Finding cores using a Brouwer's fixed point approximation algorithm

Master's thesis

Siert Wieringa

August 2007

TU Delft

Delft University of Technology

Finding cores using a Brouwer's fixed point approximation algorithm

Master's thesis

Ing. S. Wieringa

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Program Computer Science

Committee:

Dr. K.P. Hart

Ir. M.J.H. Heule

Dr. H. van Maaren

Prof. Dr. C. Witteveen

August 2007

Acknowledgements

The research leading to this thesis has been an important part of my life over the last year. It could not have been realized without the help and support of friends and supervisors. I would first of all like to thank Hans van Maaren for coming up with a challenging and novel project and for sharing his insights on the project with me.

My friends Jan van der Meer and Minze Walvius have been very important to this project as they let me work at the shared office of their companies INOXA and Advier[1]. They deserve a lot of credit for facilitating me the way they did and for their daily company and support.

I would like to thank Keijo Heljanko and Ilkka Niemelä from the Laboratory for Theoretical Computer Science at the Helsinki University of Technology as they gave me time to finish this project after they decided to hire me as a PhD in their research group. I very much appreciate Keijo's successful attempts to find me a house in his country, which meant one important thing less to worry about while finishing this project.

Finally I would like to thank Marijn Heule and Cees Witteveen for their supervision and support.

[1] INOXA builds websites (www.inoxa.nl), Advier are mobility managers (www.advier.nl)

Abstract

In this Master's thesis a novel algorithm for finding unsatisfiable cores in sets of Boolean constraints is presented. The algorithm is based on a Brouwers' fixed point approximation algorithm, which makes its approach to the satisfiability problem unique.

An important part of this project was the development of a core finder, named DUC Hunt, which contains an implementation of the algorithm. DUC Hunt has various options that can be used to make it either find multiple cores or to make it find a guaranteed MUS. Using that last feature it can also be used as a "MUS prover" for cores found by other core finders. Although the algorithm is in theory applicable to finding cores in sets of Boolean constraints in general its current implementation is limited to finding cores in sets of clauses.

For the application of DUC Hunt to proving that an unsatisfiable formula is a MUS an approach that differs from the algorithm presented was found while evaluating the algorithm. This new approach resulted in a practically useful and well performing MUS prover.

Besides that application the current implementation of the algorithm works correctly but it is too slow to be of practical use. As this is a first study into the possibilities of applying a Brouwer's fixed point approximation algorithm to Boolean satisfiability this leads us to propose future research and implementation improvements.

Contents

1 Introduction

Suppose you are taking your two children Pete and John to the zoo. At some point we have the choice to go either left or right. Only if we will turn left we will see hippos and giraffes and only if we turn right we will see sea lions and flamingos. We can not do both. Describing this in propositional logic there will be constraints stating: not Left or not Right, Left implies Hippo, Left implies Giraffe, not Left implies not Hippo and so on.

Let us assume you want to see the giraffes, Pete wants to see the hippos and John would like to see the flamingos. Now you have a problem because in order to satisfy Pete's and your own wish you have to turn left and in order to satisfy John's wish you have to turn right. In other words if we add to the constraint "Giraffe and Hippo and Flamingo" to the formula it becomes unsatisfiable.

If we would leave the constraints stating that going right leads us to the sea lions and not going right will not out of the formula the formula will still be unsatisfiable. The sea lions were not part of the cause of the unsatisfiability and therefore this smaller set of constraints is an unsatisfiable subset of the original formula. This unsatisfiable subset is still not the smallest unsatisfiable subset possible. In this formula there are two unsatisfiable subsets of minimal size that identify the two problems that cause the unsatisfiability:

1 You want to see giraffes which requires turn left, John wants to see flamingos which requires turn right, Can not turn left and right.

2 Pete wants to see hippos which requires turn left, John wants to see flamingos which requires turn right, Can not turn left and right

Now we have identified the two problems we can find a way to resolve it, for example by attempting to convince John to change his preferences. One can imagine that for larger problems identifying the cause of the unsatisfiability by finding these *unsatisfiable cores* is very useful.

1.1 Boolean constraints vs Clauses

The novel algorithm presented in this document finds unsatisfiable subsets, or *unsatisfiable cores*, in sets of Boolean constraints. Most modern SAT solvers and hence the current core finders expect the input file to be in the conjunctive normal form (CNF). A CNF formula is a conjunction of clauses. A clause is a special type of Boolean constraint that consists only of disjunctions of literals. A literal is a Boolean variable or its negation. The current core finders therefore are capable only of finding unsatisfiable subsets of clauses in CNF formulas.

As conjunctions may also be part of Boolean constraints every possible subset of clauses in a CNF formula can be regarded as a single Boolean constraint. Therefore, our algorithm could in principle be used to find unsatisfiable subsets of sets of clauses taken from a CNF formula.

Although theoretically possible we face some implementation challenges in applying the algorithm to general Boolean constraints. For this first study into

the application of the algorithm to Boolean formulas the implemented software was limited to handling CNF formulas and thereby to finding unsatisfiable subsets in sets of clauses.

Throughout this document the algorithm is described in its most general form; as an algorithm for finding sets of "contrary" Boolean constraints. Most of the examples will be limited to CNF formulas in order to keep them simple. As the implemented software is limited to CNF formulas so is the description of its implementation and the presentation of its results.

1.2 Minimal unsatisfiable cores

A minimal unsatisfiable subset (MUS), also called a minimal unsatisfiable core, is a formula that has no unsatisfiable subformula. In other words it is an unsatisfiable formula that becomes satisfiable if any of its constraints is removed. A formula may contain multiple unsatisfiable cores. In the paper presenting the unsatisfiable core finder AMUSE [4] the importance of finding minimal unsatisfiable cores in general and finding multiple of those in particular is stated. This statement does not seem to be reflected in the implementation of AMUSE 0.7 as it does not necessary find a MUS. According to the paper finding different cores in multiple runs should be possible but it was not made clear how this feature should be used with the actual implementation of the software.

To prove that a core of m constraints is a MUS all m subformulas of $m-1$ constraints must be proven to be satisfiable. As this requires m calls to a SAT solver this is very time consuming. One of the results of recent studies into this problem is the "minimal unsatisfiability prover" MUP [5].

In this document we will show that an extended version of our algorithm is capable of finding guaranteed MUS'es and can therefore also be used as a MUS prover.

1.3 DUC Hunt

As stated in the previous paragraph during this project an implementation of the algorithm was made that was limited to finding unsatisfiable subsets of clauses in CNF formulas. The software was named DUC Hunt, which is an acronym for Dedicated Unsatisfiable Core Hunter.

Throughout this document there will occasionally be a block of text like the following:

DUC Hunt program detail In this type of block details of the implementation or possible optional settings of DUC Hunt regarding the subject that was explained in the paragraph it is contained in are described.

2 Theoretical background

This chapter describes the theoretical background of the developed unsatisfiable core finder. To explain the theorem on which the algorithm's functionality is based [1] we first need to introduce *Sperner's lemma* and *Brouwer's fixed point theorem*. All these introductions will be made in this chapter but for complete and formal definitions we refer to [7].

2.1 Sperner's lemma

Before introducing Sperner's lemma some terminology must be defined.

Definition An n-dimensional simplex is an n-dimensional analogue to a triangle, or formally a convex hull around $n+1$ independent points. An $(n-1)$-dimensional face of a simplex is called a facet. A triangulation of an n-dimensional simplex is a division of the simplex into smaller n-dimensional simplices in such a way that each pair of simplices intersects either in a common facet or not at all. In the 2-dimensional case a simplex is a triangle, a facet is a line and a triangulation is a triangle constructed from smaller triangles.

Definition An n-dimensional simplex has *Sperner labeling* if all of its corners are labeled with a label from 1 to n such that each label is used once. The points on the facets must be labeled with a label of one of the corners. All points inside the triangle can be given any of the n labels.

In figure 2.1 a triangulation of a 2-dimensional simplex (a triangle) is shown. For each possible assignment of the labels 1, 2 and 3 to the points labeled with a question mark this triangulation has a valid *Sperner labeling*.

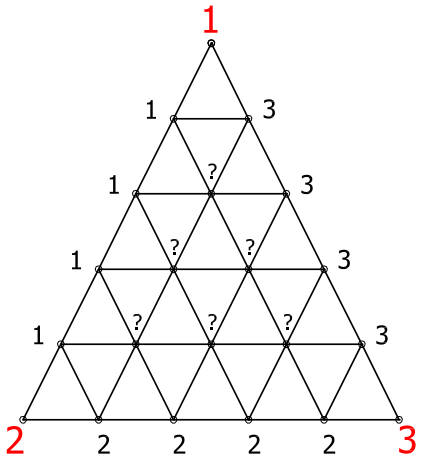

Figure 2.1: *Sperner labeling*

Theorem 2.1 (*Sperner's lemma*) *Every Sperner labeling of a triangulation of an n-dimensional simplex contains an odd number of smaller simplices labeled with a complete set of labels.*

A completely labeled simplex can be found deterministically. An example search for such a completely labeled simplex in the earlier presented 2-dimensional triangulation is shown in figure 2.2. The search starts by choosing two different labels, in this example these are the labels 1 and 2. One of the small triangles is entered from the outside of the larger triangle trough a "door". A door is an edge that connects two points that have the two chosen labels.

For each entered triangle it must be true that it is either completely labeled or it has exactly one other door which can serve as an entry to a neighboring triangle. The search will keep on moving to neighboring triangles until it ends up in a completely labeled triangle.

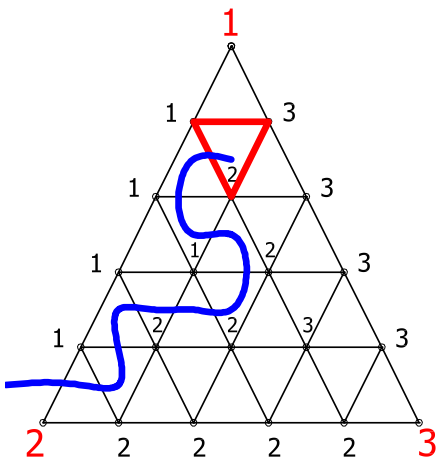

Figure 2.2: Walking trough edges with endpoints labeled 1 and 2

It is not possible to get back to an earlier visited triangle during the search. The search algorithm is therefore acyclic. We will show this by example. Consider the pictures in 2.3, these should be regarded as subsets of some triangulation with a valid Sperner labeling. The left picture shows the sort of construction that would be required to make looping possible. However that construction does not have a door on the outside and therefore can not be entered. A door can not be made in the construction without resolving the cycle.

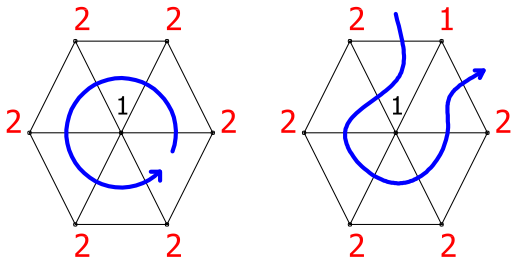

Figure 2.3: The algorithm is acyclic

2.2 Brouwer's fixed point theorem

Theorem 2.2 (*Brouwer's fixed point theorem*) *For every continuous function that maps a non-empty, compact convex set* Δ *of* $\mathbb{R}^n$ *into itself, there is a fixed point.*

$$\forall_{F:\Delta\to\Delta}\ \exists_x\ F(x) = x$$

Example If a can of paint is stirred without spilling the particles in the can are repositioned. In other words the old positions of the particles are mapped to new positions. From *Brouwer's fixed point theorem* it follows that no matter how long a can of paint is stirred after stirring there will always be at least one particle that is at the position it had before stirring.

We will limit ourselves to mappings of an n-dimensional simplex into itself. For the 2-dimensional case, a mapping of a triangle into itself, *Sperner's lemma* can be used to approximate a fixed point. In this paragraph we will show how, although only demonstrative and not meant to be a formal proof. These methods can be generalized to higher dimensions.

Definition Barycentric coordinates express a point in a triangle as a weighted average of the coordinates of the three vertices [6]. The coordinates form a triple that adds up to 1.

Let the barycentric coordinate of point x be (a, b, c) and the barycentric coordinate of point $F(x)$ be (a', b', c'). If a mapping F is possible for which $F(x) \neq x$ for all x then each point (a, b, c) in the triangle can be labeled as follows:

1 if a' $<$ a

2 if a' $\geq$ a, but b' $<$ b

3 if a' $\geq$ a and b' $\geq$ b, but c' $<$ c

No labeling is possible by this definition for a point for which $a = a'$, $b = b'$ and $c = c'$ which would be a *fixed point*. This labeling will label the corners of the triangle 1, 2 and 3 as required for a *Sperner labeling* as:

- The point in the corner with barycentric coordinate (1,0,0) will be mapped to a point with $a' < 0$ unless this is a fixed point. This corner will therefore get label 1
- The point in the corner with barycentric coordinate (0,1,0) has to be mapped to a point with $a' \geq 0$ and $b' < 1$ unless this is a fixed point and it will therefore get label 2.
- The point in the corner with barycentric coordinate (0,0,1) has to be mapped to a point with $a' \geq 0$, $b' \geq 0$ and $c' < 1$ unless this is a fixed point. It will thus get label 3.

Not only do the three corners get labels 1, 2 and 3 it can also be shown in a similar fashion that the points on the edges connecting two vertices can only get the labels of one of those two vertices. This labeling is therefore a valid *Sperner labeling*. Because of that each triangle must contain at least one smaller triangle that is completely labeled. If we continue to label the points within the small triangle we will find ever smaller completely labeled triangles. The diameter of those triangles will converge to 0. The positions of each vertex of the ever smaller triangles will form a sequence that converges to a single point. This will be the same point for all three vertices which is why that point can not be labeled and is therefore a *fixed point*.

2.3 Geometry free Sperner's lemma

The theorem by van Maaren [1] on which our unsatisfiable core finder is based can be regarded as a geometry free version of *Sperner's lemma*. It can not only be used to prove *Brouwer's fixed point theorem* but it also yields a constructive algorithm for approximating a *fixed point*, analogue to the way this can be done for triangles.

The theorem was initially applied to economic equilibria where finding a fixed point meant finding a coalition. Definitions for the theorem are:

- A set X
- A finite set P
- An ordering relation $\leq_p$ on X for each $p \in P$ satisfying:

reflexitivity:	$x \leq_p x$
transitivity:	$x \leq_p y$ and $y \leq_p z$ imply $x \leq_p z$
completeness:	$x \leq_p y$ or $y \leq_p x$

In the coalition finding application P might be regarded as a set of persons and $\leq_p$ as the preference relation of person p on the set X. No two alternatives are equally preferred.

Throughout this document we will use a graphical representation for these relations of which an example is shown in figure 2.4. The nine dots in that figure are nine different alternatives and the three lines represent three persons. The preference of a person is defined by the order in which the dots are encountered when shifting the person's line in the direction of the arrow attached to it. The further the line is shifted in the direction of the arrow the more preferred the encountered assignments are.

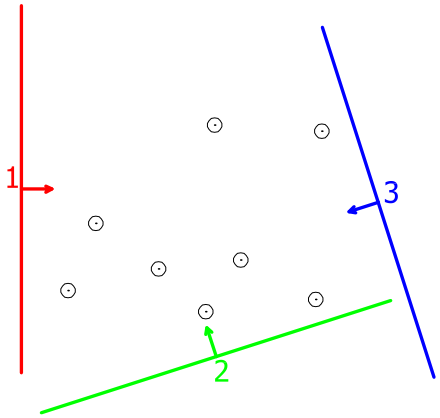

Figure 2.4: Three persons and their preferences over nine alternatives

A non-empty subset Q of P is a *coalition*. Each member of the coalition chooses an alternative a_q from the set A.

Definition A coalition *agrees* iff:

- All persons in the coalition prefer the alternatives chosen by all other members of the coalition

$$a_q \leq_q a_r \ for\ all\ q,\ r \in Q\ with\ q \neq r \tag{c2.1}$$

- There is no alternative all members of the coalition would prefer over their current alternative

$$\neg\exists_{a \in A}\ with\ a_q \leq_q a\ for\ all\ q \in Q \tag{c2.2}$$

The graphical representation of a person's choice is the line representing the person placed on the dot that represents the chosen alternative. In figure 2.5 a coalition of two persons that *agrees* is shown.

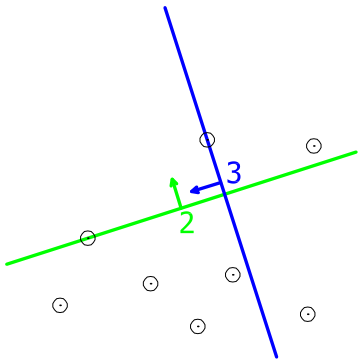

Figure 2.5: Persons 2 and 3 form a coalition that *agrees*

Each alternative is labeled with the name of a person by a labeling function L : A → P. Let A_Q be defined as:

$$A_Q = \{a_q | q \in Q\}$$

We have now completed a construction for which the following theorem holds:

Theorem 2.3 (*van Maaren's theorem*) *There exists an agreement consisting of a coalition Q in which each member of the coalition is also the label of a chosen alternative.*

$$Q = l(A_Q) = \{\ l(a)\ |\ a \in A_Q\ \}$$

Such an agreement is, analogue to *Sperner's lemma*, called a *completely labeled* crystal. An example of such a crystal is shown in figure 2.6.

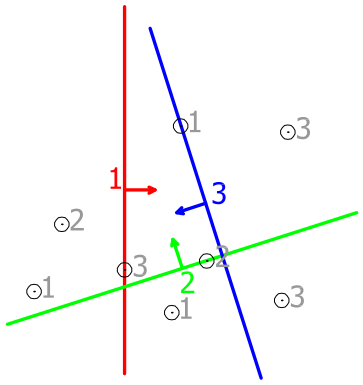

Figure 2.6: A completely labeled crystal

3 Application to Boolean formulas

In the previous chapter a theorem resulting in an algorithm for finding a completely labeled crystal in multiply ordered spaces has been introduced. In this chapter the application to the satisfiability of a set of Boolean constraints in general, and to finding unsatisfiable subsets of those sets in particular is described.

Our problem involves *assignments* that might *satisfy* a *Boolean constraint*. In the previous chapter we defined *alternatives* (or *points*), *ordering relations* and *labels*. In the application to satisfiability the set of *alternatives* is made up from all 2^n possible *assignments* to the n *variables* in the formula. Each Boolean constraint defines an *ordering relation*. Each assignment is labeled with the identifier of a Boolean constraint that it does not satisfy. Finding a *completely labeled* crystal means finding an unsatisfiable subset of constraints.

3.1 Ordering relations imposed by Boolean constraints

Each Boolean constraint in the formula must impose a unique ordering on the set of assignments. For the algorithm to function correctly it is necessary that all assignments that satisfy the constraint are preferred over all assignments that do not satisfy that constraint. The order within these two groups is not important for the correctness of the algorithm but might affect performance.

Two possible strategies to have a clause impose an ordering relation upon the set of assignments are described here and analyzed in the next paragraph. The challenge that needs to be faced in order to generalize this to Boolean constraints will also be described in the next paragraph.

Example Let us consider a possible ordering imposed by the clause $p \vee \neg q$ in a CNF formula with three variables, p, q and r.

	p	q	r	# sat
worst	0	1	0	0
1	0	1	1	0
2	0	0	0	1
3	0	0	1	1
4	1	1	0	1
5	1	1	1	1
6	1	0	0	2
best	1	0	1	2

Variable r does not occur in the clause which is why it was chosen to be the least significant variable in the order. The choice to prefer $r = true$ (1) over $r = false$ (0) should be considered to be a tiebreaker. The best assignments are those that satisfy both literals in the clause. For the assignments that satisfy one literal of the clause satisfying p was preferred over satisfying q which is a so-called lexicographic tiebreaker.

It seems very natural to use the number of literals satisfied in the clause as the main sorting criterion on the assignments. To complete it, assignments that satisfy an equal number of literals can be sorted lexicographically. The order in the previous example could have been generated that way. However, there is another form of sorting that leads to the same order; calculating the binary representation of the row number (counting from 0) and inverting the digit corresponding to q (as we want to prefer $\neg q$). The sorting of the example clause is thus a lexicographic sort of the assignments using literal preference $p\neg qr$. The literals in the clause are the most significant so the sorting of clause $q \vee r$ would be defined by qrp. We will refer to this sorting method as *fully lexicographic sorting*. Both methods presented are valid but they will not result in the same ordering for clauses with more then two literals.

3.2 Desired properties for ordering relations

It might seem that the sorting of the assignments by the number of satisfied literals in a clause is the most natural and therefore the most obvious choice. However, to give the algorithm a reasonable performance it must be computationally easy to determine wether one assignment is better or worse than another assignment given some ordering. Using the *fully lexicographic sorting* method this is very easy as the bits of the number representing the index in the sorted set can be determined directly from the assignment and so this comparison is in fact just the comparison of two binary numbers. Counting the number of satisfied literals first and if equal compare the literals not involved in the clause lexicographically would be more expensive. Such comparisons would also be harder to translate to a CNF formula, which might cause some problems with the practical implementation as will become clear further on in this document.

For general Boolean constraints it is still a question how they can impose an ordering over the assignments in such a way that the comparison of two assignments is cheap enough to actually be used in a practical implementation of the algorithm. This has not been thoroughly investigated yet as much attention was paid to getting the simpler implementation for CNF formulas right.

DUC Hunt program detail The *fully lexicographic sorting* method is used in DUC Hunt to determine the ordering relations. In the example on ordering relations presented earlier the variable r was not part of the clause that imposed the sorting on the assignments. A choice was made in that example to prefer $r = true$ over $r = false$. In DUC Hunt this choice can be influenced by the user by setting one of the three following program options:

- ***t*** Always prefer $true$ over $false$
- ***f*** Always prefer $false$ over $true$
- ***m*** For x_i prefer $false$ if $\neg x_i$ occurs more often than x_i in the input formula, $true$ otherwise.

The default settings is ***m***.

3.3 Assignment labeling

Since it is not feasible, nor necessary to label all 2^n assignments at forehand this is done the first time they are encountered by the algorithm. All assignments in the algorithm must be labeled with the identifier of a Boolean constraint that they do not satisfy. If we encounter an assignment that satisfies all constraints then the algorithm will stop instantaneously and the assignment will be recorded as a satisfying assignment for the formula. As this is the only reason apart from finding an unsatisfiable core for which the algorithm will stop it is also a *complete solver* for Boolean formulas.

It is important that once an assignment has been given a label it keeps this label as the algorithm continues. In most cases there will not be a problem with label consistency as, unless otherwise specified, the algorithm will always assign the identifier of the first Boolean constraint in the input formula that is does not satisfy. We will refer to that type of labeling as *straightforward labeling*.

3.4 The completely labeled crystal

In a completely labeled crystal (recall theorem 2.3) the set of labels on the alternatives chosen must be equal to the set of members of the coalition. In the application to satisfiability we have Boolean constraint as coalition members and assignments labeled by constraint identifiers, so each constraint that is part of the coalition must have its identifier on one of the assignments chosen by the coalition.

Each constraint in a completely labeled crystal is not satisfied by the assignment it is placed on. This is because each constraint has its identifier either on the assignment it is positioned at itself or on an assignment that it would prefer over its current choice (by condition c2.1) and the label states that assignment is not satisfying the constraint.

All constraints will prefer a satisfying assignment over their current choice. However, since they all prefer all other constraints' choices as well, a satisfying assignment for the formula must be inside the crystal. The crystal however is empty (by condition c2.2) which means that the set of constraints that forms the coalition is unsatisfiable and is therefore an *unsatisfiable core* of the input formula.

Example Consider the following formula in CNF format

- $p \vee q$
- $\neg p \vee q$
- $\neg q$

As the formula is in CNF format the three Boolean constraints are in fact *clauses*. In this example the preference of the clauses was defined by *fully lexicographic sorting* to be:

$\leq_1$ { **11**, **10**, **01**, 00 }

$\leq_2$ { **01**, **00**, **11**, 10 }

$\leq_3$ { **00**, **10**, 01, 11 }

In all examples in this document involving the variables p and q the digit sequence 11 refers to the assignment $\{p = 1, q = 1\}$. The satisfying assignments are **printed bold** and as these come first these orderings are valid. This preference relation is also defined by the tilting of the lines representing the clauses in figure 3.1. From the figure it can be seen how a completely labeled crystal represents the unsatisfiability of this formula.

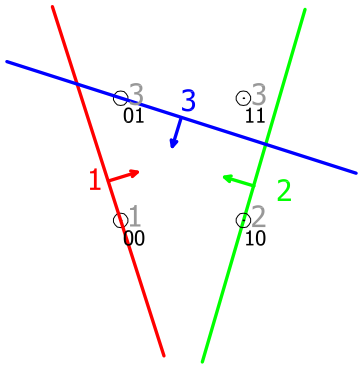

Figure 3.1: Example core

In the core found in the example formula all constraints are positioned at assignments with their own label. In principle it does not have to be that way as long as all labels on the assignments occur as a label on one of the clauses. In this example however it was inevitable as a consequence of the applied *straightforward labeling* (see 3.3).

Lemma 3.1 *Using straightforward labeling each constraint l in a completely labeled crystal will be on an assignment labeled l*

Proof In a completely labeled crystal each constraint is positioned at an assignment that is not satisfying the constraint. All assignments with a label smaller then a constraints' label satisfy that constraint. As the set of labels on the constraint in a completely labeled crystal is equal to the set of labels on the assignments the condition that each constraint is at an assignment with a label larger than or equal to the constraint label can only be satisfied for all constraints if each constraint is at the assignment with its own label.

4 The algorithm

In the second chapter van Maaren's theorem was introduced and it was stated that this theorem can be viewed upon as a geometry free version of *Sperner's lemma*. In the example on *Sperner's lemma* a "walk" trough the large triangle was shown that ended in a completely labeled smaller triangle. This chapter describes an algorithm that is an analogue to the "triangle walk" that finds a completely labeled crystal. In the previous chapter it was demonstrated how a completely labeled crystal can represent an unsatisfiable subset of Boolean constraints. As that is our goal the algorithm is described here with that application in mind.

4.1 Algorithm basics

The algorithm starts by picking a Boolean constraint from the set. Unless otherwise specified we always select the first constraint as the start constraint. Please note that this is without loss of generality as the input set might be reordered freely. The algorithm will be explained by example.

Example Recall the example CNF formula presented in paragraph 3.4:

- $p \vee q$
- $\neg p \vee q$
- $\neg q$

A completely labeled crystal for the formula in the assignment space of variables p and q was shown in figure 3.1. In this example it is shown how this crystal was found by the algorithm. Each of the pictures in figure 4.1 illustrate a step that is described here.

A) Clause 1 ($p \vee q$) is introduced at its most preferred assignment, $\{p = 1, q = 1\}$. The label for that assignment is determined to be 3 as $\neg q$ is the first clause that it does not satisfy. We now have $Q = \{1\}$ and $l(A_Q) = \{3\}$. The only possible step towards equality of the sets is to introduce clause 3.

B) Clause 3 has to be introduced at the assignment in A_Q that it prefers the least. As the only element in A_Q is the position of clause 1, clause 3 is placed there. Clause 1 has to move backwards to the next best assignment that satisfies condition c2.1. The new position for clause 1 is the assignment $\{p = 1, q = 0\}$ which becomes labeled with a 2 because it does not satisfy $\neg p \vee q$. Clause 2 will need to be introduced in the next step.

C) Clause 2 is introduced to the coalition forcing clause 1 to move further backwards to the assignment $\{p = 0, q = 1\}$. This results in a situation where label 3 occurs twice in A_Q. It has to be handled in the next step.

D) The clause placed at the "old" assignment with label 3 is moved forward to the next better position. In this case it is clause 3 which moves forward and its new position is the current position of clause 1. After the required backwards shift clause 1 ends up on an assignment that is labeled 1 making the sets Q and $l(A_Q)$ equal and thus an unsatisfiable core has been found.

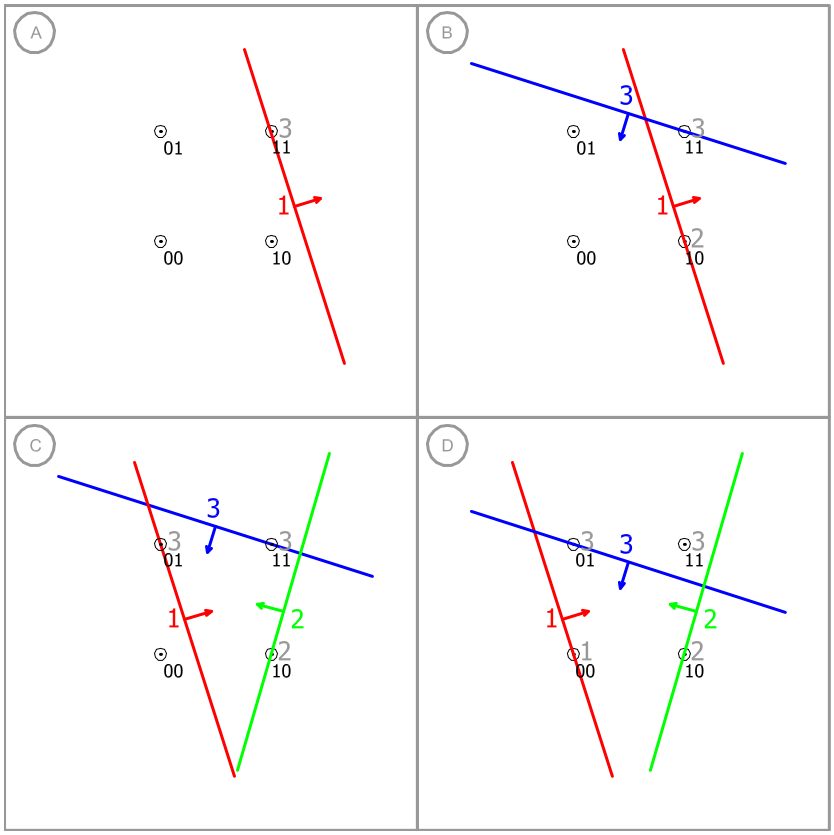

Figure 4.1: Example run of the algorithm

After showing an example run we now state six rules that constitute the algorithm:

1) The algorithm starts by positioning constraint 1 at the assignment it prefers the most and determining the label for that assignment

2) If an assignment with a label l is encountered with constraint $l \notin Q$ then constraint l is positioned at the assignment in A_Q that it prefers the least

3) If an assignment with a label l is encountered that was already in $l(A_Q)$ then the constraint on the earlier encountered assignment with label l is moved forward to the next better position in A_Q.

4) If a constraint is positioned at an assignment a at which another constraint is already positioned ($a \in A_Q$) the constraint that was already there is moved backwards (that is, in the direction it prefers less) to the best alternative assignment that satisfies condition c2.1.

5) If in the case described at point 4 no such alternative assignment exists the constraint is removed from the set Q. In other words it goes *passive*. If the constraint is not constraint 1 there will be an assignment with the corresponding label in A_Q. The constraint that is at that assignment will have to move forward to the next better position in A_Q.

6) Whenever an assignment with label 1 is encountered (1 is added to $l(A_Q)$) or the constraint with label 1 goes passive (1 is removed from Q) the sets $l(A_Q)$ and Q will be equal and a core has been found.

Please note that the scenario described in rule 5 did not occur in the example.

4.2 Finding multiple cores

In most input formulas there will be multiple unsatisfiable cores. With some simple extensions the core finder is able to continue the search after the first found core in order to find more cores.

As stated before if an assignment is encountered with label 1 a core is found and the algorithm ends. As each assignment is labeled with the identifier of the first constraint that it does not satisfy the algorithm will always assign label 1 if possible. However, if we encounter an assignment that is amongst other labels entitled to label 1 we could also record that the current set Q is an unsatisfiable subset of Boolean constraints and then give the assignment the first possible label other than 1 in order to continue.

There are more options to extend the search. In paragraph 3.3 it was noted that if an assignment is given a label it should keep this label. It is possible to investigate multiple labeling options by adding a backtracking mechanism to allow undoing everything that was done after the choice of an assignment label where multiple options where possible. After backtracking the assignment can be given another label and the algorithm can continue, thus investigating a new part of the search space and thereby possibly finding new cores. Backtracking can be done each time an assignment is only entitled to label 1 or if constraint 1 goes passive. In principle it is possible to store every labeling operation with multiple options so that the algorithm can backtrack every time it runs out of options and thus attempt all labeling options that are possible after the initial introduction of constraint 1.

In general, using our core finder to find multiple cores means we go beyond *straightforward labeling*.

DUC Hunt program detail The maximum number of levels to backtrack by DUC Hunt can be set using the parameter $\boldsymbol{l}$. If $\boldsymbol{l}$ is set to 0 the program will backtrack all the way to the first assignment labeling.

5 Minimal unsatisfiable cores

The output of our core finder as presented until now might or might not be a minimal unsatisfiable core. In this chapter we will describe extensions to our algorithm which will make it applicable to finding minimal unsatisfiable cores.

5.1 MUS Test

The MUS test is an extension to our core finder that can prove constraints necessary in a minimal unsatisfiable core. We will refer to a constraint that is proven by the MUS test to be part of a MUS as a *marked constraint*. The MUS test only works if *straightforward labeling* is used.

When a completely labeled crystal is found we have a set of constraints positioned at a set of assignments. To mark a constraint it must be proven that all other constraints together form a satisfiable formula. This is at least true for the constraint at the assignment with the highest label because that assignment satisfies all the other constraints as it would have gotten a lower label otherwise. Because of lemma 3.1 the constraint positioned at the assignment with the highest label is the constraint with that label. So, whenever a core is found the highest numbered constraint can be *marked*.

In general a constraint can be *marked* after testing if all higher numbered constraints are satisfied by the assignment it is positioned at.

Example Let us once more recall the core in the example formula as it was displayed in the figures 3.1 and 4.1. Clause 3 ($\neg q$) has the highest label which means it can be marked without further testing. To test if clause 2 can be marked we need to check if clause 3 would be satisfied by the assignment clause 2 is at. That assignment is $\{p = 1, q = 0\}$ and this is satisfying for clause 3 so clause 2 becomes marked as well. Finally, clause 1 can also be marked as both higher numbered clauses are satisfied by the assignment $\{p = 0, q = 0\}$. Because all clauses became marked this core is a MUS.

5.2 Finding a MUS by iterating

In the example in the previous paragraph all constraints became marked and the core was therefore proven to be a MUS. Unfortunately in larger formulas it is unlikely that all constraints that are necessary in a MUS are marked at once. If we want to use the algorithm to find a MUS we can iterate it a couple of times, that is the core including the marks can be used as input in a new run of the algorithm until all constraints are marked.

To ensure that new marks are found on each iteration until a MUS is found the marked constraints are placed at the top of the input file at each new iteration. If a new core constitutes more constraints than are currently marked the largest label will be larger than the label of the largest marked constraint. As the constraint with the largest label in the core can always be marked this means that at least one constraint gets a new mark in each iteration until all constraints are marked.

DUC Hunt program detail To find a MUS by iterative calls of DUC Hunt use either the option ***mT*** ("MUS top") or ***mS*** ("MUS sort"). Using option ***mT*** DUC Hunt will place marked clauses at the top of its output file and indicate how many clauses where actually marked. All clauses that are not marked appear under the marked clauses in the order they had in the original input file. If DUC Hunt reads this output file as its input file in a new iteration it first reads the line indicating which clauses were already marked. Option ***mS*** is the same as ***mT*** apart from placing all unmarked clauses in the output file in the order opposite to the order they had in the input file.

5.3 Finding a MUS without iterating

It is possible to do a *hot restart* every time a core is found until all constraints become marked. Using that technique a core will be found without iterating the program and letting the search start from scratch on the reduced formula.

In a *hot restart* all constraints that are not in a core are deleted from all datastructures of the program. The constraints that are part of the core are placed in a set in which all constraints are sorted by their label except for one unmarked constraint c_f which is placed last in the set. They are then given a new label which is their index in the sorted set. The assignments that all constraints except for c_f are positioned at must be relabeled with the new label of the constraint. This will still be the same label as would be determined by relabeling using straightforward labeling as the constraints proceeding it are still in the same order only with the unnecessary constraints and c_f missing.

The label of the assignment constraint c_f is positioned at must be redetermined. The fact that the constraints did not get marked by the MUS test means that the assignment satisfies a constraint that used to have a larger label. As c_f now has the highest label the new label for the assignment will be the label of one of the lower numbered constraints. After that label has been determined the corresponding constraint has to move forward, which is where the normal algorithm execution continues. One important thing has changed: Label 1 is now both on the set of assignments and on the set of orderings and its special status has been taken over by the last constraint, c_f. This means that the next core will be found if c_f goes passive or an assignment with c_f's new label, which is equal to the number of constraints left, is found.

Example Figure 5.1 presents an example core found in an unspecified formula in which the constraints 5, 7, and 12 are marked. A mark is indicated in the figure by associating an asterisks (*) with the constraint.

Figure 5.2 is the result of the *hot restart* where c_f was chosen to be the unmarked constraint with the highest label, constraint 9. That constraint becomes the highest numbered constraint in the new situation, constraint 6.

Now that this core has been constructed the algorithm can continue to reduce the core or mark more constraints by determining the label of the assignment constraint 6 is positioned at, which is labeled with a question mark in the figure. By construction that label will be smaller than 6 and the constraint associated

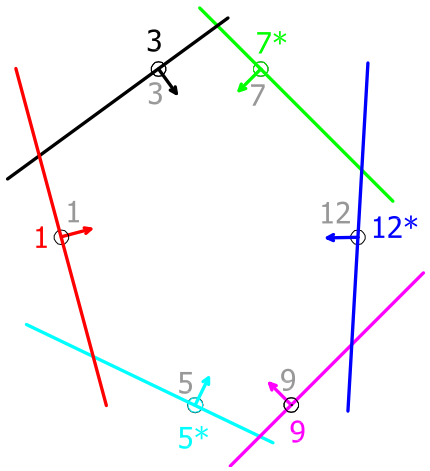

Figure 5.1: A core

with that label will have to move forward. From that point on the algorithm continues as it would normally do and a new core will be found once 6 appears as the label of an assignment or constraint 6 goes passive.

DUC Hunt program detail In DUC Hunt if the *hot restart* option is used the unmarked clause c_f is always chosen to be the unmarked clause with the largest label as was done in the example presented here. To use *hot restart* with DUC Hunt use the option ***mC*** ("MUS continue").

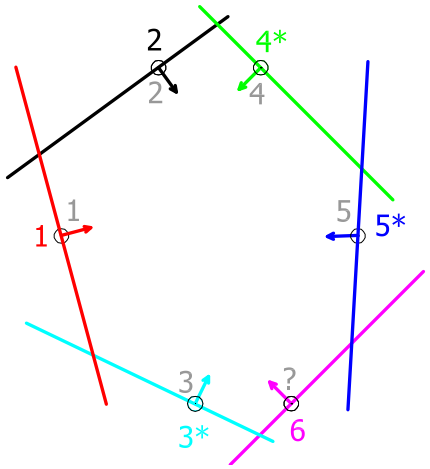

Figure 5.2: The same core after the *hot restart*

6 The implementation of DUC Hunt

In this chapter it is described how the algorithm was implemented in our software, DUC Hunt. Different implementation would of course be possible.

6.1 Shifting a clause backwards

The only computationally hard step in a run of the algorithm is the backwards shift. A clause has to be shifted backwards to a position it prefers less if another clause is placed at its current position. The new position has to be the most preferred assignment that all other clauses would prefer over their current position. The new assignment should therefore be the best possible assignment under a whole set of constraints, all of the type that require comparison of assignments under the preference relation imposed by the clause that is being shifted.

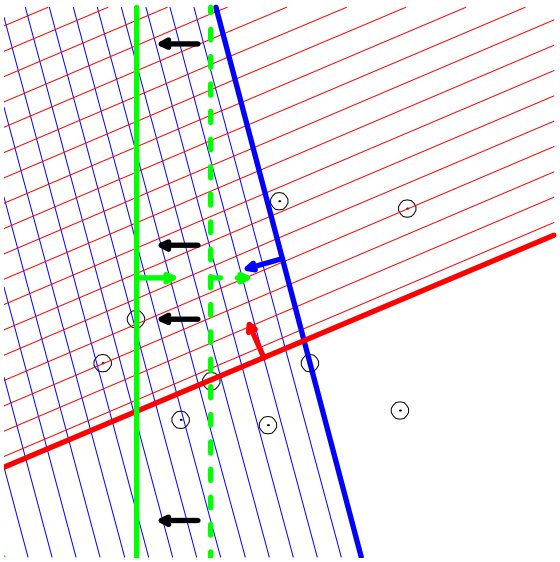

Figure 6.1: A clause forced to move. The new assignment must be in the area hatched twice.

In DUC Hunt constraints on backwards shifts are translated to Boolean CNF formulas, as this allows the program to find the new assignment by using a SAT solver. The formula might be unsatisfiable as it is possible that a clause goes passive because there is no assignment satisfying the constraints. As stated in paragraph 3.1 the translation to CNF would become very complicated if the number of satisfied literals in a clause would be taken into account which is why *fully lexicographic sorting* is used.

A constraint stating that the only satisfying solution is one that is larger than some binary number can be written by constructing a set of clauses that requires a digit that is zero in the binary number to become one while disallowing

any leading one to become a zero. This technique is used for constraints that state that all clauses must prefer the new position of the shifted clause over their own position. It can also be used to write the constraint stating that the binary representation of the new position of the shifted clause has to be smaller than the old position by requiring the two's complement[2] of the binary number to become larger.

6.2 Solving the shift constraints formula

After a formula with constraints for a backwards shift has been constructed it has to be solved. This is not just about finding a solution, it is about finding the satisfying assignment that is the most preferred by the shifted clause.

A SAT solver has to make choices on which variable assignments to investigate. By fixing the variable decision order of a SAT solver to the order in which our set of assignments is sorted the first satisfying assignment to be found will always be the best. Unfortunately, this degrades the performance of the SAT solver enormously.

DUC Hunt uses a modified version of MiniSat 2.0 [3] to solve the backwards shift formulas. It does not use the simplifier or restart techniques. The latter would bring the program in a loop since the branch order is predetermined. MiniSat's modified source code has been compiled into the DUC Hunt program. This does not mean that using MiniSat from within DUC Hunt is without overhead costs, as the programs do not use the same data structures.

Not all backwards shifts are hard so to minimize calls to MiniSat our program does its own iterative unit propagation and attempts to solve the formula using a very basic internal SAT solver first. MiniSat is involved after the internal solver has backtracked more times than the value set by the user for the parameter $\boldsymbol{b}$. The default setting for that parameter is 100.

[2]The binary representation of a negative number. Calculated by inverting all digits and adding 1 to the result

7 Results

In this chapter results of the DUC Hunt core finder are presented. To recall the program settings mentioned in this chapter please refer to the previous chapters or the summary in appendix A. The test set consists of two sets of 50 unsatisfiable random 3-SAT formulas at the threshold, one set with 50 variables and 215 clauses and a set of 60 variables and 258 clauses, in the result tables those sets will be referred to as set 1 and set 2 respectively. Unfortunately DUC Hunt's performance in these tests did not allow us to test larger benchmarks.

7.1 Finding cores

The first test was meant to demonstrate DUC Hunt's core finding abilities and discover the best possible tiebreaker decision option. The maximum backtrack depth was set to unlimited which means that the program can attempt all possible assignment labeling options which of course takes a very large amount of time if executed until the end. The program was therefore constrained with a fifteen minute time limit and the results here show statistics on the cores found in that time.

Tiebreaker type	set	shifts	minisat shifts	shifts to 1st core	cores	smallest core size
f	1	78387	28942	956	131	148
m	1	74826	33794	831	133	137
t	1	70843	26601	871	124	150
f	2	47561	22447	1387	122	186
m	2	40987	22590	1323	131	171
t	2	38428	18904	1349	113	184

Table 7.1: Results for the three different tiebreaker modes

All statistics in the table indicate that ***m*** is the best tiebreaker option. The number of shifts possible within the fifteen minute time limit is the largest while the first core was found after the smallest number of shifts, the largest number of cores was found and the smallest core was found by this version.

For the tiebreaker option ***m*** results of just finding the first core are also available. Unfortunately it must be noted that the algorithm does not scale very nicely as the CPU time is 180% higher after increasing the variable and clause count by 20%.

Tiebreaker	set	shifts	minisat shifts	size	CPU time
m	1	831	340	143	7.9s
m	2	1323	602	177	22.2s

Table 7.2: Statistics up to the first core found usign tiebreaker mode ***m***

The unsatisfiable core finder AMUSE [4] finds a core in these benchmarks within tenths of seconds. The cores it find have an average size of 141 clauses for the 50 variable formulas and 177 clauses for the 60 variables. So, the core sizes are comparable to DUC Hunt's but the time spend to find them is much smaller. By iterating AMUSE these cores may become even smaller.

The advantage of DUC Hunt in this mode is that it finds a large set of cores. In an industrial setting where some action must be taken in order to make the formula satisfiable multiple cores might suggest multiple ways of resolving the unsatisfiability which can be very useful as not all parts of the described real life system will be equally easy to modify.

None of the cores found according to the statistics in the tables presented here are supersets of other cores as the DUC Hunt program checks this. It is however possible that the intersection of two cores, that is the set of clauses that appear in both cores, is unsatisfiable. In that case the cores are not really unique as they are both a superset of the same core. The pairwise comparison of all cores found to detect this is rather time consuming so this has not been done for all cores found. For illustrative purposes it has been done for one set of cores, namely the cores found in set 1 by version ***m***. It turned out that on average 89 out of the on average 133 cores found are truly unique in the sense that their intersections are satisfiable.

7.2 Finding a MUS

DUC Hunt can be used to find a MUS in an unsatisfiable formula. Three "marker modes" are possible. All MUS marker modes where first tested using the same set of benchmarks that was used in the previous paragraph.

MUS marker mode	set	shifts	minisat shifts	calls or iterations	MUS size	CPU time
mT	1	7107	2898	13	113	55.8s
mS	1	7839	3216	14	114	61.5s
mC	1	95673	37435	79	113	825.4s
mT	2	14217	6971	17	143	206.7s
mS	2	15023	7390	18	142	221.8s
mC	2	>173731	>77919	101	141	>2313.7s[3]

Table 7.3: Results for the three MUS modes

The modes ***mT*** and ***mS*** require the program to be iterated a couple of times. The results in this table take that into account as the results of the separate iterations where summed for each benchmark before the average for all benchmarks was determined.

[3]The averages in this row were calculated over only 44 benchmarks as there were 6 benchmarks that were not solved within one hour.

The MUS marked mode ***mT*** seems to be the fastest in finding a MUS. As there might be multiple MUS'es in a formula using the different modes different MUS'es may be found which is why the average MUS sizes vary a little. Although it seems much more elegant than starting from scratch each time the MUS marker mode ***mC***, which uses hot restart, is much slower then the modes ***mS*** and ***mT***.

In the next table MUS'es found in the benchmark sets serve as input to DUC Hunt. With this test we can get an idea of how many iterations it takes to get all clauses marked in a MUS. The MUS'es used are not necessarily the same as the ones found by DUC Hunt in the previous test, as one formula may contain multiple MUS'es and the whole set of original benchmarks and MUS'es was put together at once.

The modes ***mT*** and ***mS*** take only one iteration less on average compared to using the original file as input while their total CPU time is nearly cut in half. In the ***mC*** mode the influence on the number of iterations is larger as it is approximately 30% lower compared to the original benchmarks for both sets. This is probably because during a run in mode ***mC*** with a formula that is not a MUS as input the size of the last core found decreases only very slowly while in the other two modes the iteration of the program makes the core size decrease faster as it builds up a new core from the input every time.

MUS m. mode	set	shifts	minisat shifts	calls or iterations	MUS size	CPU time
mT	MUS'es of 1	4729	1914	12	113	32.4s
mS	MUS'es of 1	5079	2032	14	113	34.8s
mC	MUS'es of 1	60245	23755	54	113	461.3s
mT	MUS'es of 2	8906	4151	16	142	107.5s
mS	MUS'es of 2	9340	4342	17	142	112.5s
mC	MUS'es of 2	116352	53166	70	142	1554.6s

7.3 Satisfiable formulas

Although not its intended use, the DUC Hunt program is also capable of finding a satisfying assignment for satisfiable formulas. Some luck is involved with finding a satisfying assignment fast as it is mainly influenced by whether there is a satisfying assignment close to the assignment where we first start introducing clauses. All three benchmark sets contained 50 satisfiable random 3-SAT formulas.

Benchmark set	shifts	hard shifts	CPU time
50 vars, 215 clauses	388	101	3.8
60 vars, 258 clauses	633	227	13.3
100 vars, 430 clauses	1316	732	175.1

8 A different approach

It is possible to construct a core before any shifts have been performed by doing a *hot start*. This option should not be confused with the *hot restart* option described in paragraph 5.3. In this chapter it will be shown that using the hot start the problem of finding a MUS can be approached differently.

8.1 Hot start

The first constraint in the input set is placed at the most preferred assignment that is not satisfying the constraint, which is the assignment that has the best value according to the tiebreaker for all variables not included in the constraint and the worst for all variables included in the constraint.

The position of every other constraint in the formula is determined by a call to a SAT solver. The input formula for that call is a formula that states that all previous constraints must be satisfied and this constraint may not be satisfied. If there is a satisfying assignment for that formula then that assignment will become the position of the constraint we are trying to add. The tiebreakers on the ordering imposed by that clause will be set accordingly to make sure the assignment is the best unsatisfying assignment. Clauses for which an unsatisfying assignment that satisfies all earlier added clauses can not be found are not part of the core.

Example Consider the following CNF formula

- $p \vee r$
- $p \vee \neg q$
- $q \vee \neg r$
- $\neg p$

The first clause has to be on an assignment with $p = false$ and $r = false$ as the clause may not be satisfied. The best assignment for q is determined by a tiebreaker to be $false$. In fact it is now determined that the sorting imposed by the first clause should be a lexicographic sort by $pr\neg q$.

The second clause needs to be on an assignment that satisfies clause 1 and has $p = false$ and $q = true$. The only possible assignment that satisfies these two constraints is the assignment $p = false, q = true, r = true$. As we want this assignment to be the assignment clause 2 prefers most amongst the assignments that do not satisfy it the tiebreaker for r is forced to prefer $true$ over $false$. The ordering relation imposed corresponds to lexicographic sorting by $p\neg qr$

To not satisfy the third clause $r = false$ and $q = true$ is required. The tiebreaker for p must be set to prefer $true$ over $false$ in order to satisfy the two other clauses. The ordering relation imposed corresponds to lexicographic sorting by $q\neg rp$.

In order to not satisfy the fourth clause p should be $true$, this already satisfies the first two clauses. The only requirement for the assignments to q and r is that it satisfies $q \vee \neg r$. An example of such assignment could be $q = true$ and $r = true$ which would make the ordering relation imposed by clause four correspond to a lexicographic sort by $\neg pqr$

If the four clauses are placed on the determined assignments and we respect the implied ordering relations then we have constructed an unsatisfiable core.[4]

The hot start should not be used for satisfiable formulas as the satisfying assignment will be inside the coalition, and therefore condition c2.2 will not be satisfied.

DUC Hunt program detail To use the hot start options with DUC Hunt use the ***h*** option in combination with one of the other three MUS marker modes. The marker mode combined with the hot start option will determine what to do after the hot start core has been generated. If hot start is used then for all variables that are not part of the clause and whose tiebreaker is not affected by a SAT solver the tiebreaker is set to prefer $true$ over $false$.

8.2 Finding a MUS without shifting

After building a core using the hot start method the hot restart function can be called in order to make the solver continue by shifting. Finding a MUS using only the hot restart option performed very badly in the results shown in paragraph 7.2. Extended with a hot start it becomes even worse as none of the benchmarks from the set used in that earlier paragraph where solved within one hour.

The potential of the hot start method however is in the possibility of finding a MUS without any shifting. This can be done by performing the MUS test on the core built using the hot start and then use that set of clauses and their marks as input to the next iteration of the algorithm until all clauses become marked.

This repeated hot start method of finding a MUS might be best suited as a MUS prover, especially since it should not be used on satisfiable formulas, which means that it is required to involve a SAT solver or core finder to prove unsatisfiability of the formula first anyway. It has been tested using four sets of 50 unsatisfiable random 3-SAT formulas and a set of benchmarks that models product configuration options for Daimler-Chrysler's production lines. This last set was added because they are large benchmarks with only a small MUS contained in them, which makes them very different from the randomly generated formulas.

[4] **Note added as postgraduate:** If for the first clause we choose to prefer $q = true$ the ordering of clause 2 does not rate clause 1's assignment over clause 2's assignment which it should do in order to reach an empty completely labeled chrystal. The constructed set of clauses however is still unsatisfiable. Future research on this approach will place it in a different perspective which will make it easier to prove correctness.

The DaimlerChrysler benchmarks where also used for the papers on AMUSE [4] and MUP [5]. The AMUSE core finder performs best for large benchmarks with small MUS'es as it builds MUS'es up from zero clauses. Finding MUS'es in such benchmarks using DUC Hunt in one of the modes other than repeated hot start will take a huge amount of time as it will introduce many clauses to the core that do not contribute to the unsatisfiability of the formula. For DUC Hunt's repeated hot start mode it is not ideal either.

The column "original" in table 8.1 shows the result of finding a MUS in the original formula. The results in the column "mus" describe the results of using the MUS that was found as input to analyze MUS proving performance. The hot start option ***h*** is combined here with either ***mT*** or ***mS***, meaning that the unmarked clauses are placed in the original order or in the order opposite of the input file in each intermediate output file used as input to the following iteration.

Mode	Benchmark set	original		MUS	
		iterations	CPU time	iterations	CPU Time
h + mS	50 vars, 215 clauses	38	0.73s	36	0.43s
h + mS	60 vars, 258 clauses	49	1.28s	46	0.91s
h + mS	150 vars, 645 clauses	127	56.19s	120	32.49s
h + mS	200 vars, 860 clauses	163	408.07s	158	190.18s
h + mS	DaimlerChrysler	10	33.15s	5	0.35s
h + mT	50 vars, 215 clauses	40	0.75s	38	0.45s
h + mT	60 vars, 258 clauses	50	1.32s	47	0.91s
h + mT	150 vars, 645 clauses	126	54.82s	119	32.13s
h + mT	200 vars, 860 clauses	164	405.78s	155	188s
h + mT	DaimlerChrysler	9	52.7s	5	0.32s

Table 8.1: DUC Hunt results for the hot start

It does not seem to make much difference wether ***h*** is combined with ***mT*** or ***mS***.

8.3 Results compared to simple prover

The simplest approach to proving a core to be a MUS without our program requires solving a number of satisfiability problems equal to the number of clauses in the formula as described in paragraph 1.2. This method can not only be used for proving a formula to be a MUS but also to find a MUS. To do this the prover must not stop when it finds an unsatisfiable subformula but instead continue to test if that subformula is a MUS.

Table 8.2 shows the sum of the run times of the SAT solver MiniSat 2.0 while used for finding and proving MUS'es in this way. The time required to generate the subformulas that are solved by MiniSat has not been taken into account as removing a clause from a formula should in theory be trivial. In our tests we

read a formula from a file on the disk, remove a clause and store it to another file. This is not the most efficient implementation and it would therefore not be fair to take it into account.

The rightmost two columns of the table restate the run times of DUC Hunt from table 8.1 for convenience.

Our repeated hot start method is in general faster at both the task of proving and of finding a MUS then the simple approach is. Please note again that no overhead costs where taken into account for the simple prover. For the set of benchmarks with the largest number of variables our approach is much faster which seems to indicate that it scales better. It should be noted that the DUC Hunt program was not written with this application in mind and that improvements should be possible. This therefore seems to be an interesting result that can lead to a practically applicable MUS prover.

Benchmark set	Simple prover		DUC Hunt $\boldsymbol{h}$ + $\boldsymbol{mT}$	
	original	MUS	orginal	MUS
50 vars, 215 clauses	3.99s	2.01s	0.75s	0.45s
60 vars, 258 clauses	4.88s	2.57s	1.32s	0.91s
150 vars, 645 clauses	70.6s	30.16s	54.82s	32.49s
200 vars, 860 clauses	746.05s	282.37s	405.78s	188s
DaimlerChrysler	-	1.96s	52.7s	0.32s

Table 8.2: Simple prover MUS test v.s. DUC Hunt $\boldsymbol{h}$

For the DaimlerChrysler benchmarks there is no result in the cell corresponding to finding MUS'es in the original benchmarks using the simple prover. The value in that cell will probably be smaller than the one for the 200 variable random 3-SAT formulas as their might be more calls to the SAT solver but they are very easy in general. However, in practice the DaimlerChrysler test took much longer and was broken off before it was completely finished because of that. The overhead cost of reducing these large formulas in such small steps is very large. Although the overhead costs might not be of theoretical interest and they could be reduced by implementing the removal of a clause smarter it is in general very unwise to attempt to find a MUS in a large formula with a small core in this way.

For DUC Hunt $\boldsymbol{h}$ it also holds that it would be wise to use a core finder like AMUSE first to reduce the formula. Our program can then be used to do what it does best: MUS Proving.

9 Future implementation tasks

From the results stated in the previous chapter we conclude what future work needs to be done.

9.1 Generalize to Boolean constraints

The presented algorithm is applicable to Boolean constraints in general. It should be investigated how the software can be generalized to that as this will greatly improve the strength of the algorithm. If an ordering relation can be imposed by a Boolean constraint then we could for example cut a CNF formula in multiple satisfiable sets of clauses that each represent only one ordering relation. In that way the core finding algorithm can first determine wether a block of clauses is part of a core and after that we can continue step-by-step by reducing the size of the blocks found that are part of an unsatisfiable subset.

To generalize the implementation of the algorithm to Boolean constraints it must be investigated how a Boolean constraint can impose an ordering over the set of assignments in such a way that assignments can be compared cheaply.

9.2 Approach to backwards shift

The backwards shifts are the only computationally hard operations in the algorithm. In our current implementation those are solved using a SAT solver that is badly handicapped by a fixed direction decision order. In a future implementation we would like to try a binary search like strategy to finding the best solution.

This method works as follows: First solve the formula using a normal solver without a fixed direction decision order. If that formula is unsatisfiable we can stop and make the constraint go passive. If it is satisfiable an alternative assignment exists and it must be either the satisfying assignment found or an assignment preferred more by the constraint that is being shifted. In that case we continue to find it by calculating the assignment that is at the half way distance between the satisfying assignment just found and the old position of the constraint. A constraint stating that the new position should be larger than or equal to that assignment is added to the formula. If after a new SAT solver call that formula turns out to be satisfiable then we can do the same thing again, if it turns out to be unsatisfiable then we weaken the added "larger than" constraint to the assignment halfway in between the earlier calculated assignment and the satisfying assignment once found. This will eventually lead us to the best satisfying assignment.

This method of solving the shift constraints formula requires multiple calls to the SAT solver but as that solver exploits its decision heuristics each call will be much faster then one single call with a fixed decision order.

9.3 Stand alone implementation of repeated hot start MUS finder

In paragraph 8.2 we introduced a way to use the hot start method to find a MUS without any shifting. The results in that chapter seem very promising. As the DUC Hunt implementation was targeted at using the algorithm with shift operations a stand alone implementation of that technique might increase performance. That implementation should ideally do all iterations in a single program call in order to minimize overhead currently caused by repeated program calls and file i/o.

10 Conclusion

In this document we presented a novel algorithm for finding unsatisfiable cores in sets of Boolean constraints. It is based on a Brouwer's fixed point approximation algorithm. Besides a core finder it is also a complete SAT solver as it is guaranteed to find a satisfying assignment eventually if there is one. Furthermore it is capable of finding a guaranteed MUS. The background of our algorithm makes it quite different from other approaches to the satisfiability problem in general and finding unsatisfiable cores in particular.

The implementation of the algorithm made during this project is limited to finding unsatisfiable subsets of clauses in CNF formulas. The current performance of the software applied to finding cores is good regarding the number of cores found but searching them requires to much time.

The algorithm was extended to find guaranteed minimal unsatisfiable cores. For that application it also holds that the current implementation does work but it is too slow to be useful in practice. It is expected that generalizing the algorithm's implementation to finding subsets in sets of Boolean constraint instead of sets of clauses will make the algorithm handle larger formulas as it allows multiple constraints to be presented as one which might speed up the initial reduction of the formula. Unfortunately it is not clear yet how to implement the ordering relations that each constraint should impose on the set of assignments for general Boolean constraints. Implementing new, possibly binary search like, approaches to solving the shift constraints formula might also improve performance.

While evaluating our algorithm we found a new approach to finding MUS'es. This new approach, which does not use the shift operations, does use the entities and datastructures that our algorithm uses. It seems to perform well at proving unsatisfiable formulas to be MUS'es, at least compared to the simplest method that requires a number of satisfiability problems to be solved equal to the number of constraints in the formula. Due to its good performance a stand alone implementation of those parts of the current DUC Hunt implementation will be practically applicable as a MUS prover.

This project has been a first case study into the application of this type of algorithm to Boolean satisfiability. It has been demonstrated that it can be done and future research to make the algorithm faster has been suggested.

The final conclusion is a personal one: It has been very rewarding to approach a well studied problem in a way that was never tried before. Even if the majority of the presented results might seem quite poor my belief in the importance of investigating such novelties has grown. The implementation of the algorithm can of course be improved and extended, which might eventually result in a practically applicable core finder. However, if this research could inspire other people because it broadened the view on this well studied problem then that would definitely be an important and worthwhile result.

References

[1] Hans van Maaren, *Pivoting Algorithms Based on Boolean Vector Labeling.* Acta Mathematica Vietnamica, Volume 22, number 1, 1997, 183-198, Dedicated to Hoang Tuy on the occasion of his seventieth birthday.

[2] Hans van Maaren, *Generalized pivoting and coalitions.* In The Computation and Modelling of Economic Equilibria, A.J.J. Talman and G. van der Laan (editors), North-Holland, Amsterdam, (1987), 155-176

[3] Niklas Eén, Niklas Sörensson, *An extensible SAT solver.* In Proc. of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 03).

[4] Yoonna Oh, Maher N. Mneimneh, Zaher S. Andraus, Karem A.Sakallah, and Igor L. Markov, *AMUSE: a minimally-unsatisfiable subformula extractor.* In Proceedings of the 41st Annual Conference on Design Automation, pages 518523.ACM Press, 2004

[5] Jinbo Huang, *MUP: A Minimal Unsatisfiability Prover.* In Proc. of the Tenth Asia and South Pacific Design Automation Conference (ASP-DAC), January 2005

[6] Alex Wright, Sperner's lemma and Brouwer's fixed point theorem, In Proc. of the 2005 Canadian Undergraduate Math Conference

[7] Eric Schechter, Handbook of Analysis and its Foundations, Academic Press 1996

Appendices

A Parameter settings for DUC Hunt

Tie breaker settings

t Always prefer $true$ over $false$

f Always prefer $false$ over $true$

m For x_i prefer $false$ if $\neg x_i$ occurs more often then x_i in the input formula, $true$ otherwise.

MUS marker mode settings

mT "MUS top" Finds one core, puts marked clauses at the top of output file

mS "MUS sort" As ***mT*** but sorts unmarked clauses in opposite order

mC "MUS continue" Uses hot restart to find core in one program run

h "MUS hot start" Use hot start, must be combined with one of the previous modes.

Numerical settings

l Sets maximum backtrack depth for attempting multiple assignment labeling options.

b Sets number of times the internal solver may backtrack before involving MiniSat

B Test environment

For the purpose of testing my current projects I make use of the computational power of the PC's at the office in Delft where I am working. There are three equal Pentium IV machines on 2.93 GHz that used to be idle most of the time.

An old AMD 900 Mhz machine running Debian Linux was hooked up to the network to "handout" tests to the office machines. In fact, it is only assigned the task of running a Samba server (or in other words sharing its harddisk). On the harddisk there are three directories with test scripts, one for each "client" machine.

The three Pentium IV's normally run Windows but for this purpose they are booted from a CD-Rom. The CD-Rom contains a Slax Linux ISO which is a small Linux system specifically designed to boot from CD. I stripped most of the modules that come with Slax off and added a Samba client module. At boot time the disk on the AMD machine is mounted on the client machine which than finds its own directory with test scripts and starts executing them.

All test scripts check what they have already done and what is still left to do. In that way, the other people working at the office can reboot the machine into Windows whenever they like and whenever they stop working they help me by rebooting the machine from the CD-Rom and it will continue where it left off.

As the office network is not under a heavy load and attention is paid that no two computers use the same file the delays caused by network file i/o should remain constant on average.

C Test results

C.1 Finding cores

Mode *f*

Filename	shifts	hard shifts	shifts to 1st core	# cores	smallest core size
50_215_3_105	70323	23116	1648	88	155
50_215_3_108	74156	32132	1116	182	136
50_215_3_109	83264	37974	615	165	148
50_215_3_110	76820	31810	1287	143	149
50_215_3_111	72880	36429	674	137	143
50_215_3_112	85636	25145	1183	128	147
50_215_3_12	63140	25582	542	142	160
50_215_3_14	96742	45821	493	134	143
50_215_3_15	77982	25087	1100	32	153
50_215_3_16	68227	30411	374	135	159
50_215_3_2	90861	32755	263	232	132
50_215_3_20	85830	27344	1273	146	149
50_215_3_21	74127	29772	976	86	138
50_215_3_23	95921	41389	776	132	144
50_215_3_26	105232	25794	1746	120	159
50_215_3_27	83958	19361	600	125	157
50_215_3_28	65429	27798	1572	247	141
50_215_3_29	78065	29564	539	150	136
50_215_3_3	69577	28701	377	149	152
50_215_3_34	71344	27926	998	160	144
50_215_3_35	86461	19011	724	84	152
50_215_3_36	78268	33227	505	181	141
50_215_3_37	81305	21119	697	199	160
50_215_3_40	69750	33295	694	89	148
50_215_3_42	74239	25438	893	197	149
50_215_3_45	75896	32581	645	134	138
50_215_3_49	68140	27739	450	65	155
50_215_3_50	69108	32470	1268	92	141
50_215_3_51	71075	28648	1382	96	161
50_215_3_56	66268	28466	877	81	148
50_215_3_57	86409	38956	252	97	137
50_215_3_60	72612	32556	1425	147	151
50_215_3_64	66038	25054	2455	163	150
50_215_3_66	74737	25031	1805	163	144
50_215_3_68	76689	24671	1329	126	150
50_215_3_71	93972	33619	1286	158	154
50_215_3_72	67767	28258	1108	111	156
50_215_3_73	102304	24126	552	129	147
50_215_3_75	76479	14546	905	88	166
50_215_3_78	67039	20020	1040	59	155
50_215_3_82	90606	34218	845	123	142
50_215_3_83	65772	35346	920	162	149
50_215_3_87	99527	23438	811	57	143
50_215_3_88	90341	39604	562	65	140
50_215_3_9	86981	19051	560	173	140
50_215_3_93	67833	31843	815	78	146
50_215_3_95	66611	24135	956	126	159
50_215_3_96	72987	31674	2266	162	141
50_215_3_97	101212	20466	957	188	146
50_215_3_99	63420	34567	650	127	154
Average	78387	28942	956	131	148

Mode *f*

Filename	shifts	hard shifts	shifts to 1st core	# cores	smallest core size
60_258_3_1	75941	27385	1883	88	173
60_258_3_100	48506	22740	3375	145	192
60_258_3_13	46639	19293	1697	163	178
60_258_3_16	40543	13992	2195	101	218
60_258_3_17	41272	25133	2087	193	193
60_258_3_18	51515	23350	2692	204	173
60_258_3_21	44709	22772	903	116	192
60_258_3_24	40220	19125	1954	72	200
60_258_3_27	43823	20293	1319	156	201
60_258_3_29	57022	20181	1281	144	187
60_258_3_3	38673	17939	1359	37	193
60_258_3_30	44173	23588	1287	132	185
60_258_3_31	40947	24498	678	97	184
60_258_3_33	50687	19868	1639	152	191
60_258_3_35	49137	17349	705	160	186
60_258_3_4	45216	20459	1900	137	191
60_258_3_40	44855	22038	1750	90	184
60_258_3_45	53803	16301	652	147	198
60_258_3_46	42657	22504	1009	127	187
60_258_3_47	41870	20829	871	102	187
60_258_3_48	47456	21746	1626	95	192
60_258_3_49	41220	24722	951	251	180
60_258_3_50	42886	25274	730	115	180
60_258_3_53	42701	26262	1289	152	171
60_258_3_54	47489	24976	704	92	176
60_258_3_55	35717	23084	1287	26	187
60_258_3_56	43849	23879	928	76	190
60_258_3_57	44971	28355	1320	198	175
60_258_3_58	39680	19662	865	55	195
60_258_3_6	40144	19997	855	101	193
60_258_3_62	56120	27481	1250	168	177
60_258_3_64	53639	27682	1019	219	185
60_258_3_66	55682	24854	871	104	188
60_258_3_67	64380	16918	1427	159	177
60_258_3_69	40268	19490	1153	193	183
60_258_3_7	53900	27517	559	60	193
60_258_3_71	37656	17010	1800	65	188
60_258_3_72	72055	30118	1793	135	189
60_258_3_74	68211	17666	1615	143	188
60_258_3_76	43590	22896	1278	163	186
60_258_3_8	50603	13321	1484	83	190
60_258_3_82	43939	22896	1993	129	188
60_258_3_83	43525	26328	2144	89	171
60_258_3_87	55024	34693	414	90	166
60_258_3_88	48382	18904	2662	61	199
60_258_3_9	47258	29215	1317	140	159
60_258_3_90	36111	22977	1065	98	193
60_258_3_94	43424	19093	718	37	191
60_258_3_96	47264	19772	1579	176	181
60_258_3_98	48683	25919	1394	71	182
Average	47561	22447	1387	122	186

Mode m

Filename	shifts	hard shifts	shifts to 1st core	# cores	smallest core size
50_215_3_105	71340	31461	1817	205	135
50_215_3_108	92183	43605	276	110	132
50_215_3_109	82344	44210	562	99	146
50_215_3_110	86881	26961	1109	117	148
50_215_3_111	58018	32853	1351	194	133
50_215_3_112	61756	29842	1165	116	137
50_215_3_12	60383	25962	608	143	147
50_215_3_14	94710	51763	736	125	131
50_215_3_15	61050	26586	878	88	142
50_215_3_16	83001	36758	396	68	137
50_215_3_2	89230	39265	361	141	124
50_215_3_20	61620	26493	476	173	148
50_215_3_21	70826	38933	710	60	140
50_215_3_23	69020	31606	1201	140	145
50_215_3_26	77521	31768	957	145	140
50_215_3_27	58776	31836	472	84	147
50_215_3_28	66285	28805	1089	137	135
50_215_3_29	67096	34374	787	71	134
50_215_3_3	65646	30318	384	258	140
50_215_3_34	72429	31059	785	70	133
50_215_3_35	65645	26628	1132	144	142
50_215_3_36	95519	39232	545	165	127
50_215_3_37	85816	38421	220	111	141
50_215_3_40	64716	35871	449	176	137
50_215_3_42	75305	36781	586	182	129
50_215_3_45	105083	49349	551	159	135
50_215_3_49	72201	34026	761	165	129
50_215_3_50	55628	37911	958	144	128
50_215_3_51	109051	45256	613	157	131
50_215_3_56	67173	37699	441	139	124
50_215_3_57	86486	41768	581	103	135
50_215_3_60	68954	39768	872	110	149
50_215_3_64	74260	28402	1579	100	141
50_215_3_66	84784	27921	777	127	136
50_215_3_68	72087	27591	864	90	144
50_215_3_71	87605	31426	811	131	148
50_215_3_72	102715	34246	280	164	139
50_215_3_73	70360	23621	1564	183	147
50_215_3_75	64138	27212	695	158	139
50_215_3_78	70866	21973	976	129	142
50_215_3_82	70055	30684	983	134	128
50_215_3_83	64177	38617	999	243	129
50_215_3_87	90493	23267	833	82	137
50_215_3_88	54058	36194	1729	121	125
50_215_3_9	71994	24364	633	70	138
50_215_3_93	72204	35043	582	161	133
50_215_3_95	68060	29081	648	102	145
50_215_3_96	81930	41861	1403	147	130
50_215_3_97	66616	24796	1559	111	142
50_215_3_99	73228	46225	786	76	142
Average	74826	33794	831	133	137

Mode *m*

Filename	shifts	hard shifts	shifts to 1st core	# cores	smallest core size
60_258_3_1	54454	26046	584	143	172
60_258_3_100	32107	17613	1773	88	184
60_258_3_13	34287	19692	908	119	170
60_258_3_16	37091	21427	2334	181	178
60_258_3_17	40727	27176	2262	151	168
60_258_3_18	40270	20761	1052	47	168
60_258_3_21	37428	20446	1613	150	182
60_258_3_24	27840	19175	1946	94	178
60_258_3_27	33859	17498	2040	146	182
60_258_3_29	42932	21138	1192	85	180
60_258_3_3	37018	20065	1891	198	177
60_258_3_30	34796	16604	1459	83	178
60_258_3_31	37023	22965	607	145	163
60_258_3_33	39292	21890	2385	106	166
60_258_3_35	36829	16465	843	172	178
60_258_3_4	36702	16157	1596	108	177
60_258_3_40	54038	23445	787	139	183
60_258_3_45	35076	15807	961	35	181
60_258_3_46	30839	18656	1576	192	182
60_258_3_47	35535	20072	413	149	169
60_258_3_48	62037	38722	1415	144	159
60_258_3_49	34103	20827	874	279	170
60_258_3_50	35626	22541	954	176	156
60_258_3_53	47013	25816	1674	175	169
60_258_3_54	47833	22221	800	77	166
60_258_3_55	45174	25664	1117	122	169
60_258_3_56	38251	25741	1485	125	163
60_258_3_57	42521	27875	687	100	161
60_258_3_58	35957	24879	372	122	159
60_258_3_6	37969	23998	1099	114	166
60_258_3_62	49441	25534	1268	106	167
60_258_3_64	59762	28725	381	202	176
60_258_3_66	67429	30273	647	114	162
60_258_3_67	39032	17980	2490	90	155
60_258_3_69	41020	24833	945	157	161
60_258_3_7	34563	20549	2150	133	175
60_258_3_71	32322	15234	2195	110	181
60_258_3_72	46228	23623	2034	107	172
60_258_3_74	50684	24588	1920	133	172
60_258_3_76	30835	19677	1526	140	161
60_258_3_8	31657	14516	1671	177	184
60_258_3_82	40127	23684	1604	105	174
60_258_3_83	43519	24157	1005	124	173
60_258_3_87	47948	30176	372	187	160
60_258_3_88	42187	22869	1227	90	183
60_258_3_9	37466	19235	2683	183	159
60_258_3_90	55705	36956	530	150	163
60_258_3_94	32837	21127	698	50	183
60_258_3_96	40784	22194	1185	107	166
60_258_3_98	41184	22166	896	103	169
Average	40987	22590	1323	131	171

Mode *t*

Filename	shifts	hard shifts	shifts to 1st core	# cores	smallest core size
50_215_3_105	106620	35502	874	86	143
50_215_3_108	85626	35443	301	170	141
50_215_3_109	55396	20421	1028	145	162
50_215_3_110	72006	24332	755	312	149
50_215_3_111	61036	20357	1540	93	154
50_215_3_112	74693	20736	1300	91	151
50_215_3_12	57289	24468	635	175	154
50_215_3_14	77236	26134	1369	105	156
50_215_3_15	71641	23906	428	149	160
50_215_3_16	73742	25707	582	186	148
50_215_3_2	67371	22256	1103	140	147
50_215_3_20	67184	19066	345	84	158
50_215_3_21	84330	19945	519	83	156
50_215_3_23	54955	26194	1678	125	168
50_215_3_26	71631	33359	1082	92	152
50_215_3_27	57664	32747	915	99	149
50_215_3_28	63057	29441	869	46	158
50_215_3_29	62229	23398	1187	33	149
50_215_3_3	75080	19171	613	121	150
50_215_3_34	66254	23689	657	107	146
50_215_3_35	56160	18371	1198	133	163
50_215_3_36	66087	29313	1291	163	138
50_215_3_37	82019	21484	623	113	148
50_215_3_40	63409	21511	1054	175	152
50_215_3_42	91446	36134	263	180	139
50_215_3_45	85334	33775	629	97	153
50_215_3_49	71719	27326	908	190	141
50_215_3_50	67645	33165	1253	121	141
50_215_3_51	81062	44438	303	44	140
50_215_3_56	65185	21788	631	86	153
50_215_3_57	70935	30536	1411	119	138
50_215_3_60	72652	35491	614	148	153
50_215_3_64	94248	27099	1202	126	150
50_215_3_66	71342	30771	461	106	144
50_215_3_68	85329	21779	952	131	148
50_215_3_71	52839	26651	1355	126	151
50_215_3_72	83432	25751	572	182	150
50_215_3_73	63014	20262	1724	122	159
50_215_3_75	67608	24404	651	101	150
50_215_3_78	66499	20316	703	68	148
50_215_3_82	57046	19528	863	80	154
50_215_3_83	56644	31305	1253	110	142
50_215_3_87	69590	17877	1283	270	149
50_215_3_88	68491	41504	1214	110	129
50_215_3_9	79585	19433	731	44	151
50_215_3_93	56981	29861	524	244	145
50_215_3_95	65701	28625	652	83	144
50_215_3_96	104345	31083	427	115	157
50_215_3_97	56222	17170	495	117	155
50_215_3_99	64541	37028	511	66	149
Average	70843	26601	871	124	150

Mode *t*

Filename	shifts	hard shifts	shifts to 1st core	# cores	smallest core size
60_258_3_1	57436	24174	773	214	171
60_258_3_100	35602	21711	1799	83	191
60_258_3_13	35411	15139	945	156	179
60_258_3_16	31231	15869	2247	108	186
60_258_3_17	29789	17512	1412	139	187
60_258_3_18	32136	16377	1091	105	192
60_258_3_21	29220	14388	2067	71	192
60_258_3_24	30135	14246	1288	67	200
60_258_3_27	41048	21966	1826	159	183
60_258_3_29	36992	15463	1267	60	192
60_258_3_3	30040	15247	996	24	197
60_258_3_30	33945	20561	809	78	187
60_258_3_31	32377	20284	1115	149	175
60_258_3_33	31796	15712	2088	170	177
60_258_3_35	39988	14893	937	227	194
60_258_3_4	32847	16913	1318	129	186
60_258_3_40	45679	17035	1132	67	196
60_258_3_45	35986	14834	1280	110	184
60_258_3_46	36188	19756	1680	135	186
60_258_3_47	30986	16418	370	46	193
60_258_3_48	55949	31883	1240	157	175
60_258_3_49	28431	14193	1174	82	189
60_258_3_50	38368	19084	1235	187	168
60_258_3_53	33873	15245	1285	51	196
60_258_3_54	27238	16028	1606	119	187
60_258_3_55	44207	28750	969	136	166
60_258_3_56	47208	21586	1198	71	180
60_258_3_57	59249	27800	913	133	176
60_258_3_58	39578	20177	632	31	191
60_258_3_6	46189	24682	499	92	179
60_258_3_62	31352	19540	1601	159	180
60_258_3_64	33384	13051	2508	124	193
60_258_3_66	47957	22158	1874	207	173
60_258_3_67	35946	17286	2405	117	176
60_258_3_69	31206	16749	1747	80	185
60_258_3_7	34026	17138	1977	94	198
60_258_3_71	39604	19320	1175	100	185
60_258_3_72	46269	27630	1207	125	180
60_258_3_74	56773	21589	1106	167	187
60_258_3_76	30032	18381	1098	86	169
60_258_3_8	34254	13414	1108	72	190
60_258_3_82	36603	20103	1224	149	174
60_258_3_83	44609	21052	1459	95	182
60_258_3_87	37313	19092	2904	59	174
60_258_3_88	40616	19059	793	113	198
60_258_3_9	53436	16173	1411	72	182
60_258_3_90	43097	21863	1373	83	195
60_258_3_94	29075	16362	1044	130	195
60_258_3_96	41235	17431	1162	162	170
60_258_3_98	45498	19878	1099	99	181
Average	38428	18904	1349	113	184

C.2 MUS test

Mode *mT*

Filename	shifts	hard shifts	calls	mus size	CPU time
50_215_3_105	8090	3724	16	114	68.92
50_215_3_108	1763	695	9	102	8.78
50_215_3_109	7059	3159	14	114	57.34
50_215_3_110	11947	5337	15	118	98.03
50_215_3_111	6270	2408	9	122	53.74
50_215_3_112	5709	2629	10	119	53.04
50_215_3_12	7446	2732	13	130	70.14
50_215_3_14	15546	7971	13	106	135.1
50_215_3_15	4039	1715	12	116	33.35
50_215_3_16	9961	4287	16	120	86.19
50_215_3_2	3647	1649	11	111	25.44
50_215_3_20	4772	1887	13	108	32.65
50_215_3_21	3939	1697	18	112	28
50_215_3_23	7785	3255	14	117	65.99
50_215_3_26	6382	2281	14	115	39.31
50_215_3_27	12526	5588	14	128	117.59
50_215_3_28	4106	1645	10	109	33.44
50_215_3_29	6122	2617	17	114	46.33
50_215_3_3	7139	2702	15	121	51.01
50_215_3_34	4318	1649	14	108	25.98
50_215_3_35	11202	3656	18	124	98.03
50_215_3_36	5888	3105	14	103	45.32
50_215_3_37	2711	897	10	110	13.63
50_215_3_40	7957	3537	14	109	54.8
50_215_3_42	3419	1577	11	113	23.51
50_215_3_45	7708	2716	13	112	54.43
50_215_3_49	2798	1207	10	107	21.69
50_215_3_50	7458	3771	14	104	58.53
50_215_3_51	2670	1362	11	101	17.94
50_215_3_56	15820	6291	16	109	126.43
50_215_3_57	4557	1798	13	106	31.04
50_215_3_60	7948	3766	15	123	74.47
50_215_3_64	6904	3232	14	108	60.77
50_215_3_66	7132	3300	11	104	51.79
50_215_3_68	10552	3437	14	109	73.82
50_215_3_71	11618	3265	20	127	85.48
50_215_3_72	4985	1943	10	121	37.46
50_215_3_73	9782	3573	13	113	75.97
50_215_3_75	8284	3746	15	115	72.16
50_215_3_78	7744	1980	17	125	57.09
50_215_3_82	9688	3835	12	111	74.39
50_215_3_83	7092	2786	12	106	49.15
50_215_3_87	4596	1423	11	101	31.33
50_215_3_88	7191	4084	13	105	63.79
50_215_3_9	5805	2025	12	113	44.72
50_215_3_93	8043	3009	14	111	55.14
50_215_3_95	7802	2564	13	129	63.14
50_215_3_96	4357	1846	13	111	33.43
50_215_3_97	9479	3170	12	110	65.94
50_215_3_99	5603	2373	12	111	42.05
Average	7107	2898	13	113	55.8

Mode *mT*

Filename	shifts	hard shifts	calls	mus size	CPU time
MUS found in 50_215_3_105	6927	3564	13	114	56.34
MUS found in 50_215_3_108	1609	713	11	102	7.38
MUS found in 50_215_3_109	4305	1810	13	114	29.73
MUS found in 50_215_3_110	5114	1783	15	118	32.66
MUS found in 50_215_3_111	1973	658	10	122	10.99
MUS found in 50_215_3_112	2389	1037	10	117	16.53
MUS found in 50_215_3_12	6812	2363	13	130	56.97
MUS found in 50_215_3_14	5121	1929	12	103	33.27
MUS found in 50_215_3_15	3303	1648	8	115	25.56
MUS found in 50_215_3_16	5468	2245	13	119	38.67
MUS found in 50_215_3_2	3909	1448	10	112	23.23
MUS found in 50_215_3_20	7632	3495	13	110	57.32
MUS found in 50_215_3_21	5556	2369	14	112	42.83
MUS found in 50_215_3_23	4312	1697	14	117	29.55
MUS found in 50_215_3_26	8993	3330	14	116	65.3
MUS found in 50_215_3_27	9971	3889	14	128	84.17
MUS found in 50_215_3_28	2697	1243	12	109	18.68
MUS found in 50_215_3_29	4919	2411	12	110	38.26
MUS found in 50_215_3_3	3275	1368	12	120	22.02
MUS found in 50_215_3_34	2972	1075	13	107	15.44
MUS found in 50_215_3_35	5038	1741	17	125	33.02
MUS found in 50_215_3_36	1990	1010	10	104	11.35
MUS found in 50_215_3_37	3513	1416	12	110	23.12
MUS found in 50_215_3_40	6478	3083	14	111	42.45
MUS found in 50_215_3_42	3020	1184	11	113	21.12
MUS found in 50_215_3_45	3039	1348	13	120	21.73
MUS found in 50_215_3_49	3896	1375	14	107	20.95
MUS found in 50_215_3_50	8512	3841	14	102	55.13
MUS found in 50_215_3_51	2758	1474	11	100	15.73
MUS found in 50_215_3_56	6939	2651	11	109	45.23
MUS found in 50_215_3_57	2798	1312	14	106	16.69
MUS found in 50_215_3_60	4600	2500	15	123	35.24
MUS found in 50_215_3_64	3553	1548	14	112	26.38
MUS found in 50_215_3_66	3887	1521	9	107	20.62
MUS found in 50_215_3_68	3457	1357	11	109	21.25
MUS found in 50_215_3_71	4573	1357	16	125	28.86
MUS found in 50_215_3_72	3690	1493	12	121	24.89
MUS found in 50_215_3_73	4220	1397	13	119	26.31
MUS found in 50_215_3_75	7521	3065	10	115	59.17
MUS found in 50_215_3_78	4850	1590	10	124	30.41
MUS found in 50_215_3_82	10506	4613	13	107	79.94
MUS found in 50_215_3_83	3794	1353	12	106	21.87
MUS found in 50_215_3_87	3489	1269	10	106	21.16
MUS found in 50_215_3_88	3891	1770	13	105	24.5
MUS found in 50_215_3_9	3861	1272	12	114	24.44
MUS found in 50_215_3_93	4016	1425	15	111	22.07
MUS found in 50_215_3_95	7031	2309	12	128	54.49
MUS found in 50_215_3_96	5614	2011	14	110	29.66
MUS found in 50_215_3_97	2344	945	10	109	14.22
MUS found in 50_215_3_99	6333	2396	15	113	44.41
Average	4729	1914	12	113	32.4

Mode *mT*

Filename	shifts	hard shifts	calls	mus size	CPU time
60_258_3_1	5958	2486	12	138	62.16
60_258_3_100	10348	5673	17	143	149.26
60_258_3_13	8728	3371	18	137	98.46
60_258_3_16	22741	10447	18	153	357.81
60_258_3_17	10577	4470	17	149	128.1
60_258_3_18	8003	3265	16	141	100.06
60_258_3_21	14478	7377	17	149	240.4
60_258_3_24	16550	9389	15	151	317.18
60_258_3_27	14750	6932	20	163	235.97
60_258_3_29	20572	8815	17	146	278.38
60_258_3_3	17649	8471	18	152	278.6
60_258_3_30	8431	3929	15	148	115.3
60_258_3_31	6614	3408	15	134	77.16
60_258_3_33	14809	6303	18	148	177.87
60_258_3_35	23757	9333	19	149	325.66
60_258_3_4	19088	9326	18	156	296.8
60_258_3_40	9238	3436	19	155	121.29
60_258_3_45	17949	8172	18	158	275.99
60_258_3_46	20862	13417	15	149	406.5
60_258_3_47	10144	4849	15	148	140.66
60_258_3_48	26224	16291	16	142	445.2
60_258_3_49	13729	5903	20	153	164.66
60_258_3_50	11312	4817	16	127	140.1
60_258_3_53	8869	4983	15	141	134.81
60_258_3_54	17834	10089	19	134	283.54
60_258_3_55	9749	5636	13	133	133.56
60_258_3_56	12109	5998	17	140	158.99
60_258_3_57	15497	6829	14	129	191.13
60_258_3_58	4405	2096	12	137	51.56
60_258_3_6	12334	6288	18	135	154.35
60_258_3_62	10320	5066	19	133	127.83
60_258_3_64	18260	10490	15	145	317.98
60_258_3_66	7948	4848	14	126	109.17
60_258_3_67	9944	4397	18	124	130.32
60_258_3_69	13378	6371	17	141	178.19
60_258_3_7	35147	23108	17	152	716.49
60_258_3_71	14884	6381	18	157	229.7
60_258_3_72	18844	11054	16	152	324.94
60_258_3_74	10428	4693	16	147	136.16
60_258_3_76	13497	5883	20	133	155.35
60_258_3_8	11787	4957	18	156	160.16
60_258_3_82	17997	8583	21	143	236.72
60_258_3_83	12116	5849	18	128	147.43
60_258_3_87	36639	17641	18	142	521.64
60_258_3_88	14580	5266	17	139	167.87
60_258_3_9	5591	2927	18	127	71.4
60_258_3_90	12849	5626	19	140	128.08
60_258_3_94	15416	6347	20	153	225.47
60_258_3_96	8141	3102	16	133	93.91
60_258_3_98	9755	4167	15	142	114.01
Average	14217	6971	17	143	206.7

Mode *mT*

Filename	shifts	hard shifts	calls	mus size	CPU time
MUS found in 60_258_3_1	6171	2572	12	133	58.15
MUS found in 60_258_3_100	11046	5300	16	146	129.35
MUS found in 60_258_3_13	6711	3173	18	136	66.42
MUS found in 60_258_3_16	12363	5789	18	154	159.42
MUS found in 60_258_3_17	8677	4178	18	150	101.25
MUS found in 60_258_3_18	3389	1169	14	135	29.24
MUS found in 60_258_3_21	12697	6141	16	150	184.41
MUS found in 60_258_3_24	11092	6746	19	148	166.02
MUS found in 60_258_3_27	13827	6555	18	163	214.52
MUS found in 60_258_3_29	8655	3438	13	145	101.88
MUS found in 60_258_3_3	11004	7110	18	151	176.72
MUS found in 60_258_3_30	11212	6583	16	148	151.12
MUS found in 60_258_3_31	6881	3071	14	133	66.14
MUS found in 60_258_3_33	7084	3093	15	140	79.57
MUS found in 60_258_3_35	11509	4623	16	146	148.86
MUS found in 60_258_3_4	10066	4543	16	154	133.05
MUS found in 60_258_3_40	12369	4466	16	157	153.29
MUS found in 60_258_3_45	9635	4129	17	155	119.6
MUS found in 60_258_3_46	6435	3324	14	150	85.34
MUS found in 60_258_3_47	11837	5932	15	147	158.74
MUS found in 60_258_3_48	7713	3002	17	140	81.01
MUS found in 60_258_3_49	17602	8863	16	153	217.68
MUS found in 60_258_3_50	10782	4254	17	128	98.3
MUS found in 60_258_3_53	5218	2911	19	137	64.61
MUS found in 60_258_3_54	8202	3829	16	133	90.31
MUS found in 60_258_3_55	3805	1874	14	131	35.68
MUS found in 60_258_3_56	7310	4004	16	142	89.67
MUS found in 60_258_3_57	5388	2819	13	129	57.34
MUS found in 60_258_3_58	10787	5090	15	138	121.89
MUS found in 60_258_3_6	5715	2744	18	132	63.35
MUS found in 60_258_3_62	8429	4062	13	133	92.19
MUS found in 60_258_3_64	7839	3980	18	140	96.91
MUS found in 60_258_3_66	3405	1777	14	125	34.95
MUS found in 60_258_3_67	10386	4055	21	127	107.52
MUS found in 60_258_3_69	11572	5262	15	141	132.19
MUS found in 60_258_3_7	10244	5608	15	153	151.89
MUS found in 60_258_3_71	13479	5691	17	156	174.43
MUS found in 60_258_3_72	9144	4526	18	150	118.53
MUS found in 60_258_3_74	11540	5060	17	146	151.05
MUS found in 60_258_3_76	7027	2941	20	130	73.45
MUS found in 60_258_3_8	9463	3673	18	157	104.81
MUS found in 60_258_3_82	6122	2688	16	140	60.99
MUS found in 60_258_3_83	4825	2427	15	126	46.23
MUS found in 60_258_3_87	7356	3386	16	146	78.99
MUS found in 60_258_3_88	7025	3270	15	142	93.32
MUS found in 60_258_3_9	5255	2215	14	127	50.59
MUS found in 60_258_3_90	5011	2390	13	135	48.9
MUS found in 60_258_3_94	9996	4235	15	151	117.69
MUS found in 60_258_3_96	11937	4479	17	133	124.29
MUS found in 60_258_3_98	10073	4483	17	139	112.96
Average	8906	4151	16	142	107.5

Mode *mS*

Filename	shifts	minisat shifts	calls	MUS size	CPU time
50_215_3_105	9115	4359	18	117	83.21
50_215_3_108	2713	1226	11	100	15.02
50_215_3_109	4318	1893	14	117	31.98
50_215_3_110	9474	3919	16	124	78.55
50_215_3_111	7608	2653	13	120	60.76
50_215_3_112	6794	2892	14	114	52.94
50_215_3_12	4777	1647	11	128	42.28
50_215_3_14	13081	6557	13	102	110.9
50_215_3_15	4045	1857	12	116	35.41
50_215_3_16	6629	2664	15	119	56.32
50_215_3_2	4049	1550	10	115	28.92
50_215_3_20	5844	2279	15	112	42.86
50_215_3_21	3694	1552	15	111	26.84
50_215_3_23	7914	3469	13	112	64.94
50_215_3_26	5273	1914	13	118	32.81
50_215_3_27	18154	7809	21	129	166.72
50_215_3_28	7172	3048	13	109	56.15
50_215_3_29	4589	2179	17	108	33.66
50_215_3_3	9316	3818	16	123	76.01
50_215_3_34	6579	2809	15	112	45.1
50_215_3_35	10352	3267	18	125	86.18
50_215_3_36	7634	3511	14	105	58.8
50_215_3_37	7560	3115	15	112	48.57
50_215_3_40	6533	2982	14	110	44.44
50_215_3_42	3084	1481	11	115	20.58
50_215_3_45	12916	5748	17	117	111.41
50_215_3_49	4942	2096	15	107	34.53
50_215_3_50	8353	4098	17	105	59.39
50_215_3_51	4523	2324	14	100	32.59
50_215_3_56	11945	4335	18	110	81.33
50_215_3_57	5229	2104	12	105	36.84
50_215_3_60	12015	5472	16	123	118.6
50_215_3_64	7468	3255	14	108	60.61
50_215_3_66	11298	4989	14	110	87.54
50_215_3_68	9854	2379	14	115	67.37
50_215_3_71	11643	3358	17	128	85.42
50_215_3_72	6875	3541	12	121	62.5
50_215_3_73	12403	4175	19	125	97.43
50_215_3_75	8848	4458	15	116	78.87
50_215_3_78	9752	3099	16	123	77.04
50_215_3_82	12524	5121	16	108	96.17
50_215_3_83	7119	2980	14	106	48.67
50_215_3_87	5063	1447	14	106	31.97
50_215_3_88	7313	4041	11	104	64.54
50_215_3_9	4832	1670	12	114	36.84
50_215_3_93	11831	4086	13	111	84.53
50_215_3_95	6872	2621	16	126	55.02
50_215_3_96	5141	2328	14	112	38.3
50_215_3_97	11077	4093	10	110	87.44
50_215_3_99	5827	2507	15	108	38.52
Average	7839	3216	14	114	61.5

Mode *mS*

Filename	shifts	minisat shifts	calls	MUS size	CPU time
MUS found in 50_215_3_105	6529	3224	14	114	51.64
MUS found in 50_215_3_108	1659	752	11	102	7.8
MUS found in 50_215_3_109	4083	1801	12	114	28.68
MUS found in 50_215_3_110	5365	1879	16	118	34.95
MUS found in 50_215_3_111	2195	790	12	122	12.4
MUS found in 50_215_3_112	2972	1230	13	117	19.86
MUS found in 50_215_3_12	6742	2225	12	130	54.26
MUS found in 50_215_3_14	5030	1811	12	103	31.97
MUS found in 50_215_3_15	4765	2385	11	115	37.22
MUS found in 50_215_3_16	7346	2967	16	119	52.53
MUS found in 50_215_3_2	4747	1712	12	112	27.87
MUS found in 50_215_3_20	7794	3424	13	110	58.82
MUS found in 50_215_3_21	4697	1888	15	112	34.54
MUS found in 50_215_3_23	4510	1767	14	117	31.34
MUS found in 50_215_3_26	8909	3176	15	116	65.36
MUS found in 50_215_3_27	9620	3536	14	128	79.59
MUS found in 50_215_3_28	3228	1508	14	109	22.66
MUS found in 50_215_3_29	5589	2786	14	110	43.34
MUS found in 50_215_3_3	3234	1247	14	120	20.35
MUS found in 50_215_3_34	3241	1277	13	107	17.49
MUS found in 50_215_3_35	5107	1692	17	125	33.01
MUS found in 50_215_3_36	2476	1260	12	104	14.73
MUS found in 50_215_3_37	4250	1710	14	110	28.06
MUS found in 50_215_3_40	6543	2836	13	111	43.46
MUS found in 50_215_3_42	3257	1335	13	113	22.98
MUS found in 50_215_3_45	3270	1500	13	120	23.99
MUS found in 50_215_3_49	3512	1288	14	107	19.2
MUS found in 50_215_3_50	7760	3403	13	102	50.19
MUS found in 50_215_3_51	2890	1465	11	100	16.07
MUS found in 50_215_3_56	7869	3084	13	109	51.67
MUS found in 50_215_3_57	3053	1398	15	106	17.93
MUS found in 50_215_3_60	5164	2780	17	123	39.81
MUS found in 50_215_3_64	3910	1680	15	112	29.38
MUS found in 50_215_3_66	4695	1667	11	107	24.82
MUS found in 50_215_3_68	4051	1642	12	109	25.06
MUS found in 50_215_3_71	4419	1319	17	125	27.1
MUS found in 50_215_3_72	4497	1779	14	121	30.35
MUS found in 50_215_3_73	4727	1572	14	119	28.81
MUS found in 50_215_3_75	9563	3823	13	115	73.76
MUS found in 50_215_3_78	5933	1873	12	124	37.64
MUS found in 50_215_3_82	11691	5400	15	107	91.08
MUS found in 50_215_3_83	3771	1347	12	106	21.53
MUS found in 50_215_3_87	3355	1202	11	106	20.04
MUS found in 50_215_3_88	4490	2089	15	105	28.11
MUS found in 50_215_3_9	5391	1549	15	114	32.81
MUS found in 50_215_3_93	4207	1512	15	111	23.6
MUS found in 50_215_3_95	8829	2896	14	128	70.04
MUS found in 50_215_3_96	5201	1882	13	110	27.88
MUS found in 50_215_3_97	2591	1007	12	109	16.1
MUS found in 50_215_3_99	5237	2221	13	113	37.76
Average	5079	2032	14	113	34.8

Mode *mS*

Filename	shifts	minisat shifts	calls	MUS size	CPU time
60_258_3_1	8889	3591	13	132	95.66
60_258_3_100	16044	7293	20	149	212.9
60_258_3_13	11197	4069	17	140	126.8
60_258_3_16	19580	8375	21	152	293.74
60_258_3_17	9751	4630	18	149	126.07
60_258_3_18	10651	4535	17	136	127.73
60_258_3_21	16666	9319	20	153	292.96
60_258_3_24	23825	13867	19	147	469.46
60_258_3_27	23391	10213	15	160	381.76
60_258_3_29	7693	3416	20	143	90.51
60_258_3_3	16850	7093	20	156	274.06
60_258_3_30	10283	5162	15	148	159.42
60_258_3_31	8511	3951	15	130	92.21
60_258_3_33	15111	6977	18	138	189.2
60_258_3_35	20150	7741	16	148	274.26
60_258_3_4	16791	7693	22	152	259.38
60_258_3_40	11035	3988	19	156	148.13
60_258_3_45	26735	14129	18	159	504.74
60_258_3_46	15250	7546	18	153	255.02
60_258_3_47	6399	3324	14	145	88.26
60_258_3_48	27467	16589	19	142	449.05
60_258_3_49	13449	5796	20	152	157.99
60_258_3_50	11386	4774	16	130	138.91
60_258_3_53	7347	4497	13	135	117.39
60_258_3_54	17608	10602	19	140	294.77
60_258_3_55	11533	6990	15	128	160.7
60_258_3_56	15101	7474	17	140	208.24
60_258_3_57	15182	6454	15	130	188.67
60_258_3_58	6550	3268	16	137	76.89
60_258_3_6	5375	2811	17	131	57.25
60_258_3_62	6854	3587	20	129	81.45
60_258_3_64	22964	12982	15	150	415.24
60_258_3_66	16868	10309	20	127	236.49
60_258_3_67	7430	3384	14	123	98.24
60_258_3_69	12161	5608	16	142	162.5
60_258_3_7	43676	27988	19	152	872.75
60_258_3_71	16312	7010	17	155	242.12
60_258_3_72	11422	5476	25	153	149.21
60_258_3_74	21492	12525	17	145	344.68
60_258_3_76	10864	4918	17	136	127.87
60_258_3_8	12036	5336	18	150	174.96
60_258_3_82	32587	17770	20	143	494.38
60_258_3_83	9494	4462	20	126	112.64
60_258_3_87	25304	12183	15	144	354.67
60_258_3_88	14425	5576	21	140	165.57
60_258_3_9	5057	2436	15	127	66.22
60_258_3_90	11945	5303	17	141	127.45
60_258_3_94	16733	5932	19	149	231.97
60_258_3_96	15761	5918	17	134	195.99
60_258_3_98	11955	4646	17	138	125.1
Average	15023	7390	18	142	221.8

Mode *mS*

Filename	shifts	minisat shifts	calls	MUS size	CPU time
MUS found in 60_258_3_1	6895	2972	14	133	65.87
MUS found in 60_258_3_100	10778	5076	16	146	125.2
MUS found in 60_258_3_13	5746	2582	16	136	55.62
MUS found in 60_258_3_16	13445	6539	19	154	177.43
MUS found in 60_258_3_17	9332	4621	19	150	111.17
MUS found in 60_258_3_18	3341	1122	15	135	28.17
MUS found in 60_258_3_21	13286	6280	17	150	190.25
MUS found in 60_258_3_24	12257	7350	22	148	179.09
MUS found in 60_258_3_27	13193	6289	19	163	201.49
MUS found in 60_258_3_29	7838	3124	13	145	90.87
MUS found in 60_258_3_3	12129	7884	19	151	196.89
MUS found in 60_258_3_30	10376	6120	16	148	139.7
MUS found in 60_258_3_31	9067	3998	18	133	87.24
MUS found in 60_258_3_33	8394	3814	15	140	93.6
MUS found in 60_258_3_35	12608	4843	17	146	158.99
MUS found in 60_258_3_4	11225	5051	17	154	149.1
MUS found in 60_258_3_40	15095	5803	17	157	195.67
MUS found in 60_258_3_45	8772	3952	17	155	110.42
MUS found in 60_258_3_46	6480	3249	15	150	84.09
MUS found in 60_258_3_47	9389	4736	13	147	124.95
MUS found in 60_258_3_48	8092	3073	19	140	83.17
MUS found in 60_258_3_49	18753	10089	18	153	239.09
MUS found in 60_258_3_50	11428	4310	17	128	103.03
MUS found in 60_258_3_53	5204	2893	19	137	64.37
MUS found in 60_258_3_54	8169	3774	17	133	88.34
MUS found in 60_258_3_55	3587	1560	13	131	32.65
MUS found in 60_258_3_56	7399	4235	17	142	91.45
MUS found in 60_258_3_57	6898	3709	16	129	73.86
MUS found in 60_258_3_58	10383	4814	15	138	117.81
MUS found in 60_258_3_6	5632	2788	18	132	62.49
MUS found in 60_258_3_62	9510	4536	16	133	101.57
MUS found in 60_258_3_64	7868	4006	19	140	96.82
MUS found in 60_258_3_66	3358	1795	14	125	34.81
MUS found in 60_258_3_67	9372	3730	21	127	99.53
MUS found in 60_258_3_69	11382	4982	16	141	129.67
MUS found in 60_258_3_7	9611	5069	13	153	143.9
MUS found in 60_258_3_71	16634	6694	20	156	215.23
MUS found in 60_258_3_72	9601	4537	19	150	123.9
MUS found in 60_258_3_74	12935	5215	19	146	168.18
MUS found in 60_258_3_76	7648	3427	20	130	82.53
MUS found in 60_258_3_8	9142	3425	19	157	99.29
MUS found in 60_258_3_82	6900	3012	19	140	68.15
MUS found in 60_258_3_83	4984	2396	16	126	46.75
MUS found in 60_258_3_87	8443	4030	19	146	91
MUS found in 60_258_3_88	7253	3445	16	142	96.49
MUS found in 60_258_3_9	6093	2479	18	127	57.39
MUS found in 60_258_3_90	5424	2681	16	135	53.96
MUS found in 60_258_3_94	12376	5060	17	151	144.71
MUS found in 60_258_3_96	13420	5283	18	133	140.75
MUS found in 60_258_3_98	9851	4661	16	139	110.57
Average	9340	4342	17	142	112.5

Mode *mC*

Filename	shifts	minisat shifts	iterations	MUS size	CPU time
50_215_3_105	93771	42709	84	115	856.19
50_215_3_108	62275	26014	69	100	514.60
50_215_3_109	105517	47827	83	114	976.33
50_215_3_110	111498	39900	80	119	1003.97
50_215_3_111	110634	44409	80	121	1045.08
50_215_3_112	89817	38876	71	117	830.09
50_215_3_12	117417	40596	76	128	1066.97
50_215_3_14	91639	37024	81	104	732.22
50_215_3_15	101588	39421	84	116	919.28
50_215_3_16	86131	36435	75	125	784.39
50_215_3_2	79853	30791	67	111	647.35
50_215_3_20	101944	46918	88	108	938.17
50_215_3_21	86308	33655	72	113	702.04
50_215_3_23	88760	34597	77	115	771.29
50_215_3_26	98619	35143	74	117	844.55
50_215_3_27	132255	61720	83	129	1409.86
50_215_3_28	86835	34676	72	109	752.75
50_215_3_29	82456	32523	86	110	678.34
50_215_3_3	97787	42403	89	121	881.66
50_215_3_34	84733	33334	80	109	712.18
50_215_3_35	130404	47069	84	126	1319.78
50_215_3_36	87445	34494	75	107	674.91
50_215_3_37	98333	33009	86	111	794.63
50_215_3_40	94495	39160	84	113	798.81
50_215_3_42	90767	40817	72	114	832.90
50_215_3_45	81037	31831	77	109	620.81
50_215_3_49	76777	29478	78	107	605.45
50_215_3_50	79165	40094	77	107	700.46
50_215_3_51	68783	25496	70	102	462.67
50_215_3_56	78067	30171	69	109	626.16
50_215_3_57	81673	30266	80	107	627.80
50_215_3_60	125373	49424	96	123	1149.60
50_215_3_64	111639	43195	87	113	948.54
50_215_3_66	85536	31000	83	104	647.79
50_215_3_68	91579	33691	73	110	744.85
50_215_3_71	128510	46570	82	126	1211.57
50_215_3_72	107967	40660	81	123	959.40
50_215_3_73	102628	37400	82	116	929.57
50_215_3_75	106957	40589	83	117	928.14
50_215_3_78	121731	42334	90	122	1067.60
50_215_3_82	80638	27461	68	111	648.59
50_215_3_83	73807	26461	77	106	563.01
50_215_3_87	98962	32629	80	101	772.70
50_215_3_88	79454	40081	75	103	681.49
50_215_3_9	94919	33722	75	113	785.23
50_215_3_93	100783	39285	75	112	822.40
50_215_3_95	86466	28509	78	128	742.63
50_215_3_96	103801	36495	86	110	825.96
50_215_3_97	97563	35056	76	109	805.99
50_215_3_99	108565	46320	85	111	901.32
Average	95673	37435	79	113	825.4

Mode *mC*

Filename	shifts	minisat shifts	iterations	MUS size	CPU time
MUS found in 50_215_3_105	62115	27197	54	114	515.48
MUS found in 50_215_3_108	35199	14435	45	102	227.29
MUS found in 50_215_3_109	60688	27400	49	114	490.64
MUS found in 50_215_3_110	73604	25698	59	118	551.50
MUS found in 50_215_3_111	75881	28060	60	122	585.94
MUS found in 50_215_3_112	50038	21079	51	117	395.33
MUS found in 50_215_3_12	89627	34339	59	130	795.40
MUS found in 50_215_3_14	41361	16897	48	103	288.13
MUS found in 50_215_3_15	44116	16111	53	115	308.15
MUS found in 50_215_3_16	64998	28596	58	119	535.99
MUS found in 50_215_3_2	42806	16052	47	112	287.39
MUS found in 50_215_3_20	63321	27216	59	110	493.01
MUS found in 50_215_3_21	71896	28701	59	112	558.91
MUS found in 50_215_3_23	56962	21455	62	117	419.01
MUS found in 50_215_3_26	60114	23230	54	116	470.34
MUS found in 50_215_3_27	92478	36676	73	128	818.97
MUS found in 50_215_3_28	49148	21248	47	109	366.52
MUS found in 50_215_3_29	57222	26319	47	110	432.66
MUS found in 50_215_3_3	72710	28104	53	120	594.33
MUS found in 50_215_3_34	45175	15467	45	107	282.30
MUS found in 50_215_3_35	100292	34649	69	125	843.57
MUS found in 50_215_3_36	41794	20736	47	104	318.05
MUS found in 50_215_3_37	55879	20710	62	110	384.91
MUS found in 50_215_3_40	49211	21334	51	111	354.29
MUS found in 50_215_3_42	53771	21797	55	113	433.20
MUS found in 50_215_3_45	77202	30919	63	120	596.49
MUS found in 50_215_3_49	53033	20585	57	107	357.79
MUS found in 50_215_3_50	40596	17249	50	102	248.34
MUS found in 50_215_3_51	41564	20623	40	100	297.37
MUS found in 50_215_3_56	51493	19639	45	109	359.25
MUS found in 50_215_3_57	43977	17105	56	106	280.49
MUS found in 50_215_3_60	84470	42339	67	123	809.59
MUS found in 50_215_3_64	65682	25402	55	112	507.78
MUS found in 50_215_3_66	40980	15682	47	107	257.36
MUS found in 50_215_3_68	59868	22565	50	109	413.70
MUS found in 50_215_3_71	93767	32267	65	125	755.95
MUS found in 50_215_3_72	58513	23562	57	121	455.09
MUS found in 50_215_3_73	88059	34324	64	119	756.47
MUS found in 50_215_3_75	58472	19810	54	115	414.59
MUS found in 50_215_3_78	78911	28460	59	124	611.05
MUS found in 50_215_3_82	42313	16009	41	107	299.60
MUS found in 50_215_3_83	47421	18517	47	106	322.43
MUS found in 50_215_3_87	48060	16617	49	106	327.44
MUS found in 50_215_3_88	47958	23112	49	105	365.66
MUS found in 50_215_3_9	60219	20743	60	114	412.12
MUS found in 50_215_3_93	55721	19182	53	111	383.47
MUS found in 50_215_3_95	85234	31867	63	128	761.26
MUS found in 50_215_3_96	71652	25319	59	110	508.99
MUS found in 50_215_3_97	39442	13851	52	109	269.55
MUS found in 50_215_3_99	67251	28486	56	113	541.09
Average	60245	23755	54	113	461.3

Mode *mC*

Filename	shifts	minisat shifts	iterations	MUS size	CPU time
60_258_3_1	161532	63077	104	139	2147.92
60_258_3_100	207055	104101			3598.82
60_258_3_13	176643	76724	102	136	2593.21
60_258_3_16	202576	93186			3598.66
60_258_3_17	149738	69841	98	149	2203.88
60_258_3_18	190876	79577	109	138	2654.67
60_258_3_21	202635	84488	123	153	3137.30
60_258_3_24	198447	102096	108	148	3363.73
60_258_3_27	194031	94670	114	157	3261.27
60_258_3_29	192673	85647	102	146	2967.06
60_258_3_3	211896	102465			3599.06
60_258_3_30	199287	94985	113	152	3269.65
60_258_3_31	166044	69220	95	131	2269.43
60_258_3_33	179233	78616	107	141	2711.19
60_258_3_35	208240	91684			3598.84
60_258_3_4	206759	85178	112	152	3341.28
60_258_3_40	195588	82638	98	157	3226.00
60_258_3_45	192887	95253			3598.58
60_258_3_46	212702	95282	108	149	3555.97
60_258_3_47	174070	84426	95	147	2931.71
60_258_3_48	143619	70816	84	140	2173.62
60_258_3_49	194776	82045	103	152	3005.97
60_258_3_50	148743	68211	101	130	2170.57
60_258_3_53	169633	87151	104	138	2741.36
60_258_3_54	181241	77797	107	139	2495.27
60_258_3_55	159310	71738	107	129	2349.35
60_258_3_56	157807	75458	87	140	2301.46
60_258_3_57	143420	61661	94	130	1911.50
60_258_3_58	114782	59513	81	137	1682.12
60_258_3_6	163188	79163	105	131	2387.46
60_258_3_62	195835	89134	102	137	2842.40
60_258_3_64	181998	82206	109	141	2909.20
60_258_3_66	143412	62016	98	130	1863.90
60_258_3_67	159290	67305	92	126	2159.82
60_258_3_69	149196	61983	88	141	2240.16
60_258_3_7	188278	80511	102	152	3060.07
60_258_3_71	211098	92164	104	156	3559.26
60_258_3_72	187846	80702	102	151	2902.03
60_258_3_74	189218	80193	113	147	2968.70
60_258_3_76	140484	64506	93	135	1994.68
60_258_3_8	220921	93317			3599.28
60_258_3_82	177639	77218	101	144	2451.64
60_258_3_83	151539	75355	97	127	2096.60
60_258_3_87	163271	78444	89	142	2408.12
60_258_3_88	201492	90475	99	141	3230.79
60_258_3_9	145622	58760	86	128	1837.98
60_258_3_90	156817	68922	102	140	2174.58
60_258_3_94	205685	93445	110	153	3449.40
60_258_3_96	177621	78488	92	132	2611.12
60_258_3_98	141063	66624	88	138	2073.70
Average	177755	80170	520	124	2313.7

Mode *mC*

Filename	shifts	minisat shifts	iterations	MUS size	CPU time
MUS found in 60_258_3_1	74694	30538	66	133	846.58
MUS found in 60_258_3_100	130257	65403	68	146	1878.23
MUS found in 60_258_3_13	95246	42640	69	136	1140.98
MUS found in 60_258_3_16	146108	68394	75	154	2178.65
MUS found in 60_258_3_17	137319	65113	77	150	1885.51
MUS found in 60_258_3_18	105723	41190	70	135	1197.99
MUS found in 60_258_3_21	143230	66013	81	150	1965.52
MUS found in 60_258_3_24	137925	74804	82	148	2027.12
MUS found in 60_258_3_27	173561	83540	87	163	2658.43
MUS found in 60_258_3_29	107993	42004	76	145	1354.44
MUS found in 60_258_3_3	152264	89105	81	151	2502.82
MUS found in 60_258_3_30	136709	67721	81	148	1837.95
MUS found in 60_258_3_31	103678	48142	59	133	1214.13
MUS found in 60_258_3_33	98589	42780	69	140	1250.20
MUS found in 60_258_3_35	124067	48644	80	146	1516.68
MUS found in 60_258_3_4	144394	62416	71	154	2132.14
MUS found in 60_258_3_40	128733	54343	79	157	1842.36
MUS found in 60_258_3_45	157409	67158	79	155	2145.82
MUS found in 60_258_3_46	110331	49508	65	150	1527.51
MUS found in 60_258_3_47	101362	47760	58	147	1372.63
MUS found in 60_258_3_48	102149	47740	65	140	1326.96
MUS found in 60_258_3_49	140657	65294	73	153	1938.91
MUS found in 60_258_3_50	69328	28040	62	128	708.40
MUS found in 60_258_3_53	127560	67059	76	137	1832.89
MUS found in 60_258_3_54	93181	44014	67	133	1166.58
MUS found in 60_258_3_55	90350	42043	58	131	1058.67
MUS found in 60_258_3_56	125892	64825	69	142	1741.95
MUS found in 60_258_3_57	92119	43227	63	129	1038.12
MUS found in 60_258_3_58	81966	37632	65	138	999.81
MUS found in 60_258_3_6	86444	39976	59	132	1098.28
MUS found in 60_258_3_62	87666	40674	60	133	1046.27
MUS found in 60_258_3_64	134027	63192	75	140	1829.68
MUS found in 60_258_3_66	63885	30067	57	125	703.27
MUS found in 60_258_3_67	107590	44353	78	127	1211.37
MUS found in 60_258_3_69	146754	67404	72	141	1932.62
MUS found in 60_258_3_7	143903	63359	85	153	1998.04
MUS found in 60_258_3_71	172296	76464	82	156	2552.33
MUS found in 60_258_3_72	147487	70382	70	150	2114.40
MUS found in 60_258_3_74	130825	56056	67	146	1745.58
MUS found in 60_258_3_76	88038	40556	64	130	1092.42
MUS found in 60_258_3_8	168166	68571	81	157	2552.64
MUS found in 60_258_3_82	115994	52404	78	140	1531.41
MUS found in 60_258_3_83	73141	34243	58	126	847.50
MUS found in 60_258_3_87	115818	55526	64	146	1585.21
MUS found in 60_258_3_88	97986	40764	61	142	1251.86
MUS found in 60_258_3_9	78257	31024	62	127	851.89
MUS found in 60_258_3_90	84529	40435	58	135	1094.51
MUS found in 60_258_3_94	132703	57626	68	151	1868.21
MUS found in 60_258_3_96	101416	40274	76	133	1180.53
MUS found in 60_258_3_98	107882	47856	70	139	1353.66
Average	116352	53166	70	142	1554.6

C.3 Hot start

Option *mS*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
50_215_3_105	40	0.78	33	0.38
50_215_3_108	40	0.67	41	0.43
50_215_3_109	36	0.77	38	0.42
50_215_3_110	49	0.99	36	0.46
50_215_3_111	39	0.76	36	0.51
50_215_3_112	45	0.87	31	0.4
50_215_3_12	33	0.67	37	0.62
50_215_3_14	38	0.76	38	0.39
50_215_3_15	38	0.72	41	0.49
50_215_3_16	42	0.86	35	0.43
50_215_3_2	35	0.67	29	0.32
50_215_3_20	33	0.67	34	0.38
50_215_3_21	40	0.7	43	0.49
50_215_3_23	32	0.61	37	0.43
50_215_3_26	45	0.77	32	0.38
50_215_3_27	38	0.77	34	0.59
50_215_3_28	39	0.7	44	0.47
50_215_3_29	36	0.69	25	0.29
50_215_3_3	40	0.79	33	0.48
50_215_3_34	36	0.61	35	0.36
50_215_3_35	38	0.72	38	0.61
50_215_3_36	32	0.6	31	0.32
50_215_3_37	40	0.67	28	0.32
50_215_3_40	33	0.67	32	0.35
50_215_3_42	41	0.76	38	0.41
50_215_3_45	37	0.58	36	0.48
50_215_3_49	42	0.8	33	0.34
50_215_3_50	37	0.69	36	0.36
50_215_3_51	39	0.71	38	0.39
50_215_3_56	42	0.83	28	0.28
50_215_3_57	32	0.6	36	0.37
50_215_3_60	31	0.65	42	0.54
50_215_3_64	40	0.77	35	0.4
50_215_3_66	35	0.6	31	0.33
50_215_3_68	39	0.73	32	0.33
50_215_3_71	48	0.89	43	0.59
50_215_3_72	42	0.83	37	0.5
50_215_3_73	37	0.75	38	0.46
50_215_3_75	40	0.74	38	0.43
50_215_3_78	40	0.83	48	0.7
50_215_3_82	33	0.65	33	0.35
50_215_3_83	35	0.61	35	0.37
50_215_3_87	39	0.74	38	0.4
50_215_3_88	36	0.58	36	0.36
50_215_3_9	30	0.59	32	0.35
50_215_3_93	38	0.7	41	0.44
50_215_3_95	38	0.7	45	0.73
50_215_3_96	43	0.81	40	0.42
50_215_3_97	39	0.78	38	0.41
50_215_3_99	44	0.88	43	0.47
Average	38	0.73	36	0.43

Option *mS*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
60_258_3_1	42	0.94	40	0.77
60_258_3_100	53	1.43	44	0.88
60_258_3_13	44	1.07	50	0.98
60_258_3_16	56	1.65	46	0.95
60_258_3_17	46	1.36	44	0.88
60_258_3_18	47	1.23	44	0.88
60_258_3_21	51	1.49	52	1.04
60_258_3_24	45	1.29	47	0.95
60_258_3_27	49	1.52	46	1.01
60_258_3_29	44	1.2	48	0.96
60_258_3_3	44	1.34	44	0.88
60_258_3_30	46	1.14	47	0.94
60_258_3_31	47	1.19	40	0.78
60_258_3_33	52	1.26	48	0.96
60_258_3_35	54	1.44	49	0.98
60_258_3_4	46	1.13	51	1.02
60_258_3_40	48	1.25	49	0.99
60_258_3_45	45	1.27	50	1.01
60_258_3_46	47	1.31	45	0.91
60_258_3_47	58	1.47	50	1
60_258_3_48	51	1.25	50	1
60_258_3_49	53	1.4	48	0.96
60_258_3_50	43	1	42	0.78
60_258_3_53	39	1.06	48	0.94
60_258_3_54	50	1.27	51	0.98
60_258_3_55	51	1.18	41	0.76
60_258_3_56	42	1.04	48	0.96
60_258_3_57	54	1.23	42	0.73
60_258_3_58	43	1.14	35	0.67
60_258_3_6	46	1.11	48	0.9
60_258_3_62	45	1.08	34	0.66
60_258_3_64	57	1.44	43	0.86
60_258_3_66	52	1.3	40	0.71
60_258_3_67	55	1.3	49	0.84
60_258_3_69	53	1.53	51	1.01
60_258_3_7	48	1.38	47	0.94
60_258_3_71	46	1.28	54	1.08
60_258_3_72	52	1.28	50	1
60_258_3_74	51	1.28	46	0.92
60_258_3_76	47	1.19	45	0.86
60_258_3_8	58	1.49	49	1
60_258_3_82	57	1.51	45	0.9
60_258_3_83	47	1.29	47	0.81
60_258_3_87	48	1.12	42	0.85
60_258_3_88	54	1.53	46	0.92
60_258_3_9	42	0.97	45	0.75
60_258_3_90	53	1.33	47	0.91
60_258_3_94	59	1.6	65	1.3
60_258_3_96	52	1.29	45	0.86
60_258_3_98	55	1.32	46	0.91
Average	49	1.28	46	0.91

Option *mS*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
150_645_3_1	120	59.03	120	34.79
150_645_3_11	128	55.88	122	32.38
150_645_3_14	127	50.43	125	31.67
150_645_3_15	129	61.02	117	33.75
150_645_3_16	130	71.16	121	43.48
150_645_3_17	114	50.59	111	27.39
150_645_3_2	131	64.66	118	30.89
150_645_3_21	129	66	119	39.73
150_645_3_24	123	57.46	120	34.7
150_645_3_25	123	55.41	121	33.31
150_645_3_26	123	60.63	120	32.16
150_645_3_28	122	48.53	115	28.52
150_645_3_3	134	57.54	120	31.55
150_645_3_34	122	67.02	123	39.12
150_645_3_36	122	52.37	118	28.66
150_645_3_37	124	79.14	114	46.41
150_645_3_38	127	56.73	124	36.88
150_645_3_39	133	62.17	118	35.8
150_645_3_41	122	48.44	114	27.03
150_645_3_42	124	54.42	119	32.66
150_645_3_44	139	48.5	127	31.26
150_645_3_45	128	54.19	121	31.06
150_645_3_46	120	53.86	119	31.98
150_645_3_47	126	59.39	120	35.08
150_645_3_48	138	51.18	128	32.08
150_645_3_49	134	56.56	122	29.98
150_645_3_5	125	55.65	130	34.32
150_645_3_50	124	57.42	117	31.57
150_645_3_51	129	54.4	119	30.24
150_645_3_52	136	54.79	123	31.05
150_645_3_57	123	50.24	117	27.34
150_645_3_58	127	58.01	111	34.15
150_645_3_59	127	51.73	126	29.77
150_645_3_6	130	53.81	119	30.26
150_645_3_60	119	43.81	121	27.42
150_645_3_61	127	71.43	124	40.51
150_645_3_62	124	63.07	124	40.82
150_645_3_63	132	44.75	118	25.04
150_645_3_69	130	55.39	121	31.58
150_645_3_70	119	69.56	114	42.03
150_645_3_71	129	51.82	116	26.22
150_645_3_72	129	37.83	120	22.81
150_645_3_74	128	52.24	128	30.99
150_645_3_78	121	64.32	117	37.09
150_645_3_79	126	56.81	123	33.51
150_645_3_8	127	47.35	127	27.3
150_645_3_80	124	55.99	117	31.09
150_645_3_83	129	46.95	120	26.43
150_645_3_85	128	57.09	127	31.72
150_645_3_9	137	52.63	121	29.12
Average	127	56.19	120	32.49

Option *mS*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
200_860_3_1	168	575.66	165	251.69
200_860_3_10	165	479.61	152	232.94
200_860_3_11	169	486.08	160	210.84
200_860_3_12	162	483.82	154	204.11
200_860_3_13	160	452.84	173	200.84
200_860_3_14	170	342.78	160	155.73
200_860_3_15	166	403.3	159	211.42
200_860_3_18	161	416.73	160	198.15
200_860_3_2	155	458.84	161	226.38
200_860_3_20	171	459.9	154	202.57
200_860_3_21	147	355.78	142	154.03
200_860_3_22	168	375.53	153	161.56
200_860_3_24	170	449.06	152	208.97
200_860_3_27	160	582.76	162	277.76
200_860_3_28	173	227.29	160	109.89
200_860_3_29	164	413.63	161	219.93
200_860_3_3	160	405.19	156	187.54
200_860_3_30	160	551.7	156	294.59
200_860_3_32	160	445.34	161	181.41
200_860_3_33	161	390.35	154	149.27
200_860_3_34	153	371.15	161	181.41
200_860_3_36	164	277.35	157	141.83
200_860_3_37	166	490.45	166	232.77
200_860_3_39	165	452.23	154	214.11
200_860_3_4	161	384.51	161	192.08
200_860_3_41	168	380.39	152	163.82
200_860_3_45	157	451.7	149	220.13
200_860_3_46	168	306.41	149	134.32
200_860_3_47	158	527.85	157	254
200_860_3_48	164	443.18	166	189
200_860_3_5	170	374.51	160	192.71
200_860_3_50	158	336.01	163	150.17
200_860_3_52	160	324.72	163	142.9
200_860_3_55	163	405.83	156	162.78
200_860_3_57	164	366.88	150	181.87
200_860_3_58	156	342.11	162	164.9
200_860_3_59	159	414.05	151	185.13
200_860_3_6	155	525.49	155	264.08
200_860_3_60	162	326.4	162	149.8
200_860_3_61	171	352.62	165	177.9
200_860_3_62	150	321.31	159	164.05
200_860_3_63	169	395.27	159	185.49
200_860_3_64	157	420.79	153	183.22
200_860_3_65	158	519.74	155	228.23
200_860_3_66	161	460.85	155	202.39
200_860_3_67	172	318.43	165	149.82
200_860_3_68	164	315.24	157	161.43
200_860_3_7	178	418.37	171	193.25
200_860_3_8	161	339.55	163	157.03
200_860_3_9	150	284.02	155	149
Average	163	408.07	158	190.18

Option *mS*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
C168_FW_SZ_128	9	20.97	7	0.37
C168_FW_SZ_41	10	22.05	3	0.03
C168_FW_SZ_66	15	26.04	7	0.13
C168_FW_SZ_75	12	21.19	5	0.1
C168_FW_UT_2463	15	49.03	8	0.1
C168_FW_UT_2468	17	58.71	7	0.12
C168_FW_UT_2469	16	51.17	7	0.12
C168_FW_UT_714	6	43.25	3	0
C168_FW_UT_851	4	29.94	3	0
C168_FW_UT_852	4	30.01	3	0
C168_FW_UT_854	4	29.96	3	0
C168_FW_UT_855	4	30.01	3	0
C170_FR_RZ_32	4	12.95	2	0.15
C170_FR_SZ_58	7	20.57	3	0.06
C170_FR_SZ_92	2	9.18	2	0.1
C170_FR_SZ_95	6	15.84	4	0.08
C170_FR_SZ_96	7	16.73	4	0.08
C202_FS_RZ_44	6	19.18	3	0.03
C202_FS_SZ_104	7	24.3	5	0.05
C202_FS_SZ_121	5	25.06	4	0.04
C202_FS_SZ_122	5	19.89	3	0.08
C202_FS_SZ_74	6	27.52	5	0.3
C202_FS_SZ_84	28	41.92	17	1.82
C202_FS_SZ_95	8	22.72	3	0.03
C202_FS_SZ_97	9	23.72	5	0.1
C202_FW_RZ_57	5	35.45	2	0.29
C202_FW_SZ_100	11	47.6	5	0.1
C202_FW_SZ_103	24	98.1	15	1.46
C202_FW_SZ_118	5	41.93	2	0.12
C202_FW_SZ_123	4	41.43	5	0.09
C202_FW_SZ_124	5	33.33	3	0.09
C202_FW_SZ_61	14	54.93	9	0.19
C202_FW_SZ_77	5	46.89	3	0.21
C202_FW_SZ_87	30	71.73	14	3.77
C202_FW_SZ_96	28	56.88	15	1.8
C202_FW_SZ_98	7	42.27	4	0.02
C202_FW_UT_2814	9	99.68	3	0.03
C202_FW_UT_2815	9	99.61	3	0.04
C208_FA_RZ_43	7	18.25	2	0
C208_FA_RZ_64	4	13.1	2	0.14
C208_FA_SZ_120	5	11.2	4	0.06
C208_FA_SZ_121	5	11.26	4	0.05
C208_FA_SZ_87	5	13.98	3	0.03
C208_FA_UT_3254	8	40.25	5	0.15
C208_FA_UT_3255	7	40.19	5	0.15

Continued on next page

Option *mS*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
C208_FC_RZ_65	8	25.37	4	0
C208_FC_RZ_70	4	14.29	2	0.15
C208_FC_SZ_107	11	33.9	7	0.14
C208_FC_SZ_127	3	11.99	3	0.06
C208_FC_SZ_128	3	11.88	3	0.06
C210_FS_RZ_23	11	24.9	5	0.1
C210_FS_RZ_38	9	18.26	5	0.05
C210_FS_RZ_40	5	16.01	2	0.09
C210_FS_SZ_103	12	27	8	0.23
C210_FS_SZ_107	6	18.41	3	0.03
C210_FS_SZ_123	4	18.38	2	0.16
C210_FS_SZ_129	5	15.99	3	0.06
C210_FS_SZ_130	5	15.79	3	0.06
C210_FS_SZ_55	12	30.38	5	0.11
C210_FS_SZ_78	11	32.79	5	0.35
C210_FW_RZ_30	11	41.38	6	0.12
C210_FW_RZ_57	10	27.79	5	0.05
C210_FW_RZ_59	5	25.24	2	0.1
C210_FW_SZ_106	13	46.46	9	0.3
C210_FW_SZ_111	7	34.34	5	0.05
C210_FW_SZ_128	8	34.68	3	0.03
C210_FW_SZ_129	4	29.91	2	0.16
C210_FW_SZ_135	5	25.02	3	0.05
C210_FW_SZ_136	5	24.88	3	0.06
C210_FW_SZ_80	7	41.43	4	0.29
C210_FW_SZ_90	33	77.78	22	4.74
C210_FW_SZ_91	29	65.8	21	4.47
C210_FW_UT_8630	8	76.52	3	0.06
C210_FW_UT_8634	8	76.75	3	0.03
C220_FV_RZ_12	9	18.15	3	0
C220_FV_RZ_13	8	16.61	3	0.01
C220_FV_RZ_14	8	15.88	3	0
C220_FV_SZ_114	4	13.5	2	0.11
C220_FV_SZ_121	8	13.5	6	0.15
C220_FV_SZ_39	27	41.74	15	1.3
C220_FV_SZ_46	9	18.37	4	0
C220_FV_SZ_55	35	48.44	20	2.89
C220_FV_SZ_65	7	11.82	3	0.03
Average	10	33.15	5	0.35

Option *mT*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
50_215_3_105	38	0.74	37	0.44
50_215_3_108	40	0.75	43	0.43
50_215_3_109	42	0.84	36	0.39
50_215_3_110	43	0.84	35	0.41
50_215_3_111	44	0.89	36	0.54
50_215_3_112	42	0.81	30	0.39
50_215_3_12	43	0.88	36	0.59
50_215_3_14	34	0.62	38	0.41
50_215_3_15	44	0.94	41	0.49
50_215_3_16	47	0.88	37	0.49
50_215_3_2	41	0.68	29	0.34
50_215_3_20	37	0.74	31	0.33
50_215_3_21	37	0.64	45	0.48
50_215_3_23	32	0.69	37	0.45
50_215_3_26	39	0.75	33	0.44
50_215_3_27	39	0.81	37	0.62
50_215_3_28	39	0.7	47	0.53
50_215_3_29	29	0.53	26	0.28
50_215_3_3	43	0.83	36	0.51
50_215_3_34	38	0.64	37	0.38
50_215_3_35	38	0.77	37	0.56
50_215_3_36	36	0.58	39	0.4
50_215_3_37	40	0.78	30	0.35
50_215_3_40	39	0.78	34	0.36
50_215_3_42	37	0.69	40	0.43
50_215_3_45	34	0.61	32	0.43
50_215_3_49	39	0.73	34	0.36
50_215_3_50	46	0.74	33	0.35
50_215_3_51	43	0.72	38	0.38
50_215_3_56	42	0.81	26	0.29
50_215_3_57	40	0.65	39	0.42
50_215_3_60	42	0.78	42	0.57
50_215_3_64	34	0.65	38	0.44
50_215_3_66	40	0.7	33	0.36
50_215_3_68	43	0.74	35	0.38
50_215_3_71	38	0.82	46	0.6
50_215_3_72	36	0.75	43	0.61
50_215_3_73	35	0.65	41	0.51
50_215_3_75	45	0.88	38	0.45
50_215_3_78	47	0.98	45	0.61
50_215_3_82	41	0.73	39	0.4
50_215_3_83	39	0.66	41	0.46
50_215_3_87	39	0.67	38	0.4
50_215_3_88	36	0.6	33	0.32
50_215_3_9	43	0.8	36	0.41
50_215_3_93	40	0.7	43	0.48
50_215_3_95	51	1	48	0.83
50_215_3_96	42	0.8	43	0.46
50_215_3_97	34	0.6	41	0.43
50_215_3_99	41	0.79	45	0.54
Average	40	0.75	38	0.45

Option *mT*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
60_258_3_1	44	0.94	47	0.87
60_258_3_100	57	1.53	42	0.84
60_258_3_13	52	1.31	50	0.97
60_258_3_16	59	1.82	45	0.91
60_258_3_17	51	1.44	39	0.78
60_258_3_18	49	1.34	46	0.92
60_258_3_21	58	1.68	49	0.97
60_258_3_24	44	1.31	47	0.94
60_258_3_27	52	1.59	48	0.99
60_258_3_29	46	1.26	46	0.92
60_258_3_3	51	1.47	44	0.88
60_258_3_30	45	1.11	49	0.98
60_258_3_31	52	1.24	39	0.74
60_258_3_33	52	1.32	49	0.97
60_258_3_35	50	1.35	50	1
60_258_3_4	46	1.23	48	0.96
60_258_3_40	54	1.46	50	1
60_258_3_45	54	1.44	48	0.96
60_258_3_46	45	1.2	48	0.96
60_258_3_47	52	1.43	52	1.04
60_258_3_48	59	1.48	49	0.98
60_258_3_49	51	1.4	45	0.9
60_258_3_50	48	1.17	38	0.7
60_258_3_53	47	1.24	50	0.99
60_258_3_54	44	1.26	53	1.05
60_258_3_55	48	1.18	44	0.81
60_258_3_56	47	1.16	49	0.98
60_258_3_57	42	1.04	41	0.7
60_258_3_58	52	1.34	42	0.84
60_258_3_6	56	1.37	44	0.86
60_258_3_62	47	1.2	36	0.69
60_258_3_64	47	1.21	47	0.94
60_258_3_66	45	1.13	43	0.71
60_258_3_67	51	1.27	44	0.74
60_258_3_69	61	1.63	50	1
60_258_3_7	49	1.33	49	0.98
60_258_3_71	54	1.4	54	1.08
60_258_3_72	53	1.33	51	1.02
60_258_3_74	44	1.15	51	1.02
60_258_3_76	51	1.21	50	0.89
60_258_3_8	53	1.49	45	0.9
60_258_3_82	55	1.43	46	0.91
60_258_3_83	53	1.33	40	0.71
60_258_3_87	47	1.13	39	0.78
60_258_3_88	47	1.4	47	0.93
60_258_3_9	47	1.07	38	0.61
60_258_3_90	49	1.19	46	0.9
60_258_3_94	52	1.49	62	1.24
60_258_3_96	49	1.23	50	0.95
60_258_3_98	51	1.33	47	0.93
Average	50	1.32	47	0.91

Option *mT*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
150_645_3_1	122	58.39	113	32.64
150_645_3_11	129	51.24	123	31.55
150_645_3_14	133	45.76	115	28.71
150_645_3_15	127	64.44	124	33.95
150_645_3_16	120	71.1	119	41.9
150_645_3_17	132	54.56	105	26.5
150_645_3_2	119	55.79	121	31.83
150_645_3_21	124	62.31	116	37.79
150_645_3_24	126	58.42	129	36.4
150_645_3_25	123	59.03	119	32.87
150_645_3_26	116	57.49	109	28.8
150_645_3_28	117	50.76	118	28.81
150_645_3_3	128	56.78	125	32.65
150_645_3_34	125	64.52	118	36.65
150_645_3_36	114	50.75	115	27.71
150_645_3_37	132	69.09	112	47.5
150_645_3_38	126	52.16	122	35.17
150_645_3_39	132	54.79	120	35.64
150_645_3_41	133	48.78	119	27.85
150_645_3_42	130	52.31	128	34.12
150_645_3_44	146	50.37	124	29.27
150_645_3_45	123	47.8	127	31.54
150_645_3_46	126	49.5	117	30.99
150_645_3_47	121	57.21	111	32.84
150_645_3_48	128	50.62	121	30.71
150_645_3_49	125	50.27	124	30.39
150_645_3_5	133	56.82	125	33.86
150_645_3_50	134	69.44	116	31.46
150_645_3_51	134	52.09	124	30.95
150_645_3_52	127	49.2	126	32.03
150_645_3_57	124	48.24	121	28.12
150_645_3_58	122	55.7	118	36.66
150_645_3_59	136	51.7	124	29.66
150_645_3_6	117	48.93	113	28.24
150_645_3_60	130	51.95	108	24.51
150_645_3_61	124	62.92	126	41.2
150_645_3_62	114	56.85	118	39.04
150_645_3_63	120	42.66	118	25.54
150_645_3_69	133	55.55	118	32.16
150_645_3_70	119	70.88	116	42.6
150_645_3_71	133	65.65	122	27.71
150_645_3_72	125	38.75	125	23.73
150_645_3_74	122	50.79	125	30.23
150_645_3_78	124	56.58	119	39.1
150_645_3_79	135	54.99	123	33.88
150_645_3_8	124	44.22	120	26.45
150_645_3_80	122	51.39	114	29.99
150_645_3_83	128	52.2	111	24.85
150_645_3_85	125	62.84	120	31.19
150_645_3_9	125	46.24	118	28.6
Average	126	54.82	119	32.13

Option *mT*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
200_860_3_1	173	538.27	164	261.17
200_860_3_10	174	486.95	153	225.32
200_860_3_11	158	342.27	148	197.97
200_860_3_12	171	481.6	160	210.74
200_860_3_13	151	449.17	163	201.88
200_860_3_14	170	305.36	170	159.55
200_860_3_15	164	381.5	161	216.54
200_860_3_18	162	382.78	159	195.8
200_860_3_2	161	353.6	160	224.28
200_860_3_20	168	457.96	157	196.58
200_860_3_21	158	383.49	151	165.54
200_860_3_22	164	414.4	157	166.73
200_860_3_24	164	536.09	148	206.08
200_860_3_27	154	517.81	157	273.55
200_860_3_28	154	230.73	157	112.83
200_860_3_29	169	448.37	156	208.35
200_860_3_3	159	426.82	158	197.08
200_860_3_30	157	540.76	151	285.5
200_860_3_32	169	448	166	196.21
200_860_3_33	169	373.99	149	148.83
200_860_3_34	159	298.36	157	182.36
200_860_3_36	163	253.75	155	141.6
200_860_3_37	162	454.09	153	223.12
200_860_3_39	160	550.27	152	209.68
200_860_3_4	174	373.54	158	186.33
200_860_3_41	175	349.28	154	166.75
200_860_3_45	154	491.03	142	211.93
200_860_3_46	168	317.94	158	144.64
200_860_3_47	156	515.13	154	257.08
200_860_3_48	166	478.98	160	186.46
200_860_3_5	155	371.28	149	180.75
200_860_3_50	151	345.35	153	142.53
200_860_3_52	176	366.02	151	136.59
200_860_3_55	162	390.06	152	162
200_860_3_57	168	409.06	156	186.11
200_860_3_58	158	350.37	153	158.23
200_860_3_59	167	400.58	160	199.55
200_860_3_6	167	530.11	148	253.62
200_860_3_60	166	328.66	161	143.36
200_860_3_61	172	311.71	160	179.02
200_860_3_62	158	291.7	150	161.02
200_860_3_63	161	467.76	149	177.93
200_860_3_64	164	459.69	151	175.22
200_860_3_65	155	451.28	146	204.44
200_860_3_66	166	443.55	154	199.42
200_860_3_67	168	396.21	157	140.78
200_860_3_68	161	276.43	152	153.73
200_860_3_7	169	439.46	168	193.07
200_860_3_8	165	334.12	156	152.34
200_860_3_9	162	343.06	149	139.75
Average	164	405.78	155	188

Option *mT*

Formula	original		MUS	
	calls	CPU time	calls	CPU Time
C168_FW_SZ_128	14	48.68	7	0.39
C168_FW_SZ_41	10	41.99	2	0.01
C168_FW_SZ_66	11	44.57	5	0.1
C168_FW_SZ_75	10	38.65	3	0.06
C168_FW_UT_2463	14	73.73	7	0.12
C168_FW_UT_2468	15	90.72	6	0.12
C168_FW_UT_2469	14	82.24	6	0.11
C168_FW_UT_714	5	36.37	3	0
C168_FW_UT_851	5	27.55	2	0
C168_FW_UT_852	5	27.53	2	0
C168_FW_UT_854	5	27.53	2	0
C168_FW_UT_855	5	27.55	2	0
C170_FR_RZ_32	4	20.95	2	0.14
C170_FR_SZ_58	10	29.17	2	0.04
C170_FR_SZ_92	3	15.49	2	0.08
C170_FR_SZ_95	6	17.75	5	0.1
C170_FR_SZ_96	6	19.51	5	0.1
C202_FS_RZ_44	7	43.36	3	0.03
C202_FS_SZ_104	7	46.47	3	0.03
C202_FS_SZ_121	7	46.1	4	0.04
C202_FS_SZ_122	5	45.3	3	0.09
C202_FS_SZ_74	6	42.42	3	0.18
C202_FS_SZ_84	25	59.74	15	1.76
C202_FS_SZ_95	8	48.64	3	0.03
C202_FS_SZ_97	10	51.43	4	0.08
C202_FW_RZ_57	5	90.39	2	0.28
C202_FW_SZ_100	9	71.65	5	0.1
C202_FW_SZ_103	25	192.03	15	1.41
C202_FW_SZ_118	5	70.88	3	0.18
C202_FW_SZ_123	5	53.39	4	0.08
C202_FW_SZ_124	5	88.04	3	0.08
C202_FW_SZ_61	13	113.24	7	0.14
C202_FW_SZ_77	5	60.09	3	0.2
C202_FW_SZ_87	26	116.47	14	3.79
C202_FW_SZ_96	16	62.32	14	1.71
C202_FW_SZ_98	8	74.81	3	0.02
C202_FW_UT_2814	11	149.1	2	0.02
C202_FW_UT_2815	11	150.86	2	0.03
C208_FA_RZ_43	7	22.79	2	0
C208_FA_RZ_64	4	22.37	2	0.15
C208_FA_SZ_120	5	21.2	3	0.05
C208_FA_SZ_121	5	21.8	3	0.04
C208_FA_SZ_87	6	23.79	2	0.01
C208_FA_UT_3254	9	58.51	5	0.15
C208_FA_UT_3255	9	58.37	5	0.15

Continued on next page

Option *mT*

Filename	original		MUS	
	calls	CPU time	calls	CPU Time
C208_FC_RZ_65	9	32.41	5	0.01
C208_FC_RZ_70	4	24.51	2	0.14
C208_FC_SZ_107	12	48.64	6	0.12
C208_FC_SZ_127	4	18.81	3	0.03
C208_FC_SZ_128	4	18.95	3	0.03
C210_FS_RZ_23	11	58.29	4	0.08
C210_FS_RZ_38	9	42.18	4	0.04
C210_FS_RZ_40	5	34.14	2	0.09
C210_FS_SZ_103	10	28.69	6	0.17
C210_FS_SZ_107	7	35.01	2	0
C210_FS_SZ_123	4	26.07	3	0.24
C210_FS_SZ_129	5	28.25	3	0.06
C210_FS_SZ_130	5	28.38	3	0.04
C210_FS_SZ_55	15	67.62	3	0.07
C210_FS_SZ_78	8	41.35	5	0.35
C210_FW_RZ_30	14	113.21	5	0.1
C210_FW_RZ_57	8	70.98	6	0.07
C210_FW_RZ_59	6	61.77	2	0.09
C210_FW_SZ_106	11	64.72	7	0.21
C210_FW_SZ_111	9	63.07	4	0.02
C210_FW_SZ_128	11	83.65	5	0.05
C210_FW_SZ_129	4	45.24	3	0.24
C210_FW_SZ_135	6	59.15	3	0.06
C210_FW_SZ_136	5	48.87	3	0.03
C210_FW_SZ_80	7	55.46	4	0.28
C210_FW_SZ_90	28	85.84	18	4.05
C210_FW_SZ_91	28	87.71	18	3.84
C210_FW_UT_8630	6	96.28	4	0.08
C210_FW_UT_8634	6	82.23	4	0.05
C220_FV_RZ_12	8	29.24	2	0
C220_FV_RZ_13	8	29	2	0.01
C220_FV_RZ_14	7	22.87	2	0
C220_FV_SZ_114	5	21.08	3	0.15
C220_FV_SZ_121	11	34.5	6	0.18
C220_FV_SZ_39	20	28.7	13	1.12
C220_FV_SZ_46	9	28.65	4	0.01
C220_FV_SZ_55	20	27.81	14	2.03
C220_FV_SZ_65	7	21.09	3	0.03
Average	9	52.7	5	0.32

C.4 Solving satisfiable formulas

Filename	shifts	hard shifts	CPU time
50_215_3_1	268	30	1.96
50_215_3_10	803	150	8.07
50_215_3_11	466	193	5.50
50_215_3_13	958	293	11.35
50_215_3_17	78	12	0.27
50_215_3_18	756	103	7.19
50_215_3_19	47	5	0.08
50_215_3_22	767	383	11.17
50_215_3_24	530	96	5.01
50_215_3_25	538	94	3.88
50_215_3_30	156	24	1.03
50_215_3_31	203	35	1.09
50_215_3_32	105	32	0.62
50_215_3_33	196	49	1.63
50_215_3_38	850	229	11.09
50_215_3_39	720	146	6.38
50_215_3_4	324	107	2.80
50_215_3_41	532	169	6.11
50_215_3_43	180	41	1.08
50_215_3_44	339	55	2.54
50_215_3_46	1070	243	11.64
50_215_3_47	96	21	0.40
50_215_3_48	308	47	2.12
50_215_3_5	47	6	0.09
50_215_3_52	97	27	0.50
50_215_3_53	533	184	6.69
50_215_3_54	74	18	0.27
50_215_3_55	842	255	9.94
50_215_3_58	432	142	4.71
50_215_3_59	521	85	4.46
50_215_3_6	83	33	0.40
50_215_3_61	34	12	0.07
50_215_3_62	224	52	1.14
50_215_3_63	283	69	2.88
50_215_3_65	795	145	7.72
50_215_3_67	73	13	0.21
50_215_3_69	208	51	1.66
50_215_3_7	413	142	3.47
50_215_3_70	57	8	0.14
50_215_3_74	320	57	2.38
50_215_3_76	463	222	4.92
50_215_3_77	47	23	0.18
50_215_3_79	93	32	0.51
50_215_3_8	54	7	0.13
50_215_3_80	727	98	6.69
50_215_3_81	712	282	8.06
50_215_3_84	359	78	2.96
50_215_3_85	680	148	7.02
50_215_3_86	68	38	0.34
50_215_3_89	889	270	7.69
Average	388	101	3.8

Filename	shifts	hard shifts	CPU time
60_258_3_10	1506	360	35.00
60_258_3_11	977	417	18.88
60_258_3_12	77	35	0.47
60_258_3_14	416	168	7.26
60_258_3_15	620	114	7.50
60_258_3_19	431	124	4.94
60_258_3_2	61	23	0.27
60_258_3_20	151	19	1.24
60_258_3_22	2455	1066	69.37
60_258_3_23	106	27	0.68
60_258_3_25	491	108	7.75
60_258_3_26	1131	356	21.00
60_258_3_28	115	52	1.02
60_258_3_32	936	256	21.32
60_258_3_34	1493	633	32.35
60_258_3_36	941	349	16.51
60_258_3_37	405	151	8.64
60_258_3_38	141	41	1.45
60_258_3_39	1311	425	27.87
60_258_3_41	486	170	9.21
60_258_3_42	214	70	3.10
60_258_3_43	149	34	1.36
60_258_3_44	106	31	0.84
60_258_3_5	84	18	0.42
60_258_3_51	483	160	7.04
60_258_3_52	132	28	1.02
60_258_3_59	159	32	1.66
60_258_3_60	68	13	0.28
60_258_3_61	330	76	4.18
60_258_3_63	1125	348	21.12
60_258_3_65	1140	422	20.57
60_258_3_68	1727	559	36.37
60_258_3_70	153	24	1.30
60_258_3_73	58	15	0.26
60_258_3_75	2019	663	53.30
60_258_3_77	88	54	0.76
60_258_3_78	121	79	1.34
60_258_3_79	2047	648	51.40
60_258_3_80	26	3	0.03
60_258_3_81	491	155	7.61
60_258_3_84	410	161	6.75
60_258_3_85	77	13	0.33
60_258_3_86	115	40	1.08
60_258_3_89	980	315	18.13
60_258_3_91	144	54	1.72
60_258_3_92	530	252	9.90
60_258_3_93	143	40	1.38
60_258_3_95	3252	1774	99.64
60_258_3_97	758	272	15.72
60_258_3_99	264	84	3.84
Average	633	227	13.3

Filename	shifts	hard shifts	CPU time
100_430_3_1	3469	2256	569.84
100_430_3_105	179	93	5.66
100_430_3_106	4411	2857	760.01
100_430_3_108	1724	729	226.07
100_430_3_110	2047	1025	196.34
100_430_3_112	273	165	16.87
100_430_3_17	275	120	18.72
100_430_3_2	254	127	13.67
100_430_3_20	170	129	5.42
100_430_3_26	1684	871	228.70
100_430_3_31	1689	761	194.44
100_430_3_33	271	144	17.03
100_430_3_34	1437	779	166.61
100_430_3_36	1240	613	174.47
100_430_3_37	1369	810	229.49
100_430_3_44	2662	1471	468.22
100_430_3_46	4608	3331	723.27
100_430_3_48	1911	741	191.34
100_430_3_50	1755	817	205.62
100_430_3_52	668	316	89.93
100_430_3_53	226	105	8.99
100_430_3_54	149	63	3.14
100_430_3_55	1239	737	208.53
100_430_3_57	2227	1344	352.69
100_430_3_58	197	102	5.65
100_430_3_59	1219	694	129.62
100_430_3_6	199	154	7.39
100_430_3_60	888	460	105.79
100_430_3_61	1601	786	203.89
100_430_3_63	266	91	15.56
100_430_3_65	3782	1774	486.92
100_430_3_67	272	142	15.98
100_430_3_68	281	140	17.11
100_430_3_7	133	88	3.63
100_430_3_74	197	122	6.70
100_430_3_75	231	100	9.10
100_430_3_76	119	56	2.26
100_430_3_78	2556	1625	358.26
100_430_3_85	1174	609	158.30
100_430_3_86	261	107	13.91
100_430_3_87	2334	1025	331.03
100_430_3_88	2608	1514	455.00
100_430_3_9	258	94	11.47
100_430_3_91	3507	2381	377.78
100_430_3_92	246	124	12.32
100_430_3_93	3733	2032	585.78
100_430_3_94	2300	1249	238.69
100_430_3_96	261	141	13.06
100_430_3_98	990	456	100.19
100_430_3_99	248	152	15.10
Average	1316	732	175.1

C.5 Proving MUS using MiniSat call for each clause

Filename	CPU Time	Filename	CPU Time	Filename	CPU Time	Filename	CPU Time
50_215_3_105	4.024	60_258_3_1	4.701	150_645_3_1	76.542	200_860_3_1	746.503
50_215_3_108	4.119	60_258_3_100	4.987	150_645_3_11	65.299	200_860_3_10	933.139
50_215_3_109	4.088	60_258_3_13	5.036	150_645_3_14	71.597	200_860_3_11	849.23
50_215_3_110	4.103	60_258_3_16	4.813	150_645_3_15	80.707	200_860_3_12	1011.82
50_215_3_111	4.166	60_258_3_17	4.828	150_645_3_16	62.239	200_860_3_13	903.543
50_215_3_112	3.768	60_258_3_18	4.941	150_645_3_17	72.739	200_860_3_14	604.708
50_215_3_12	3.96	60_258_3_21	5.036	150_645_3_2	77.85	200_860_3_15	470.565
50_215_3_14	3.864	60_258_3_24	4.635	150_645_3_21	82.275	200_860_3_18	704.031
50_215_3_15	4.008	60_258_3_27	5.147	150_645_3_24	65.208	200_860_3_2	997.481
50_215_3_16	3.944	60_258_3_29	5.005	150_645_3_25	60.088	200_860_3_20	708.127
50_215_3_2	4.152	60_258_3_3	4.925	150_645_3_26	51.635	200_860_3_21	855.252
50_215_3_20	4.087	60_258_3_30	4.749	150_645_3_28	78.64	200_860_3_22	1165.82
50_215_3_21	3.911	60_258_3_31	4.909	150_645_3_3	62.373	200_860_3_24	673.925
50_215_3_23	4.055	60_258_3_33	4.748	150_645_3_34	105.016	200_860_3_27	778.305
50_215_3_26	4.007	60_258_3_35	5.085	150_645_3_36	58.586	200_860_3_28	350.545
50_215_3_27	4.039	60_258_3_4	4.939	150_645_3_37	124.841	200_860_3_29	398.796
50_215_3_28	4.072	60_258_3_40	4.893	150_645_3_38	92.003	200_860_3_3	1135.15
50_215_3_29	3.912	60_258_3_45	4.844	150_645_3_39	75.82	200_860_3_30	1206.69
50_215_3_3	3.96	60_258_3_46	4.795	150_645_3_41	58.027	200_860_3_32	903.459
50_215_3_34	3.815	60_258_3_47	5.004	150_645_3_42	98.161	200_860_3_33	844.849
50_215_3_35	3.992	60_258_3_48	4.669	150_645_3_44	47.352	200_860_3_34	564.977
50_215_3_36	4.2	60_258_3_49	4.781	150_645_3_45	67.953	200_860_3_36	441.009
50_215_3_37	3.896	60_258_3_50	5.019	150_645_3_46	71.311	200_860_3_37	923.671
50_215_3_40	4.04	60_258_3_53	4.797	150_645_3_47	62.61	200_860_3_39	731.194
50_215_3_42	3.959	60_258_3_54	4.829	150_645_3_48	54.995	200_860_3_4	424.576
50_215_3_45	4.008	60_258_3_55	4.669	150_645_3_49	59.237	200_860_3_41	581.266
50_215_3_49	4.006	60_258_3_56	4.891	150_645_3_5	78.069	200_860_3_45	926.972
50_215_3_50	3.928	60_258_3_57	4.861	150_645_3_50	103.267	200_860_3_46	544.912
50_215_3_51	3.944	60_258_3_58	4.795	150_645_3_51	74.856	200_860_3_47	906.339
50_215_3_56	3.992	60_258_3_6	4.828	150_645_3_52	45.488	200_860_3_48	777.386
50_215_3_57	3.944	60_258_3_62	5.085	150_645_3_57	58.092	200_860_3_5	770.314
50_215_3_60	4.103	60_258_3_64	5.036	150_645_3_58	73.984	200_860_3_50	797.244
50_215_3_64	4.04	60_258_3_66	4.861	150_645_3_59	72.648	200_860_3_52	511.628
50_215_3_66	4.069	60_258_3_67	4.813	150_645_3_6	62.274	200_860_3_55	867.679
50_215_3_68	3.912	60_258_3_69	4.796	150_645_3_60	49.285	200_860_3_57	723.292
50_215_3_71	4.15	60_258_3_7	4.893	150_645_3_61	103.143	200_860_3_58	427.697
50_215_3_72	4.024	60_258_3_71	5.53	150_645_3_62	84.068	200_860_3_59	730.151
50_215_3_73	3.991	60_258_3_72	4.764	150_645_3_63	45.041	200_860_3_6	1198.51
50_215_3_75	3.944	60_258_3_74	4.845	150_645_3_69	69.085	200_860_3_60	574.208
50_215_3_78	4.072	60_258_3_76	4.525	150_645_3_70	108.388	200_860_3_61	604.911
50_215_3_82	4.104	60_258_3_8	4.925	150_645_3_71	60.68	200_860_3_62	878.766
50_215_3_83	3.912	60_258_3_82	4.845	150_645_3_72	41.509	200_860_3_63	800.427
50_215_3_87	3.862	60_258_3_83	4.86	150_645_3_74	50.03	200_860_3_64	753.205
50_215_3_88	3.784	60_258_3_87	4.685	150_645_3_78	81.55	200_860_3_65	917.262
50_215_3_9	3.895	60_258_3_88	5.053	150_645_3_79	78.394	200_860_3_66	733.069
50_215_3_93	3.927	60_258_3_9	4.765	150_645_3_8	51.453	200_860_3_67	446.822
50_215_3_95	3.975	60_258_3_90	4.669	150_645_3_80	56.224	200_860_3_68	728.583
50_215_3_96	4.102	60_258_3_94	4.957	150_645_3_83	63.765	200_860_3_7	705.75
50_215_3_97	3.864	60_258_3_96	4.972	150_645_3_85	77.994	200_860_3_8	524.158
50_215_3_99	3.926	60_258_3_98	4.893	150_645_3_9	57.8	200_860_3_9	544.591
Average	3.99		4.88		70.6		746.05

MUS'es as input

MUS found in	CPU Time	MUS found in	CPU Time	Filename	CPU Time	MUS found in	CPU Time
50_215_3_99	2.13	60_258_3_98	2.53	150_645_3_1	36.88	200_860_3_9	178.69
50_215_3_97	1.92	60_258_3_96	2.27	150_645_3_11	29.35	200_860_3_8	223.48
50_215_3_96	1.95	60_258_3_94	2.74	150_645_3_14	24.3	200_860_3_7	260.68
50_215_3_95	2.37	60_258_3_90	2.36	150_645_3_15	32.03	200_860_3_68	212.75
50_215_3_93	1.95	60_258_3_9	2.22	150_645_3_16	44.04	200_860_3_67	168.21
50_215_3_9	2.08	60_258_3_88	2.55	150_645_3_17	25.24	200_860_3_66	308.59
50_215_3_88	1.83	60_258_3_87	2.68	150_645_3_2	29.73	200_860_3_65	411.49
50_215_3_87	1.8	60_258_3_83	2.29	150_645_3_21	42.53	200_860_3_64	254.87
50_215_3_83	1.83	60_258_3_82	2.66	150_645_3_24	32.56	200_860_3_63	277.41
50_215_3_82	1.84	60_258_3_8	2.8	150_645_3_25	25.3	200_860_3_62	249.62
50_215_3_78	2.34	60_258_3_76	2.33	150_645_3_26	29.99	200_860_3_61	223.59
50_215_3_75	1.95	60_258_3_74	2.7	150_645_3_28	24.51	200_860_3_60	198.35
50_215_3_73	2.25	60_258_3_72	2.7	150_645_3_3	33.43	200_860_3_6	485.02
50_215_3_72	2.04	60_258_3_71	3.07	150_645_3_34	42.34	200_860_3_59	282.62
50_215_3_71	2.24	60_258_3_7	2.85	150_645_3_36	26.53	200_860_3_58	184.54
50_215_3_68	1.83	60_258_3_69	2.47	150_645_3_37	74.7	200_860_3_57	248.58
50_215_3_66	1.76	60_258_3_67	2.37	150_645_3_38	33.65	200_860_3_55	239.91
50_215_3_64	1.87	60_258_3_66	2.07	150_645_3_39	39.51	200_860_3_52	191.9
50_215_3_60	2.13	60_258_3_64	2.56	150_645_3_41	26.18	200_860_3_50	217.21
50_215_3_57	2.05	60_258_3_62	2.46	150_645_3_42	27.41	200_860_3_5	291.06
50_215_3_56	1.95	60_258_3_6	2.44	150_645_3_44	24.71	200_860_3_48	305.59
50_215_3_51	1.68	60_258_3_58	2.45	150_645_3_45	32.08	200_860_3_47	386.97
50_215_3_50	1.79	60_258_3_57	2.3	150_645_3_46	28.33	200_860_3_46	191.08
50_215_3_49	1.96	60_258_3_56	2.53	150_645_3_47	26.4	200_860_3_45	341.09
50_215_3_45	2.15	60_258_3_55	2.3	150_645_3_48	31.41	200_860_3_41	230.7
50_215_3_42	2.05	60_258_3_54	2.44	150_645_3_49	24.97	200_860_3_4	280.51
50_215_3_40	1.94	60_258_3_53	2.45	150_645_3_5	30.54	200_860_3_39	284.53
50_215_3_37	2	60_258_3_50	2.36	150_645_3_50	29.83	200_860_3_37	456.08
50_215_3_36	1.83	60_258_3_49	2.85	150_645_3_51	25.82	200_860_3_36	158.94
50_215_3_35	2.4	60_258_3_48	2.55	150_645_3_52	27.7	200_860_3_34	265.59
50_215_3_34	2.07	60_258_3_47	2.7	150_645_3_57	19.46	200_860_3_33	192.83
50_215_3_3	2.1	60_258_3_46	2.73	150_645_3_58	34.39	200_860_3_32	271.23
50_215_3_29	2	60_258_3_45	2.93	150_645_3_59	27.15	200_860_3_30	457.69
50_215_3_28	1.75	60_258_3_40	2.88	150_645_3_6	22.52	200_860_3_3	283.8
50_215_3_27	2.22	60_258_3_4	2.71	150_645_3_60	18.97	200_860_3_29	312.8
50_215_3_26	2.14	60_258_3_35	2.6	150_645_3_61	41.88	200_860_3_28	105.36
50_215_3_23	2.22	60_258_3_33	2.48	150_645_3_62	38.36	200_860_3_27	458.59
50_215_3_21	1.92	60_258_3_31	2.47	150_645_3_63	18.27	200_860_3_24	375.66
50_215_3_20	1.92	60_258_3_30	2.79	150_645_3_69	29.74	200_860_3_22	269.34
50_215_3_2	1.97	60_258_3_3	2.66	150_645_3_70	39.47	200_860_3_21	246.86
50_215_3_16	2.06	60_258_3_29	2.65	150_645_3_71	25.73	200_860_3_20	307.34
50_215_3_15	1.93	60_258_3_27	3.02	150_645_3_72	16.97	200_860_3_2	331.64
50_215_3_14	1.69	60_258_3_24	2.56	150_645_3_74	24.13	200_860_3_18	285.25
50_215_3_12	2.41	60_258_3_21	2.68	150_645_3_78	33.57	200_860_3_15	278.83
50_215_3_112	2.12	60_258_3_18	2.25	150_645_3_79	29.81	200_860_3_14	201.48
50_215_3_111	2.2	60_258_3_17	2.76	150_645_3_8	22.11	200_860_3_13	308.05
50_215_3_110	2.17	60_258_3_16	2.74	150_645_3_80	24.47	200_860_3_12	385.81
50_215_3_109	2.04	60_258_3_13	2.34	150_645_3_83	22.66	200_860_3_11	288.53
50_215_3_108	1.64	60_258_3_100	2.68	150_645_3_85	27.86	200_860_3_10	391.35
50_215_3_105	2.01	60_258_3_1	2.39	150_645_3_9	28.47	200_860_3_1	356.29
Average	2.01		2.57		30.16		282.37

MUS found in	CPU Time	MUS found in	CPU Time
C168_FW_SZ_128	1.898	C210_FS_RZ_23	0.667
C168_FW_SZ_41	0.559	C210_FS_RZ_38	0.529
C168_FW_SZ_66	0.849	C210_FS_RZ_40	2.547
C168_FW_SZ_75	0.935	C210_FS_SZ_103	0.954
C168_FW_UT_2463	0.713	C210_FS_SZ_107	0.334
C168_FW_UT_2468	0.681	C210_FS_SZ_123	3.263
C168_FW_UT_2469	0.682	C210_FS_SZ_129	0.606
C168_FW_UT_714	0.15	C210_FS_SZ_130	0.56
C168_FW_UT_851	0.197	C210_FS_SZ_55	0.863
C168_FW_UT_852	0.181	C210_FS_SZ_78	3.247
C168_FW_UT_854	0.181	C210_FW_RZ_30	0.699
C168_FW_UT_855	0.165	C210_FW_RZ_57	0.529
C170_FR_RZ_32	4.186	C210_FW_RZ_59	2.69
C170_FR_SZ_58	0.94	C210_FW_SZ_106	1.171
C170_FR_SZ_92	2.475	C210_FW_SZ_111	0.409
C170_FR_SZ_95	0.955	C210_FW_SZ_128	0.47
C170_FR_SZ_96	0.954	C210_FW_SZ_129	3.55
C202_FS_RZ_44	0.486	C210_FW_SZ_135	0.622
C202_FS_SZ_104	0.606	C210_FW_SZ_136	0.592
C202_FS_SZ_121	0.439	C210_FW_SZ_80	3.281
C202_FS_SZ_122	0.638	C210_FW_SZ_90	16.788
C202_FS_SZ_74	2.713	C210_FW_SZ_91	17.16
C202_FS_SZ_84	5.04	C210_FW_UT_8630	0.62
C202_FS_SZ_95	0.425	C210_FW_UT_8634	0.468
C202_FS_SZ_97	0.989	C220_FV_RZ_12	0.212
C202_FW_RZ_57	4.327	C220_FV_RZ_13	0.181
C202_FW_SZ_100	0.756	C220_FV_RZ_14	0.212
C202_FW_SZ_103	3.774	C220_FV_SZ_114	2.521
C202_FW_SZ_118	2.54	C220_FV_SZ_121	1.029
C202_FW_SZ_123	0.761	C220_FV_SZ_39	5.014
C202_FW_SZ_124	0.874	C220_FV_SZ_46	0.318
C202_FW_SZ_61	0.894	C220_FV_SZ_55	9.463
C202_FW_SZ_77	2.976	C220_FV_SZ_65	0.47
C202_FW_SZ_87	14.053	C208_FA_SZ_87	0.347
C202_FW_SZ_96	5.419	C208_FA_UT_3254	1.492
C202_FW_SZ_98	0.364	C208_FA_UT_3255	1.416
C202_FW_UT_2814	0.592	C208_FC_RZ_65	0.257
C202_FW_UT_2815	0.528	C208_FC_RZ_70	4.072
C208_FA_RZ_43	0.213	C208_FC_SZ_107	1
C208_FA_RZ_64	3.834	C208_FC_SZ_127	0.653
C208_FA_SZ_120	0.685	C208_FC_SZ_128	0.575
C208_FA_SZ_121	0.543		
Average	1.96		