf(407):  0.148s, sat  ,    76 confl,   43984 dec,  43787 nd
cachet/DQMR/qmr-100/or-100-5-4-UC-20.cnf(408): expired
cachet/DQMR/qmr-100/or-100-5-9-UC-20.cnf(409): expired
 cachet/DQMR/qmr-100/or-100-10-8.cnf(410): expired
cachet/DQMR/qmr-100/or-100-5-7-UC-10.cnf(411): expired
cachet/DQMR/qmr-100/or-100-5-8-UC-10.cnf(412): expired
cachet/DQMR/qmr-100/or-100-20-1-UC-50.cnf(413): 18.190s, sat  ,    70 confl, 3178559 dec, 3178381 nd
cachet/DQMR/qmr-100/or-100-5-8-UC-40.cnf(414): 10.222s, sat  ,    71 confl, 3939405 dec, 3939269 nd
cachet/DQMR/qmr-100/or-100-10-4-UC-60.cnf(415):  0.001s, sat  ,    88 confl,     393 dec,    206 nd
  cachet/DQMR/qmr-100/or-100-5-6.cnf(416): expired
 cachet/DQMR/qmr-100/or-100-20-3.cnf(417): expired
cachet/DQMR/qmr-100/or-100-5-4-UC-10.cnf(418): expired
cachet/DQMR/qmr-100/or-100-10-6-UC-50.cnf(419):  1.099s, sat  ,    73 confl,  236689 dec, 236525 nd
cachet/DQMR/qmr-100/or-100-10-8-UC-10.cnf(420): expired
  cachet/DQMR/qmr-100/or-100-5-5.cnf(421): expired
cachet/DQMR/qmr-100/or-100-10-3-UC-30.cnf(422): expired
```

```
cachet/DQMR/qmr-100/or-100-20-10-UC-60.cnf(423): 14.147s, sat  ,    78 confl, 2493315 dec, 2493132 nd
cachet/DQMR/qmr-100/or-100-10-4-UC-10.cnf(424): expired
cachet/DQMR/qmr-100/or-100-5-3-UC-50.cnf(425): 16.790s, sat  ,    66 confl, 4835013 dec, 4834947 nd
cachet/DQMR/qmr-100/or-100-5-10-UC-10.cnf(426): expired
cachet/DQMR/qmr-100/or-100-5-7-UC-60.cnf(427):  0.006s, sat  ,    77 confl,    1496 dec,   1367 nd
cachet/DQMR/qmr-100/or-100-20-3-UC-50.cnf(428):  3.426s, sat  ,    75 confl,  827894 dec, 827733 nd
cachet/DQMR/qmr-100/or-100-20-2-UC-40.cnf(429): expired
cachet/DQMR/qmr-100/or-100-5-5-UC-50.cnf(430):  8.636s, sat  ,    75 confl, 2645314 dec, 2645071 nd
cachet/DQMR/qmr-100/or-100-20-5-UC-20.cnf(431): expired
cachet/DQMR/qmr-100/or-100-5-6-UC-20.cnf(432): expired
cachet/DQMR/qmr-100/or-100-10-2-UC-30.cnf(433): expired
cachet/DQMR/qmr-100/or-100-20-10-UC-50.cnf(434): expired
cachet/DQMR/qmr-100/or-100-5-3-UC-10.cnf(435): expired
cachet/DQMR/qmr-100/or-100-5-7-UC-30.cnf(436): expired
 cachet/DQMR/qmr-100/or-100-20-8.cnf(437): expired
 cachet/DQMR/qmr-100/or-100-20-2.cnf(438): expired
cachet/DQMR/qmr-100/or-100-20-8-UC-30.cnf(439): expired
cachet/DQMR/qmr-100/or-100-5-1-UC-40.cnf(440):  0.537s, sat  ,    74 confl,  170361 dec, 170246 nd
cachet/DQMR/qmr-100/or-100-10-5-UC-60.cnf(441):  0.010s, sat  ,    78 confl,    2758 dec,   2601 nd
cachet/DQMR/qmr-100/or-100-20-1-UC-60.cnf(442):  0.854s, sat  ,    75 confl,  136436 dec, 136277 nd
  cachet/DQMR/qmr-100/or-100-5-4.cnf(443): expired
cachet/DQMR/qmr-100/or-100-10-1-UC-40.cnf(444):  0.442s, sat  ,    77 confl,  184481 dec, 184325 nd
cachet/DQMR/qmr-100/or-100-5-4-UC-50.cnf(445):  0.100s, sat  ,    82 confl,   37141 dec,  37004 nd
cachet/DQMR/qmr-100/or-100-10-1-UC-60.cnf(446):  0.003s, sat  ,    83 confl,    1120 dec,    964 nd
cachet/DQMR/qmr-100/or-100-20-4-UC-30.cnf(447): expired
cachet/DQMR/qmr-100/or-100-20-1-UC-40.cnf(448): expired
cachet/DQMR/qmr-100/or-100-10-1-UC-50.cnf(449):  0.005s, sat  ,    83 confl,    1880 dec,   1732 nd
cachet/DQMR/qmr-100/or-100-10-6-UC-30.cnf(450): expired
cachet/DQMR/qmr-100/or-100-5-4-UC-30.cnf(451): expired
cachet/DQMR/qmr-100/or-100-10-9-UC-20.cnf(452): expired
 cachet/DQMR/qmr-100/or-100-10-3.cnf(453): expired
cachet/DQMR/qmr-100/or-100-5-9-UC-60.cnf(454):  0.006s, sat  ,    82 confl,    2281 dec,   2150 nd
cachet/DQMR/qmr-100/or-100-10-2-UC-20.cnf(455): expired
  cachet/DQMR/qmr-100/or-100-5-8.cnf(456): expired
cachet/DQMR/qmr-100/or-100-10-8-UC-60.cnf(457):  4.215s, sat  ,    82 confl, 1805148 dec, 1804952 nd
cachet/DQMR/qmr-100/or-100-20-5-UC-40.cnf(458): expired
cachet/DQMR/qmr-100/or-100-5-2-UC-10.cnf(459): expired
  cachet/DQMR/qmr-100/or-100-5-3.cnf(460): expired
cachet/DQMR/qmr-100/or-100-20-10-UC-20.cnf(461): expired
cachet/DQMR/qmr-100/or-100-20-8-UC-40.cnf(462): expired
cachet/DQMR/qmr-100/or-100-10-3-UC-60.cnf(463):  0.041s, sat  ,    77 confl,   12136 dec,  12039 nd
cachet/DQMR/qmr-100/or-100-10-5-UC-10.cnf(464): expired
cachet/DQMR/qmr-100/or-100-20-3-UC-30.cnf(465): expired
cachet/DQMR/qmr-100/or-100-5-1-UC-50.cnf(466):  0.004s, sat  ,    81 confl,    1026 dec,    903 nd
cachet/DQMR/qmr-100/or-100-20-6-UC-20.cnf(467): expired
cachet/DQMR/qmr-100/or-100-10-3-UC-10.cnf(468): expired
cachet/DQMR/qmr-100/or-100-10-10.cnf(469): expired
cachet/DQMR/qmr-100/or-100-20-7-UC-50.cnf(470): expired
cachet/DQMR/qmr-100/or-100-20-10.cnf(471): expired
cachet/DQMR/qmr-100/or-100-20-3-UC-20.cnf(472): expired
cachet/DQMR/qmr-100/or-100-5-6-UC-60.cnf(473):  0.006s, sat  ,    80 confl,    1689 dec,   1551 nd
cachet/DQMR/qmr-100/or-100-10-6-UC-10.cnf(474): expired
cachet/DQMR/qmr-100/or-100-5-3-UC-40.cnf(475): expired
cachet/DQMR/qmr-100/or-100-20-7-UC-10.cnf(476): expired
cachet/DQMR/qmr-100/or-100-10-10-UC-20.cnf(477): expired
  cachet/DQMR/qmr-100/or-100-5-2.cnf(478): expired
cachet/DQMR/qmr-100/or-100-20-9-UC-20.cnf(479): expired
cachet/DQMR/qmr-100/or-100-20-9-UC-50.cnf(480):  0.451s, sat  ,    76 confl,   92390 dec,  92145 nd
cachet/DQMR/qmr-100/or-100-20-7-UC-30.cnf(481): expired
cachet/DQMR/qmr-100/or-100-5-1-UC-30.cnf(482):  5.380s, sat  ,    73 confl, 1909973 dec, 1909861 nd
  cachet/DQMR/qmr-100/or-100-5-7.cnf(483): expired
cachet/DQMR/qmr-100/or-100-5-10-UC-30.cnf(484): expired
cachet/DQMR/qmr-100/or-100-10-10-UC-10.cnf(485): expired
cachet/DQMR/qmr-100/or-100-5-6-UC-40.cnf(486):  8.686s, sat  ,    69 confl, 2666907 dec, 2666698 nd
cachet/DQMR/qmr-100/or-100-10-7-UC-20.cnf(487): expired
  cachet/DQMR/qmr-100/or-100-5-9.cnf(488): expired
cachet/DQMR/qmr-100/or-100-5-7-UC-40.cnf(489): 52.687s, sat  ,    74 confl, 17670548 dec, 17670399 nd
cachet/DQMR/qmr-100/or-100-20-4-UC-20.cnf(490): expired
cachet/DQMR/qmr-100/or-100-20-3-UC-60.cnf(491):  0.016s, sat  ,    79 confl,    3236 dec,   2999 nd
cachet/DQMR/qmr-100/or-100-10-1-UC-20.cnf(492): expired
 cachet/DQMR/qmr-100/or-100-10-2.cnf(493): expired
cachet/DQMR/qmr-100/or-100-20-10-UC-10.cnf(494): expired
cachet/DQMR/qmr-100/or-100-20-8-UC-10.cnf(495): expired
cachet/DQMR/qmr-100/or-100-20-3-UC-40.cnf(496): expired
cachet/DQMR/qmr-100/or-100-5-2-UC-30.cnf(497): expired
cachet/DQMR/qmr-100/or-100-20-7-UC-40.cnf(498): expired
cachet/DQMR/qmr-100/or-100-10-7-UC-50.cnf(499):  5.399s, sat  ,    68 confl, 1165089 dec, 1164936 nd
cachet/DQMR/qmr-100/or-100-5-9-UC-40.cnf(500):  0.162s, sat  ,    82 confl,   76890 dec,  76807 nd
cachet/DQMR/qmr-100/or-100-10-3-UC-50.cnf(501):  2.630s, sat  ,    74 confl,  900741 dec, 900616 nd
cachet/DQMR/qmr-100/or-100-20-2-UC-20.cnf(502): expired
cachet/DQMR/qmr-100/or-100-10-8-UC-50.cnf(503): expired
 cachet/DQMR/qmr-100/or-100-10-4.cnf(504): expired
cachet/DQMR/qmr-100/or-100-5-2-UC-50.cnf(505):  6.653s, sat  ,    70 confl, 1553944 dec, 1553761 nd
cachet/DQMR/qmr-100/or-100-10-4-UC-30.cnf(506): expired
cachet/DQMR/qmr-100/or-100-5-6-UC-10.cnf(507): expired
cachet/DQMR/qmr-100/or-100-10-10-UC-60.cnf(508):  0.020s, sat  ,    75 confl,    4874 dec,   4619 nd
cachet/DQMR/qmr-100/or-100-20-9-UC-10.cnf(509): expired
cachet/DQMR/qmr-70/or-70-5-8-UC-40.cnf(510):  0.026s, sat  ,    55 confl,   16199 dec,  16134 nd
cachet/DQMR/qmr-70/or-70-20-8-UC-40.cnf(511): expired
cachet/DQMR/qmr-70/or-70-10-2-UC-30.cnf(512): expired
cachet/DQMR/qmr-70/or-70-20-7-UC-10.cnf(513): expired
cachet/DQMR/qmr-70/or-70-5-3-UC-30.cnf(514):  2.576s, sat  ,    47 confl, 2299472 dec, 2299426 nd
cachet/DQMR/qmr-70/or-70-10-6-UC-10.cnf(515): expired
cachet/DQMR/qmr-70/or-70-20-3-UC-10.cnf(516): expired
cachet/DQMR/qmr-70/or-70-5-2-UC-20.cnf(517): expired
cachet/DQMR/qmr-70/or-70-10-1-UC-30.cnf(518): expired
    cachet/DQMR/qmr-70/or-70-5-8.cnf(519): expired
   cachet/DQMR/qmr-70/or-70-10-9.cnf(520): expired
cachet/DQMR/qmr-70/or-70-5-10-UC-30.cnf(521): expired
cachet/DQMR/qmr-70/or-70-10-8-UC-20.cnf(522):  5.064s, sat  ,    41 confl, 2359336 dec, 2359298 nd
```

```
cachet/DQMR/qmr-70/or-70-10-10-UC-20.cnf(523): expired
cachet/DQMR/qmr-70/or-70-5-8-UC-20.cnf(524): expired
cachet/DQMR/qmr-70/or-70-10-5-UC-40.cnf(525):  0.092s, sat  ,    49 confl,   39404 dec,  39265 nd
cachet/DQMR/qmr-70/or-70-5-5-UC-40.cnf(526):  0.009s, sat  ,    54 confl,    6837 dec,   6787 nd
    cachet/DQMR/qmr-70/or-70-5-3.cnf(527): expired
   cachet/DQMR/qmr-70/or-70-20-9.cnf(528): expired
   cachet/DQMR/qmr-70/or-70-20-1.cnf(529): expired
   cachet/DQMR/qmr-70/or-70-10-3.cnf(530): expired
cachet/DQMR/qmr-70/or-70-5-3-UC-20.cnf(531):  0.636s, sat  ,    50 confl,  550532 dec, 550457 nd
cachet/DQMR/qmr-70/or-70-20-10-UC-30.cnf(532): expired
cachet/DQMR/qmr-70/or-70-5-2-UC-30.cnf(533):  0.782s, sat  ,    41 confl,  325462 dec, 325396 nd
cachet/DQMR/qmr-70/or-70-5-5-UC-30.cnf(534):  0.133s, sat  ,    50 confl,   96754 dec,  96706 nd
    cachet/DQMR/qmr-70/or-70-5-9.cnf(535): expired
cachet/DQMR/qmr-70/or-70-10-9-UC-10.cnf(536): expired
cachet/DQMR/qmr-70/or-70-5-1-UC-30.cnf(537):  0.004s, sat  ,    52 confl,    2367 dec,   2309 nd
cachet/DQMR/qmr-70/or-70-10-2-UC-10.cnf(538): expired
    cachet/DQMR/qmr-70/or-70-5-2.cnf(539): expired
cachet/DQMR/qmr-70/or-70-20-3-UC-20.cnf(540): expired
cachet/DQMR/qmr-70/or-70-5-6-UC-40.cnf(541):  0.001s, sat  ,    59 confl,     604 dec,    514 nd
cachet/DQMR/qmr-70/or-70-10-4-UC-30.cnf(542): expired
cachet/DQMR/qmr-70/or-70-10-1-UC-10.cnf(543): expired
cachet/DQMR/qmr-70/or-70-5-1-UC-20.cnf(544): 32.001s, sat  ,    39 confl, 17586751 dec, 17586698 nd
cachet/DQMR/qmr-70/or-70-5-6-UC-20.cnf(545): expired
cachet/DQMR/qmr-70/or-70-20-2-UC-10.cnf(546): expired
cachet/DQMR/qmr-70/or-70-10-4-UC-40.cnf(547): expired
cachet/DQMR/qmr-70/or-70-10-9-UC-40.cnf(548):  0.286s, sat  ,    48 confl,  139483 dec, 139392 nd
cachet/DQMR/qmr-70/or-70-5-10-UC-40.cnf(549):  7.912s, sat  ,    40 confl, 3927813 dec, 3927699 nd
cachet/DQMR/qmr-70/or-70-5-9-UC-10.cnf(550): expired
cachet/DQMR/qmr-70/or-70-5-9-UC-30.cnf(551):  0.094s, sat  ,    50 confl,   55137 dec,  55093 nd
   cachet/DQMR/qmr-70/or-70-20-6.cnf(552): expired
cachet/DQMR/qmr-70/or-70-20-4-UC-40.cnf(553):  0.876s, sat  ,    47 confl,  345155 dec, 345025 nd
cachet/DQMR/qmr-70/or-70-10-3-UC-40.cnf(554):  0.006s, sat  ,    53 confl,    3457 dec,   3366 nd
cachet/DQMR/qmr-70/or-70-20-10-UC-40.cnf(555): 19.100s, sat  ,    52 confl, 7904229 dec, 7904169 nd
    cachet/DQMR/qmr-70/or-70-5-6.cnf(556): expired
```

Epilogue & Erratum

1.) chapter 2.6.2, 3rd paragraph, 2nd sentence
"The variable values of variables in unsatisfied clauses can be deduced because they need to be false there"
The sentence is wrong because it should be about not-yet-satisfied clauses rather than unsatisfied clauses. A singleton false clause makes the whole CNF false so that no more components can be cached. In order to detect contingency components it is sufficient to look at the set of assigned variables and the set of not-yet-satisfied higher order clauses. If you remove nodes corresponding to assigned variables from the graph it may already fall apart into contingency components. If you do furthermore remove edges for satisfied clauses then you have the full picture. I have supposed that SharpSAT may count wrong because of the premature caching of components in false CNFs. However if you look at not-yet-satisfied clauses to yield contingency components then you need to be able to detect without doubt whether such a clause is really satisfied. I wanna argue that this is not possible with a state-of-the-art watched literal solver because such solvers may or may not meet a literal that has become true newly. If a true literal is not found a larger contingency component with more clauses and thus fewer solutions will be assumed. This does exactly correspond to the finding that SharpSAT returns too little solutions in some cases. Consequently we argue that a solver that does component caching in the way SharpSAT does, will require dual data structures and literal counting for core clauses. The only solver we know that does this at the present time is DualSAT. Concerning dual clauses such a clause can either be true or not-yet-satisfied. If there is an unsatisfied clause a conflict is detected and no components must be cached. Consequently, if there is only one assigned variable in a dual clause then the whole clause must be true. If both variables are assigned one of them also needs to be

true as the whole clause.
typing error: It should be 17288325 solutions for SharpSAT
on page 32 instead of 1728832.

2.) chapter 2.5.3, page 29, 3rd paragraph
"compensate_literal_movement_of_false"
The watched literals can be
- unassigned literals
- true literals
- one of them: a false literal of maximum level
The benefit of compensate_literal_movement_of_false can be explained because it may find a true literal instead of a false literal of maximum level. Such a literal is worth more because it does not need to change for a longer time especially when it is deep in the stack. It pays off to look at variables which have changed level during stack redo because such variables may have changed value and as corresponding clauses are still in the cache.

3.) chapter 2.2.4, 3rd paragraph
instead of "backtrack level for backjumping" say "current backtrack level ..." as the final backtrack level always equals the assertion level. It would be most accurate to say "reduced conflict level after initial chronological backtracking".

4.) chapter 2.2.4.1, 2nd paragraph after figure 7
"non(y)" instead of y: It would discover that the fact non(y) is atomically true.

5.) spelling mistakes
chapter 2.5.2, page 27, last paragraph:
"rationale" instead of "rational"
chapter 2.6.1, page 31, 3rd paragraph:
"one" instead of "on", two times

chapter 2.6.2, page 32, 4th paragraph:
"easy to compute" instead of "easy to computer"
chapter 2.2.4, figure 5:
"set all redone variables" instead of "... variable"

6.) I could see the following error in the final version of my thesis as I had just submitted it:
chapter 1.1, page 7, 1st paragraph:
"f.i. execute a non-linear optimization problem"
instead of "f.i. execute a optimization problem"
The word non-linear is important here as there are integrated solutions for linear optimization problems being attached to a CNF. I could verify that in previous printouts of my thesis the word non-linear was not missing. As I have not deleted that word myself, this points to an external manipulation by some third party. Fortunately the word non-linear was not missing before "optimization problem" in chapter 3. When I finally received the submitted version back from my professor in order to write this erratum the error was no more there. Either the error had been inserted directly after submission or the submitted version has been changed by Western Secret Services a few month after submission when I was emailing with my professor about my intent to uncover this manipulation. It needs to be said that I have written DualSAT and this thesis on a computer that has been kept offline since the very installation of the Linux operating system. No data was exchanged with other computers, neither by USB-stick nor at that time by burning CDs. I had taken such extreme security measures since I knew Western Secret Services were not only spying at me since 2008 but also actively terrorizing me. You can read more about it at https://www.elstel.org/aktionen/blocked-on-debian-security.html. I did even carry the notebook with me, every time I was going for a walk in the Europe park near my university. That way it should be impossible to infect my machine with a Stuxnet-like worm. However all of that did not help me.

I was terrorized while writing my thesis. They temporarily exchanged the dot character on my keyboard with a colon so that I had to copy and paste the dot from a previous sentence. Unfortunately this was not the only act of terror against me and my work. It was in late summer when I chose to continue the work of DualSAT by replacing the literal counting for false literals by watched literals. Counting only true literals would have given a considerable speedup. When I presented my professor the first benchmark results of my solver, he was very excited and all solution counts did match those for SharpSAT except the two problems where SharpSAT was known to count wrong. DualSAT and Clasp calculated the correct results. However when I tried to implement hybrid and dual data structures for DualSAT it suddenly gave me totally wrong solution counts. Shocked by this result, I was searching for the last known good version. I remembered to have run the tests every now and then on from version 0.3.4. Unfortunately I did not correct the should-be solution counts for the two problems where SharpSAT fails, so that these problems had gone unchecked since. It showed all versions back to but excluding 0.3.4 to be faulty in one problem where SharpSAT fails and in one another problem. It was the version where I had manually checked the two results with my professor. I can well remember having run the tests on many intermediate versions and I would have seen that immediately if another problem would have failed. I was a bit in wonder but I started to search for an error. I finally believed in a memory access error because DualSAT was yielding different results if I exchanged two code blocks working on totally different and separate data structures. I fired up asan and valgrind, two tools especially designed to find memory access errors but could not find any error. The tools reported that everything was correct. I continued my search manually and saw that inserting a byte in front of

a bit field did change the results. Strangely inserting more than one byte did no more change the results, something that should be more or less impossible to observe with an error like this, since the variable bitfield contains many true and false bits in an arbitrarily distributed fashion. Moreover all bits tend to be set till the end so that an overwritten value that is not FF should mind. An alteration in this bitfield should normally trigger an assertion error. I had been searching for an error for a whole month, but I did not suspect anything yet. The time it made click and the time I suddenly knew that my machine was bugged, was when I could prove from the variable setting in gdb, the GNU debugger, that the statements directly before the breakpoint could not have been executed. Consequently I tried to install my system on another computer and on another. None did seem to give me any results at all or at least better results till I tested on a Core 2 machine from 2008 which did not have an Intel ME. On this machine I started again to undo every changed line of code from 0.3.4 towards the next version. The only thing that was still different was clause deletion. I commented the only different code block out by writing || true to skip the else. The loop did output two values on each iteration: the processed learned clauses and the kept learned clauses. As the code block to delete clauses was commented out, it must have output two times the same value for all learned clauses. In a fact it did not do so from a certain point in the middle on. I have made screenshots and double-checked that I had photographed the version with || true instead of the control version with && true. I later showed the photographs to my professor and I can remember having explained him on what happened when I replaced the || true by && true. Unfortunately the photo I still have was apparently manipulated to show && instead of || which makes the photos worthless unless you explain the whole story.

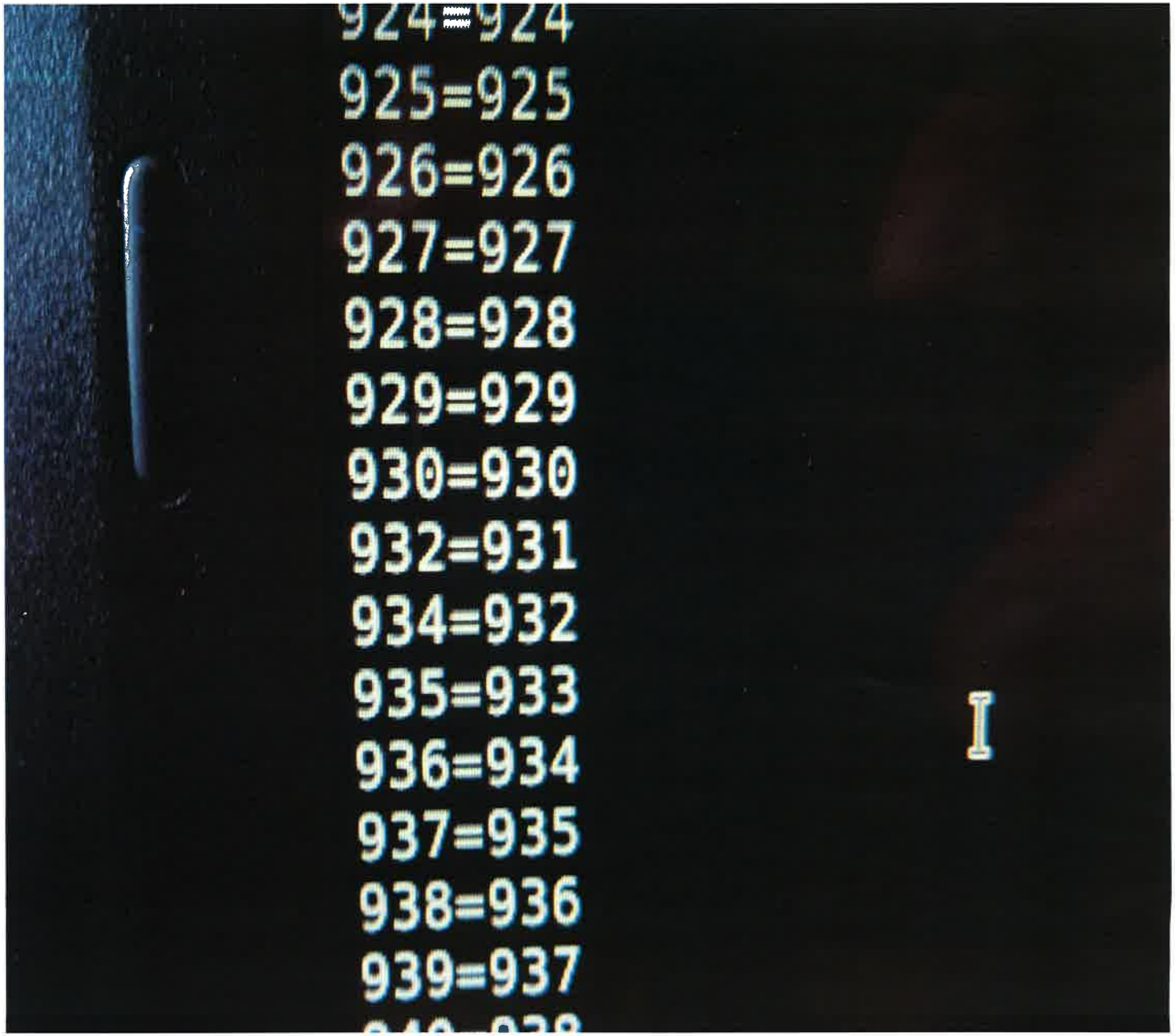
Directly after I took the photos the same version, the same compilation of DualSAT applied on the same problem did suddenly crash with a memory access error. The program is not multithreaded. My compact camera seems also to be bugged since it sometimes hangs and crashes. Something I would not expect from a device with original firmware. The camera did not crash or hang in the first years I was using it. I did never flash the firmware nor did I connect the camera to some network. Finally I had a proof that someone did manipulate the results on my computer and I showed the yet unaltered photos to my professor. All the benchmark results I have printed in my thesis are from the very first benchmark I have sent to my professor. As far as I know this one had not been manipulated. If it would have been manipulated then to my detriment. They have likely bugged my results because they were better than they wanted to allow me. After I had finished my thesis they did also evoke errors when I programmed xchroot for my homepage offline. This is a totally different programme the secret services should not be interested in. I assigned a variable in one line and output it in the next line but the variable did not have the value assigned to it just before. Likely they wanted to stop the programme from becoming too successful. I can also report about a third programme that I had programmed on my 80486 to spot the ROM BIOS region in memory. On the 80486 the programme did suddenly stop to work just before it gets to the ROM BIOS region in memory at 0xF0000. I have later on run the same executable on a Core 2 system and it did not expose this error any more. Looking at the source code such an error is rather improbable. To complete my examination of bugged machines I also bought a new computer at Media Market, removed the Wifi module as on my offline machine, installed from scratch and it did expose the same errors. This was a test with an AMD machine. It means that all PCs you can buy are bugged right on from their product-

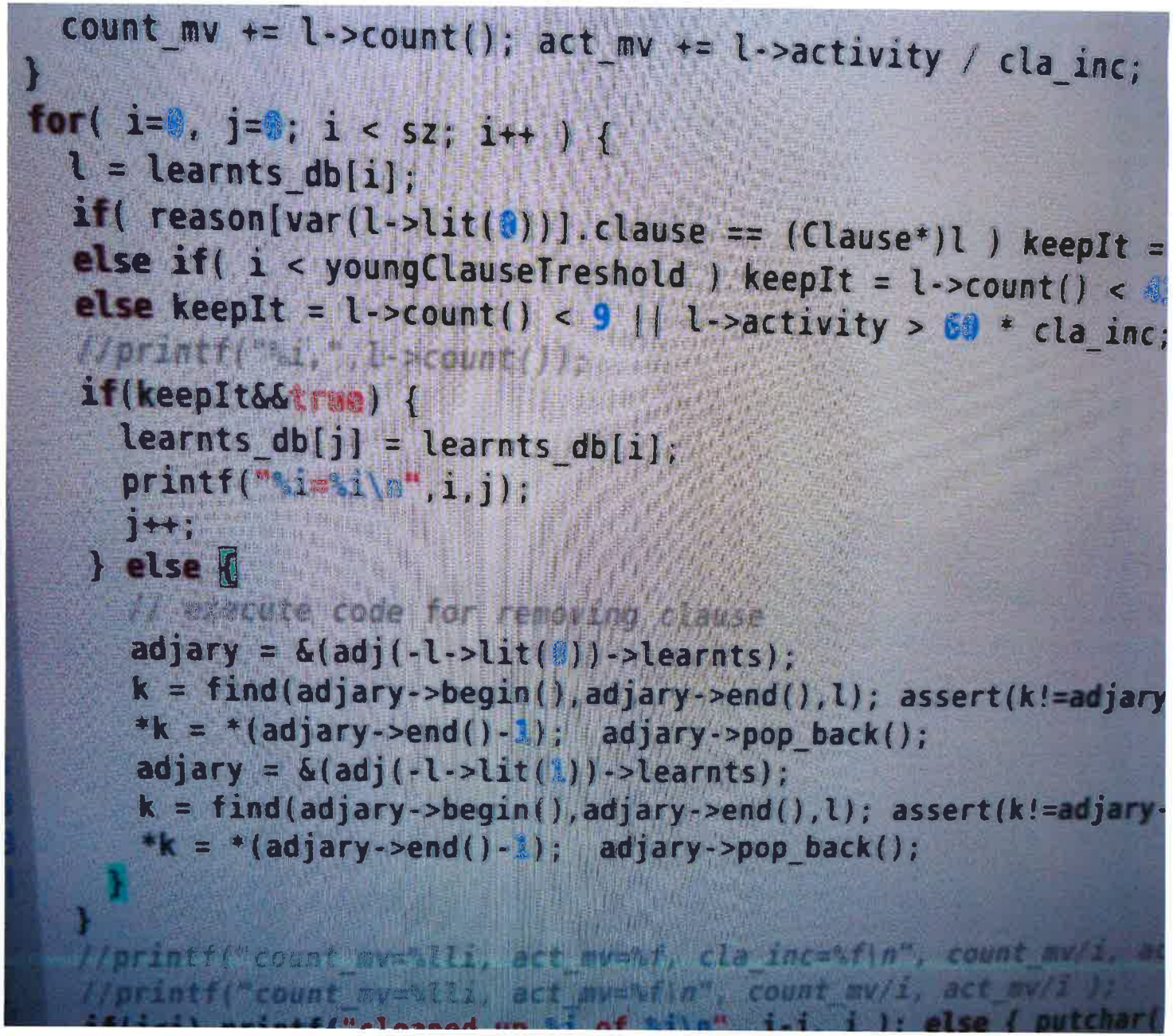
ion. The 80486 machine has for sure initially not been bugged since it is too old for this. However I expect the ROM BIOS module to have been exchanged by Western Secret Services.

I am writing here because I do very much believe that my findings are far reaching. If all computers developed by countries of the 5 eyes including the British Raspberry Pi are bugged on from their very production, this is not only a severe security risk to me as a person but to all Europeans and the whole rest of this world. The film "Zero Days" speaks of the information weaupon to be able to be more devastating than a nuclear weaupon but it does not speak out about why it exists and who possesses this weaupon.

I hope that uncovering these issues does not raise doubt about the correctness of my scientific results. In deed I could not test the purely chronological-equivalent version of the stack redo with an untampered machine. This is the version which can do without nogoods. With nogoods DualSAT used to return the two manipulated counters. When I switched the nogoods off the two previously manipulated counters started to return the correct result and two other errors at two other problems did suddenly appear. These counters returned fewer solutions than necessary. However this is impossible because removing a constraint can only lead to more results, not less. We inferenced that this version of stack redo should consequently work correctly once it is executed on an unbugged machine. We additionally put considerable effort into making the algorithm work on paper which can be seen in simplified form from chapter 2.2.4.1.

addendum for 6.) In a fact non-linear was missing two times in the manipulated version. The original version did contain non-linear only once at "a coherent non-linear optimization problem" (same paragraph). I have later on added the non-linear at "f.i." a second time to be absolutely sure that the word would not start to become missing again. This proves that the submitted version as I received it back from my professor equals the corrected version from later on and not the original version. The submitted version must thus have been changed itself at the university by the secret services and not my own version directly after submission.


924=924
925=925
926=926
927=927
928=928
929=929
930=930
932=931
934=932
935=933
936=934
937=935
938=936
939=937



```
  count_mv += l->count(); act_mv += l->activity / cla_inc;
}
for( i=0, j=0; i < sz; i++ ) {
  l = learnts_db[i];
  if( reason[var(l->lit(0))].clause == (Clause*)l ) keepIt =
  else if( i < youngClauseTreshold ) keepIt = l->count() < 8
  else keepIt = l->count() < 9 || l->activity > 60 * cla_inc;
  //printf("%i,",l->count());
  if(keepIt&&true) {
    learnts_db[j] = learnts_db[i];
    printf("%i=%i\n",i,j);
    j++;
  } else {
    // execute code for removing clause
    adjary = &(adj(-l->lit(0))->learnts);
    k = find(adjary->begin(),adjary->end(),l); assert(k!=adjary
    *k = *(adjary->end()-1);  adjary->pop_back();
    adjary = &(adj(-l->lit(1))->learnts);
    k = find(adjary->begin(),adjary->end(),l); assert(k!=adjary-
    *k = *(adjary->end()-1);  adjary->pop_back();
  }
}
//printf("count_mv=%lli, act_mv=%f, cla_inc=%f\n", count_mv/i, a
//printf("count_mv=%lli, act_mv=%f\n", count_mv/i, act_mv/i );
if(i-j) printf("cleaned up %i of %i\n", i-j, i ); else { putchar(
```

# Table of Figures

# 5 List of References


[Bc02] Fahiem Bacchus: Enhancing Davis Putnam with extended binary clause reasoning - AAAI-02 Proceedings/IAAI, 2002 – aaai.org

[BFR15] Armin Biere, Andreas Fröhlich: Evaluating CDCL restart schemes - Pragmatics of SAT, 2015 – yahootechpulse.easychair.org

[BFS15] Armin Biere, Andreas Fröhlich: Evaluating CDCL Variable Scoring Schemes - International Conference on Theory and Applications of Satisfiability Testing, SAT 2015: Theory and Applications of Satisfiability Testing -- SAT 2015 pp 405-422, 2015 – Springer

[BHMW08] Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh: Handbook of Satisfiability - Frontiers in Artificial Intelligence and Applications, 2009 - IOS Press

[Br04] Ronen I. Brafman: A simplifier for propositional formulas with many binary clauses - IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) ( Volume: 34 , Issue: 1), Feb 2004 – ieeexplore.ieee.org

[By86] Randal E. Bryant: Graph-based algorithms for boolean function manipulation - IEEE Transactions on Computers, Volume C-35 Issue: 8, 1986 – ieeexplore.ieee.org

[DvPt60] Martin Davis, Hillary Putnam: A Computing Procedure for Quantification Theory - Journal of the ACM (JACM), 1960 – dl.acm.org

[DwCD02] Adnan Darwiche: A compiler for deterministic, decomposable negation normal form - AAAI-02 Proceedings/IAAI, 2002 – aaai.org

[DwDN01] Adnan Darwiche: Decomposable negation normal form - Journal of the ACM (JACM), 2001 – dl.acm.org

[EeBi05] Niklas Eén, Armin Biere: Effective preprocessing in SAT through variable and clause elimination - International Conference on Theory and Applications of Satisfiability Testing, SAT 2005: Theory and Applications of Satisfiability Testing pp 61-75, 2005 – Springer

[EeS03] Niklas Eén, Niklas Sörensson: An extensible SAT-solver - International Conference on Theory and Applications of Satisfiability Testing, SAT 2003: Theory and Applications of Satisfiability Testing pp 502-518, 2003 – Springer

[GeKa07] Martin Gebser, Benjamin Kaufmann, André Neumann, Torsten Schaub: Conflict-Driven Answer Set Enumeration - International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2007: Logic Programming and Nonmonotonic Reasoning pp 136-148, 2007 – Springer

[GKAS07] Martin Gebser, Benjamin Kaufmann, André Neumann, Torsten Schaub: Conflict-Driven Answer Set Solving - IJCAI, 2007 – aaai.org

[GoNk07] Eugene Goldberg, Yakov Novikov: BerkMin: A fast and robust SAT-solver - Discrete Applied Mathematics, 2007 – Elsevier

[GuMi05] Yuri Gurevich, David G. Mitchell: A sat solver primer - Bull. EATCS 85: 112-132, 2005 – Citeseer

[HJS10] Yousef Hamadi, Saïd Jabbour, Lakhdar Saïs: Learning for dynamic subsumption - International Journal on Artificial Intelligence Tools, Vol. 19, No. 04, pp. 511-529 , 2010 - World Scientific

[JiSo06] HoonSang Jin, Fabio Somenzi: Strong conflict analysis for propositional satisfiability - Proceedings of the Design Automation & Test in Europe Conference, ISBN: 3-9810801-1-4, 2006 - IEEE Xplore

[JK] Jak Kirman: A modest STL tutorial, jak@cs.brown.edu, http://cs.brown.edu/people/jak/proglang/cpp/stltut/tut.html - Brown University

[KS05] Stefan Kuhlins, Martin Schader: Die C++ Standardbibliothek, Einführung und Nachschlagwerk, 4. Auflage, ISBN 3-540-25693-8, 2005 – Springer

[Lo13] Robert Love: Linux System Programming, Second Edition, ISBN: 978-1-449-33953-1, 2013 – O'Reilly

[Lu96] Dirk Louis: C und C++: Programmierung und Referenz, Haar bei München, ISBN 3-8272-5066-8, 1996 - Markt und Technik

[LyMS05] Inês Lynce, João Marques-Silva: Efficient data structures for backtrack search SAT solvers - Annals of Mathematics and Artificial Intelligence, 2005 – Springer

[MMZZ01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik: Chaff: engineering an efficient SAT solver - DAC '01: Proceedings of the 38th annual Design Automation ConferenceJune 2001 Pages 530–535, 2001 – dl.acm.org

[MS08] Norman Matloff, Peter Jay Salzman: The art of Debugging, with GDB, DDD and Eclipse, ISBN-13: 978-1-59327-174-9, 2008 - no starch press

[MqSv99] João Marques-Silva: The impact of branching heuristics in propositional satisfiability algorithms - Portuguese Conference on Artificial Intelligence, 1999 – Springer

[Ng13] C. K. Nagpal: Formal Languages and Automata Theory, ISBN: 0-19-807106-X, 2013 - Oxford University Press

[PiDw07] Knot Pipatsrisawat, Adnan Darwiche: A lightweight component caching scheme for satisfiability solvers - International Conference on Theory and Applications of Satisfiability Testing, SAT 2007: Theory and Applications of Satisfiability Testing, SAT 2007 pp 294-299 – Springer

[SBB04] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, Toniann Pitassi: Combining Component Caching and Clause Learning for Effective Model Counting - SAT, 2004 – Citeseer

[SBK05] Tian Sang, Paul Beame, Henry Kautz: Heuristics for Fast Exact Model Counting - International Conference on Theory and Applications of Satisfiability Testing, SAT 2005: Theory and Applications of Satisfiability Testing pp 226-240, 2005 – Springer

[So08] Rolf Socher: Theoretische Grundlagen der Informatik, 3.Auflage, ISBN 978-3-446-41260-6, 2008 - Hanser

[SuPr04] Sathiamoorthy Subbarayan, Dhiraj K. Pradhan: NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances - International Conference on Theory and Applications of Satisfiability Testing, SAT 2004: Theory and Applications of Satisfiability Testing pp 276-291 , 2004 – Springer

[Th06] Marc Thurley: sharpSAT – Counting Models with Advanced Component Caching and Implicit BCP - International Conference on Theory and Applications of Satisfiability Testing, SAT 2006: Theory and Applications of Satisfiability Testing - SAT 2006 pp 424-429, 2006 – Springer

[Zh97] Hantao Zhang: SATO: An efficient propositional prover - International Conference on Automated Deduction, 1997 – Springer

[ZMM01] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, Sharad Malik: Efficient conflict driven learning in a boolean satisfiability solver - IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281), 2001 – ieeexplore.ieee.org

[Ru00] benchmark examples rutgers:
http://archive.dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf/

[Ca00] benchmark examples cachet DQMR.zip:
http://www.cs.rochester.edu/u/kautz/Cachet/index.htm

[cl00] Clasp solver download: https://potassco.org

[ZC00] ZChaff solver download: https://www.princeton.edu/~chaff/zchaff.htm

[sS00] sharpSAT: https://sites.google.com/site/marcthurley/sharpsat/benchmarks/collected-model-counts

[sSS0] sharpSAT solver download: https://github.com/marcthurley/sharpSAT/archive/v12.08.1.zip

[c2d00] c2d d-DNNF solver download: http://reasoning.cs.ucla.edu/c2d/download2.php

[SR06] benchmarks SATRace 2006: http://fmv.jku.at/sat-race-2006/downloads.html

[BM00] benchmarks bmc: https://www.cs.cmu.edu/~modelcheck/bmc/bmc-benchmarks.html

[MS00] benchmarks miroslav: http://www.miroslav-velev.com/sat_benchmarks.html

[GCC0] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=95665