Effect of Autarky Pruning on Random and Circuit Formulas: An Experimental Study[*]

Fumiaki Kamiya

UCSC-CRL-96-13
June 30, 1996

Baskin Center for
Computer Engineering & Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

Modoc was proposed by Van Gelder as an improvement to model elimination. The main contribution of Modoc is in its new pruning technique based on the concept of autarky, first introduced by Monien and Speckenmeyer. Compared to programs based on model search, Modoc has since been observed to excel in speed on circuit formulas yet to fall behind on random formulas. This paper reports on a study conducted to explain this behavior in Modoc. The study is based on experiment in which the effectiveness of the two pruning techniques used in Modoc—autarky pruning and lemma pruning—are examined on random and circuit formulas. We observe that the effectiveness of autarky pruning differs tremendously between the two classes of formulas. We also observe that for circuit formulas, autarkies, which are believed to be very few of, are more likely to be found by simplifying the formula for some partial truth assignment. This may lead to possible new ways to use autarkies to solve the satisfiability problem.

Keywords: Propositional satisfiability, refutation, model elimination, Modoc, autarky pruning, conditional autarky.

[*]Submitted for publication.

formula class	num fmlas	modoc avg	modoc max	2cl avg	2cl max
rand100	200	1.54	4.74	1.10	3.23
rand141	200	25.39	89.95	8.76	21.73
bf2670	53	7.15	181.49	439.01	17724.57
ssa2670	12	30.42	118.78	1539.60	2424.46

Figure 1: CPU times in seconds of modoc and 2cl on Sun Sparcstation 10/41. 2cl is based on model search and is described in [9]. Formula classes are explained in Section 3.

1 Introduction

Modoc was proposed by Van Gelder in [7] as an improvement to model elimination ([3]). Its main contribution to model elimination is in its new pruning technique based on the concept of autarky, first introduced by Monien and Speckenmeyer in [6]. Whereas Monien and Speckenmeyer's use of autarkies was to find a model, Modoc uses them to exclude clauses whose use will not lead to a successful refutation.

In [8], it was observed that compared to satisfiability testers based on model search, Modoc lagged behind on random formulas yet it performed far better on circuit formulas. (For completeness of this paper, the relevant numbers are reproduced in Figure 1.)

This paper reports on a study conducted to explain the difference in performance of Modoc on random and circuit formulas. The study is based on experiment in which the changes in the autarky size and the number of lemmas, both of which support the two pruning techniques employed in Modoc, are examined.

This paper is organized as follows: In the next section, we standardize the terminology used in the remainder of this paper and give a brief description on the concept of autarky and on the Modoc algorithm. Following a short section on the experiment, in Section 4, we examine the outcome of the experiment with figures depicting the changes in the autarky size and the number of lemmas. Section 5 contains an experiment performed to show that our conclusion on random formulas made in Section 4 applies to other random formulas. In Section 6, we discuss what the outcome may lead to. The paper ends with a conclusion. Appendix contains figures that could not be placed in the main body of the paper.

2 Terminology

This section gives a brief summary of the terms used in the remainder of this paper and a brief description on the concept of autarky and on the essence of the Modoc algorithm.

We assume the readers to be familiar with terms such as CNF formula, clause, literal, variable, truth assignment, etc.

One possible deviation from standard use is the term *satisfying truth assignment*. Following [7], we consider any partial truth assignment that satisfies all the clauses in the formula to be a satisfying truth assignment. This means that a satisfying truth assignment is not necessarily total and that some of the variables may remain unassigned.

By definition, truth assignments are functions that map variables to truth values. Note, however, that each truth assignment can be viewed as a set of literals that are made true by the truth assignment. In this paper, we will often take this set-theoretic view of the truth assignment as opposed to the usual functional view.

2.1 Autarky

Intuitively, an *autarky* of a formula is a partial truth assignment that can reduce the satisfiability problem of the formula to the satisfiability problem on the subset of the clauses that are not satisfied by the autarky. More formally, an autarky A of a CNF formula $\mathcal{F}$ partitions $\mathcal{F}$ into two subsets, $autsat(\mathcal{F}, A)$ and $autrem(\mathcal{F}, A)$, such that any clause in $autsat(\mathcal{F}, A)$ contains a literal in A (and hence is satisfied by A), and any clause in $autrem(\mathcal{F}, A)$ contains no literal in $A \cup \bar{A}$, where $\bar{A}$ consists of the complements of the literals in A. Obviously, the satisfiability problem of $\mathcal{F}$ reduces to the satisfiability problem on $autrem(\mathcal{F}, A)$. In case $\mathcal{F}$ is satisfiable, a satisfying truth assignment for $\mathcal{F}$ can be constructed from the disjoint union of A and a satisfying truth assignment of $autrem(\mathcal{F}, A)$.

Example 1: Let $\mathcal{F}$ be $\{\{a, \neg c, \neg e\}, \{\neg b, c\}, \{\neg a, b, d\}, \{\neg d, e\}\}$ where a, b, c, d, e are variables. Then, $\{a, b, c\}$ is an autarky of $\mathcal{F}$ but $\{a, c\}$ is not.

Using an autarky $\{a, b, c\}$, the satisfiability problem of $\mathcal{F}$ is reduced to the satisfiability problem on $\{\{\neg d, e\}\}$. Since $\{\{\neg d, e\}\}$ is satisfiable, so is $\mathcal{F}$. A satisfying truth assignment for $\mathcal{F}$ can be obtained from the disjoint union of $\{a, b, c\}$ and a satisfying truth assignment for $\{\{\neg d, e\}\}$, say $\{\neg d\}$, as $\{a, b, c, \neg d\}$.

We say that an autarky is *properly partitioning* if it is not empty and it is not a satisfying truth assignment. A properly partitioning autarky A can partition a formula $\mathcal{F}$ into two subsets, $autsat(\mathcal{F}, A)$ and $autrem(\mathcal{F}, A)$, such that neither is empty. A *conditional autarky* is a partial truth assignment that becomes an autarky after simplifying the formula using some other partial truth assignment. Reason for such autarkies will become clear in the next section (Section 2.2).

2.2 Modoc

We now give an informal description of the *Modoc* algorithm focusing on the parts of the algorithm that are most relevant to this study. Full detail of the algorithm can be found in [7].

Modoc is based on model elimination ([3]) which is a refutation procedure. Unlike model elimination which uses linear "chains" to represent the state of search, Modoc uses a tree-based data structure ([4]) to represent the search space which it explores.

The aim of Modoc is to refute a clause, called the *top clause*; this shows that the formula is unsatisfiable. If no clause can be refuted, then the formula is satisfiable. To refute a clause, Modoc tries to refute all the literals in it, and to refute a literal, it tries to find a resolving clause that can be refuted; the process continues recursively. Whenever a refutation attempt for a literal succeeds, the result is saved as a *lemma*. This eliminates the need to re-refute the same literal later under the same premise. When search leaves the subtree that supports the premise of a lemma, that lemma is thrown out.

The main difference between Modoc and model elimination is in Modoc's response to failed refutation attempts of literals. Whenever such attempts fail, the literals are added to the current autarky. Theoretical basis (correctness, etc.) can be found in [7]. Autarkies are used to prune subspaces that are known not to have a refutation. To be precise, any clause that contains a literal in the current autarky can be excluded from resolution as it cannot be refuted. As with lemmas, when search leaves the subtree that supports the premise in which the refutation failed for a literal, the autarky is retracted to the state before the literal was added.

Readers are reminded that an autarky found during search is not necessarily an autarky for the original formula. It is only an autarky in the subtree rooted at the place it was found. Such autarkies are "conditional" in the sense that if literals whose refutations are attempted along the

formula name	# of vars	num of clauses							# of lits
		len=1	len=2	len=3	len=4	len=5	len=6	total	
`bf2670-126.cnf`	694	5	1152	346	104	6	0	1613	3793
`bf2670-240.cnf`	1734	5	3084	1609	206	33	4	4941	12013

Figure 2: Some statistics on the `bf2670` family of formulas. `bf2670-126.cnf` is the smallest formula in `bf2670` in terms of the numbers of variables, clauses, and literals. `bf2670-240.cnf` is one of the largest formulas in terms of the numbers of clauses and literals.

path from the top clause to the current position in the tree are assumed true and the formula simplified accordingly, then would the autarky become an autarky for the simplified formula. To emphasize the conditional nature of such autarkies, we shall call them *conditional autarkies.*

Example 2: We continue with Example 1. Although $\{a, c\}$ is not an autarky, it is a conditional autarky, with respect to a partial truth assignment in which only d is assigned true. This can be verified by simplifying $\mathcal{F}$ with the partial truth assignment, which yields $\{\{a, \neg c, \neg e\}, \{\neg b, c\}, \{e\}\}$.

3 Experiment

In this section, we describe the experiment; the results are summarized in the next section (Section 4).

A C implementation of Modoc reported in [8], `modoc`, was modified to collect the necessary information. Modification was made so that every time a call to one of the autarky manipulating functions or to one of the lemma management functions is made, `modoc` will print the new autarky size or the new number of lemmas, respectively.

Two classes of formulas were considered for the study, random formulas and circuit formulas.

Random Formulas: This class consists of 50 141-variable 602-clause 3CNF random formulas generated from a probability model in which every non-redundant 3-clause is equally probable. Clauses-to-variables ratio was chosen to be 4.27. This ratio is believed to generate the most difficult formulas ([5, 2]). (`rand141` in Figure 1 refers to this class. `rand100` is a class of 100-variable 427-clause formulas generated from the same probability model.)

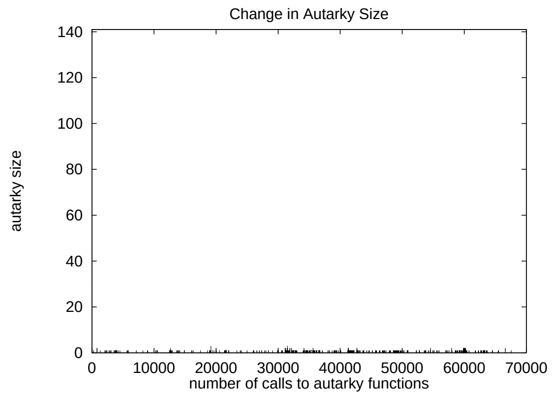
Circuit Formulas: This class consists of the 53 formulas of the `bf2670` family of formulas, which are generated from an *automated test pattern generation* program ([1]). The formulas are satisfiable if and only if the outputs of the fault-free and faulty circuits differ for some common input. The `2670` circuit is an ALU and has 1193 gates. The fault simulated for `bf` formulas is bridge fault. (The fault simulated for `ssa` formulas in Figure 1 is "single stuck-at" fault.) The circuit is taken from the ISCAS85 benchmark. Some statistics for this class of formulas are shown in Figure 2.

Note that whereas random formulas are purely artificial, circuit formulas come from real applications.

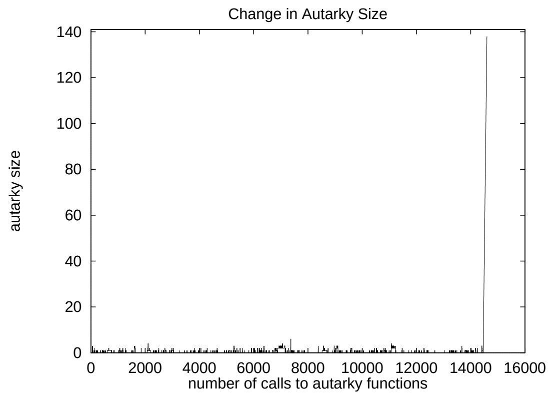
Output from each run of `modoc` was tabulated and fed into `gnuplot` to plot the changes in the autarky size and the number of lemmas during search.

4 Observation

This section summarizes the changes in the autarky size and the number of lemmas encountered by `modoc` during search for each formula tested. For each graph shown in this paper, the horizontal



(a) Unsatisfiable Random Formulas



(b) Satisfiable Random Formulas

Figure 3: Typical changes in autarky size encountered by `modoc` on random formulas. (Figure (a) is from `3.141.602.002-386059.cnf`. Figure (b) is from `3.141.602.001-386058.cnf`.)

range was determined automatically by `gnuplot` whereas the vertical range was chosen to be the range of variables used in the formula.

4.1 Change in the Autarky Size

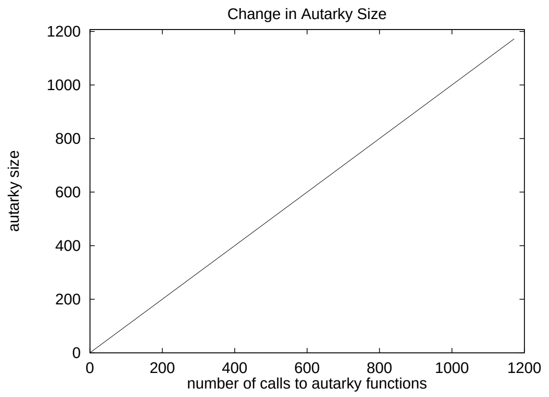
We first look at the change in the autarky size as this shows a tremendous difference between the two classes of formulas. Readers are reminded that autarkies mentioned in this section are conditional autarkies and not necessarily autarkies for the original formula.

For random formulas, there was a clear dichotomy in the change in the autarky size, each corresponding to satisfiable and unsatisfiable formulas, yet, they shared a common characteristic. For unsatisfiable formulas, the graphs show a line at size zero with some "noise" on it. A typical graph is shown in Figure 3(a). For satisfiable formulas, the graphs are similar except for the addition of a "sudden surge" to a satisfying truth assignment at the very end. A typical graph is shown in Figure 3(b). What appears to be a surge is in fact a slope. This is because autarky literals are always added one at a time (although they can be removed more than one at a time). Due to extreme scaling along the horizontal axis, the slope is steepened to an extreme and hence drawn like a sudden surge. (See Figure 9 in Appendix for a close-up of the last 200 function calls.)

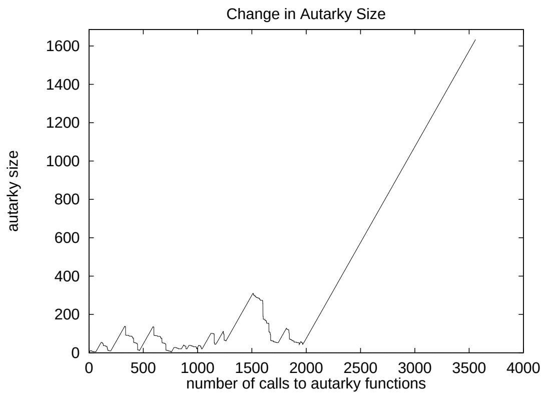
Both figures show that only extremely small autarkies were found by `modoc`. Of course, there were exceptions where "spikes" were observed. (See Figure 10 in Appendix for an example.) The fact that `modoc` was only able to find extremely small autarkies in the formulas means that Modoc was not able to prune effectively based on autarkies. This may, at least in part, explain the deficiency of Modoc on random formulas compared to circuit formulas, as we will see next.

Circuit formulas, on the other hand, showed various behaviors. There were quite a few formulas that led straight to a satisfying truth assignment, without any waste of work, as in Figure 4(a). Figure 4(b) is similar to Figure 4(a) in the second half, yet it shows some "struggle" in the first half. There was also an opposite of Figure 4(b). (See Figure 11 in Appendix for an example.)

Figure 5 shows some of the other often-observed graphs in which a repetition of various "spires" is seen. In Figure 5(a), the spires are similar in height and seen at regular intervals, whereas in

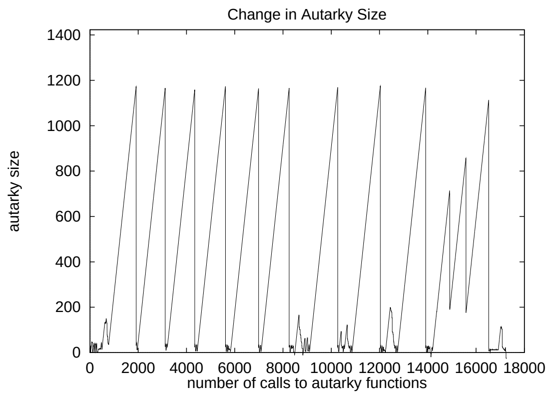


(a) `bf2670-414.cnf`

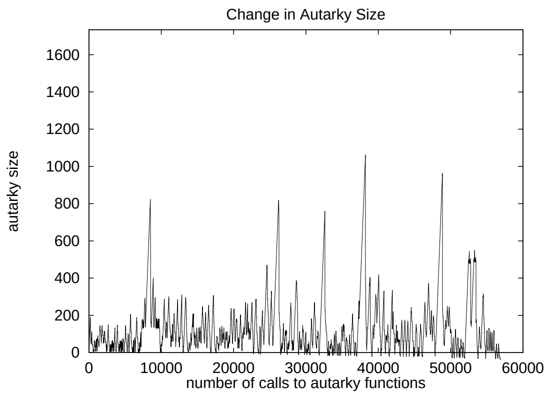


(b) `bf2670-483.cnf`

Figure 4: Changes in the autarky size encountered by `modoc` on circuit formulas.



(a) `bf2670-487.cnf`



(b) `bf2670-501.cnf`

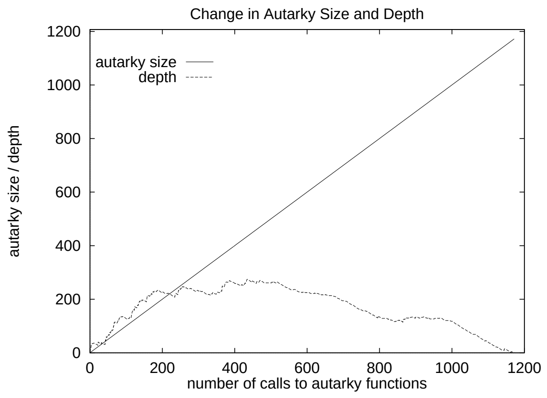
Figure 5: Changes in the autarky size encountered by `modoc` on circuit formulas.

Figure 5(b), the spires are of various heights and seen at irregular intervals. Also note the heights of the spires in each of these graphs. In Figure 5(a), `modoc` was able to find autarkies that contained approximately 80% of the variables, whereas in Figure 5(b), the largest autarky `modoc` was able to find was slightly above 50%. Apart from the graphs in Figure 5, there were graphs in which the spires were in increasing heights. (See Figure 12 in Appendix for an example.)

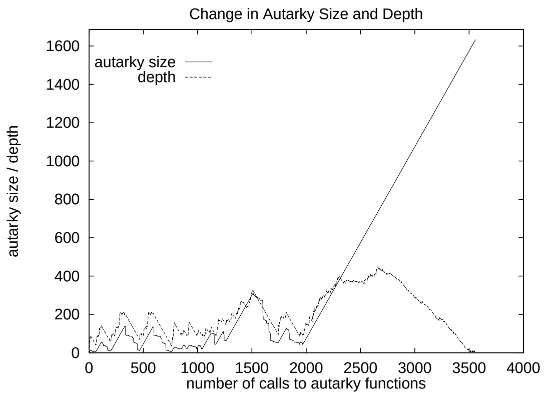
Other interesting graphs include "plateaus". (See Figure 13 in Appendix for examples.)

Although upward slopes in the figures appear to have different gradient, this is simply due to the difference in horizontal scaling. Should the graphs be drawn with the same horizontal scaling, all upward slopes would have been drawn with the same gradient.

With the exception of one formula (`bf2670-035.cnf` shown in Figure 13(b) in Appendix), a common observation was that autarkies `modoc` found for circuit formulas were much larger than those found for random formulas. With the previous observation of `modoc` being able to find only extremely small autarkies for random formulas, it appears that we may attribute the difference in



(a) `bf2670-414.cnf`



(b) `bf2670-483.cnf`

Figure 6: Changes in the autarky size and the depth at which the changes occurred in `modoc` on circuit formulas.

behavior to the relative abundance of autarkies in circuit formulas and the near lack of autarkies in random formulas.

As mentioned at the beginning of this section, the autarkies found by Modoc are not necessarily autarkies for the original formula. So one may ask, how many of them are autarkies for the original formulas? Of the 103 formulas tested, only in two of them was `modoc` able to find properly partitioning autarkies for the original formulas. (See Figure 14 in Appendix for details.) Interestingly, both formulas were satisfiable random formulas, and all properly partitioning autarkies that were found were very close to the satisfying truth assignments found by `modoc` in terms of the number of literals in them.

Instead of looking at (unconditional) autarkies, we now look at the depths at which conditional autarkies are found. Note that the depth corresponds to the size of the partial truth assignment on which the conditional autarky depends. Figure 6 shows two graphs with the depths at which the changes occurred also plotted. Observe that toward the end of search, large autarkies are found at shallow depths. This means that there are large conditional autarkies with relatively small partial truth assignments on which they depend.

4.2 Change in the Number of Lemmas

While there was a significant difference in the change in the autarky size between the two classes of formulas, nothing similar was observed for the change in the number of lemmas derived during search. The changes in the number of lemmas for random formulas were all quite similar; a typical graph is shown in Figure 7. For circuit formulas, there were variations, yet they were not as dramatic as the graphs for the change in the autarky size; two graphs are shown in Figure 8.

Although the graphs differ in the average number of lemmas relative to the number of variables, they share the same characteristic that the lemmas are abundant. Rate of fluctuation for circuit formulas appears to be slower than that for random formulas. (The rates of fluctuation for the two graphs in Figure 8 are actually quite similar. What makes the two look different is the use of different scalings along the horizontal axes. Figure 15 in Appendix shows a graph for `bf2670-501.cnf` using the same scaling as Figure 8(a).) However, we do not believe that the rate of fluctuation may have

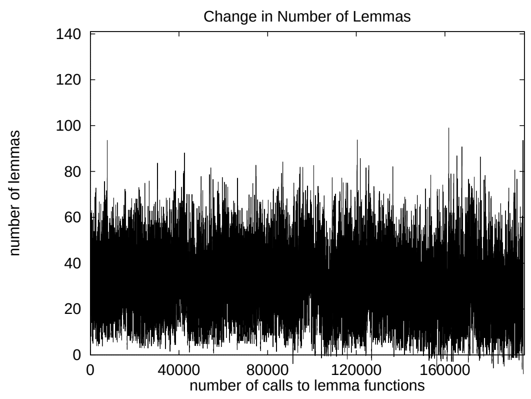
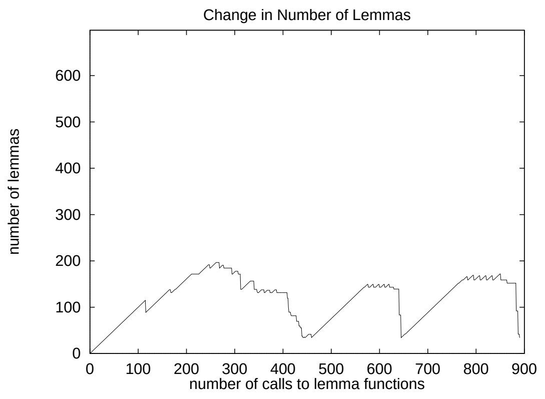
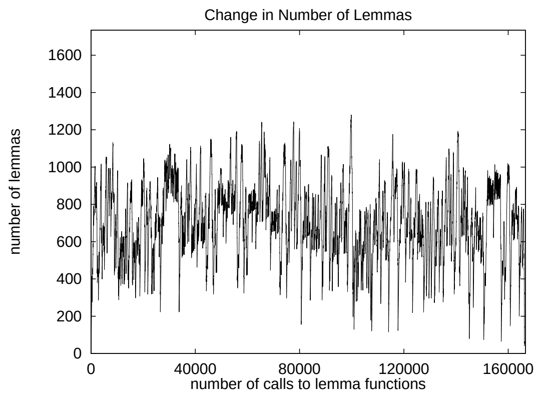


Figure 7: Typical change in the number of lemmas encountered by `modoc` on random formulas. (The graph is from `3.141.602.001-386058.cnf`.)



(a) `bf2670-007.cnf`



(b) `bf2670-501.cnf`

Figure 8: Changes in the number of lemmas encountered by `modoc` on circuit formulas.

an effect on the performance of Modoc. Overall, we believe that lemmas are not a crucial factor in explaining the difference in the performance of Modoc.

5 More Experiment

It was pointed out after running the experiment in Section 3 that using random 3CNF formulas with clauses-to-variables ratio of 4.27 may be "unfair" to the random formulas, as the ratio is believed to generate the most difficult set of formulas ([5, 2]). So to be fair, two more classes of random formulas were generated and tested.

Under-constrained Random Formulas: This class consists of 50 141-variable 482-clause 3CNF random formulas. The clauses-to-variables ratio is 3.416 which is 20% smaller than the ratio used in Section 3. This makes the formulas under-constrained and highly likely to be satisfiable.

Over-constrained Random Formulas: This class consists of 50 141-variable 722-clause 3CNF random formulas. The clauses-to-variables ratio is 5.124, which is 20% larger than the ratio used in Section 3. This makes the formula over-constrained and highly likely to be unsatisfiable.

Both classes of formulas were generated from the same probability model used in Section 3.

All under-constrained random formulas turned out to be satisfiable. Although there were variations in the change in the autarky size, all graphs shared a common characteristic that the "rise" to a satisfying truth assignment was preceded by an optional line at size zero, possibly with some "bumps" on it. (Some of the graphs are shown in Figure 16 in Appendix.)

All over-constrained random formulas turned out to be unsatisfiable. All graphs were very similar to the graph in Figure 3(a), only much more "barren". (A typical graph is shown in Figure 17 in Appendix.)

Although the graphs show a slight difference compared to Figure 3, they continue to show the near lack of autarkies as observed in Section 4. We speculate that autarky pruning is not likely to be successful on any random formulas using any (reasonable) clauses-to-variables ratio.

6 Discussion

So what does all the observation mean? For one, it reconfirms the observation first made by Monien and Speckenmeyer in [6] that autarkies are rare and that trying to partition a formula using a properly partitioning autarky is not likely to be an effective approach. On the other hand, our observation suggests abundance of conditional autarkies in circuit formulas[1]. Is there anything we can do to exploit this characteristic?

The observation reported in this paper may lead to new uses of autarkies in solving the satisfiability problem. However, it does not appear to be a trivial task to incorporate them into a satisfiability testing algorithm. The difficulty arises from having to cope with the partial truth assignments that the conditional autarkies depend on. Suppose we have a formula that has a conditional autarky. We could simplify the formula using the partial truth assignment on which the conditional autarky depends, partition the formula based on the (conditional) autarky, and solve the satisfiability problem on the set of the remaining clauses. If the set is satisfiable, the original formula is satisfiable; but what if it is unsatisfiable? In this case, we don't know about the satisfiability of the original formula. (Or do we know *anything*?) The reduction that took place from the original formula to the set of the remaining clauses was conditional, just like the autarky.

7 Conclusion

Autarky pruning is one of the two pruning techniques used in Modoc (other being lemma pruning). Its effectiveness is hence expected to have a major impact on the success of Modoc. Experimentally, we have seen that Modoc is not able to take advantage of this pruning technique on random formulas. In a way, random formulas defy autarky pruning. This may be explained by their randomness and hence their evenness in their distribution of variables. It will be interesting to do a mathematical analysis on the autarky sizes of random formulas. (What is the probability that a random formula has a properly partitioning autarky whose size is $\geq$1%? $\geq$5%? $\geq$10%?)

[1]It is the author's belief that Monien and Speckenmeyer's observation on the rarity of autarkies was for random formulas. Unfortunately, this is not clear from their paper [6]. If the author's belief is correct, our observation does not contradict with their observation.

We also showed, through experiment, that (unconditional) autarkies are extremely rare, even for circuit formulas for which autarky pruning appears to be effective. However, for circuit formulas, it appears that there is abundance of conditional autarkies. Also, it appears that there are large conditional autarkies with small partial truth assignments on which they depend. These may lead to new ways to use autarkies to solve the satisfiability problem, but currently, we don't know of any. A hypothetical algorithm that "conditionally" reduces the problem to a subformula was described. The algorithm works fine if the subformula is satisfiable, but the question remains on what we can do in case the subformula is unsatisfiable. Are all efforts lost? Can we recover any information about anything to circumvent the complete loss of efforts? Many questions remain to be answered regarding the use of conditional autarkies.

Acknowledgments

The author wishes to thank Allen Van Gelder for valuable discussions on Modoc and for reading earlier drafts of this paper.

References

[1] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, January 1992.

[2] T. Larrabee and Y. Tsuji. Evidence for a satisfiability threshold for random 3CNF formulas. Technical Report UCSC–CRL–92–42, UC Santa Cruz, Santa Cruz, CA., October 1992.

[3] D. W. Loveland. A simplified format for the model elimination theorem-proving procedure. *JACM*, 16(3):349–363, 1969.

[4] J. Minker and G Zanon. An extension to linear resolution with selection function. *Information Processing Letters*, 14(3):191–194, June 1982.

[5] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), San Jose, CA.*, pages 459–465, July 1992.

[6] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[7] A. Van Gelder. Simultaneous construction of refutations and models for propositional formulas. Technical Report UCSC–CRL–95–61, UC Santa Cruz, Santa Cruz, CA., 1995. (submitted for publication).

[8] A. Van Gelder and F. Kamiya. The partial rehabilitation of propositional resolution. Technical Report UCSC–CRL–96–04, UC Santa Cruz, Santa Cruz, CA., 1996.

[9] A. Van Gelder and Y. K. Tsuji. Satisfiability testing with more reasoning and less guessing. In D. S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge.*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.

A More Figures

This section contains figures that could not be placed in the main body of the paper.

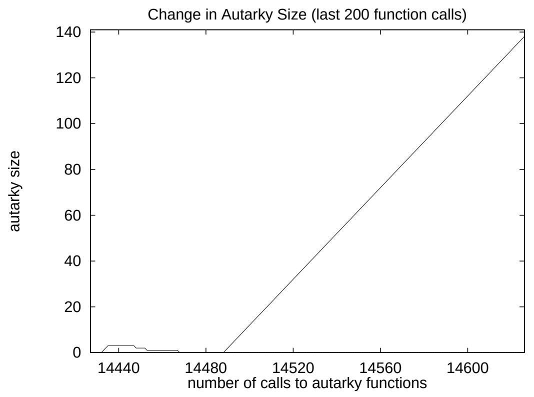


Figure 9: Change in the autarky size during the last 200 calls to autarky functions encountered by `modoc` on a random formula `3.141.602.001-386058.cnf`. This is a close-up of the last 200 function calls in Figure 3(b) on Page 4. We can see that what was drawn as a "sudden surge" was in fact a slope.

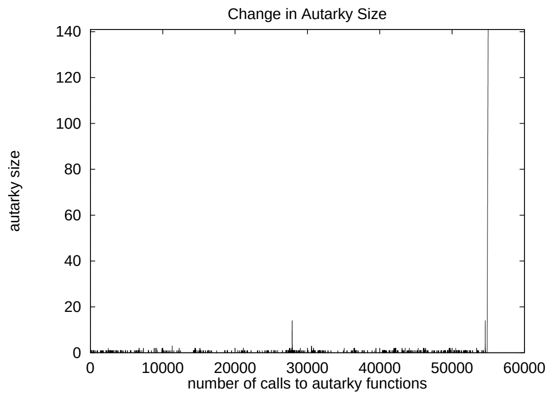


Figure 10: Change in the autarky size encountered by `modoc` on a random formula `3.141.602.040-386097.cnf`. In this graph, two "spikes" can be seen.

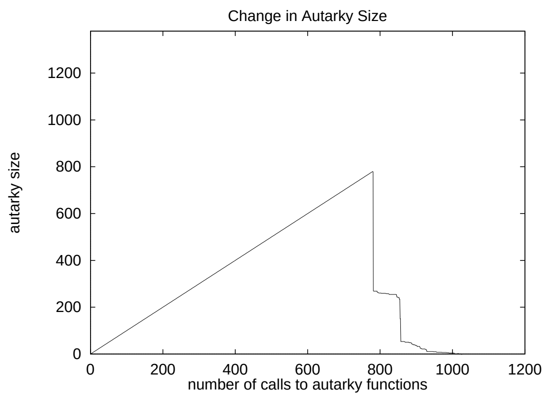


Figure 11: Change in the autarky size encountered by `modoc` on a circuit formula `bf2670-208.cnf`. This graph is an opposite of the graph in Figure 4(b) on Page 5 in that all falls through after two-thirds of the way into search.

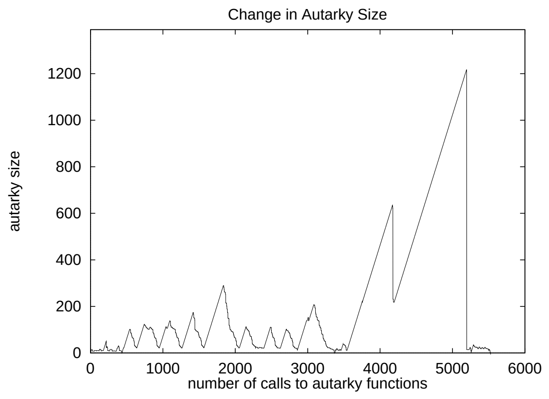
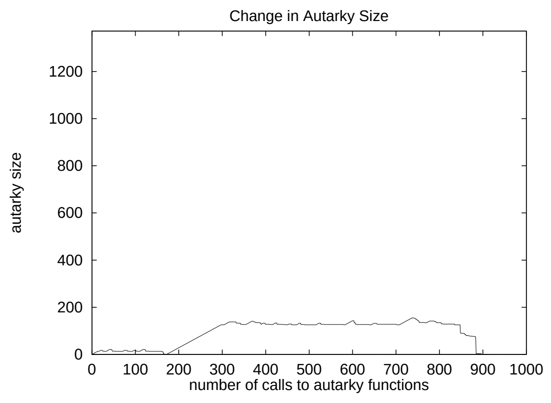
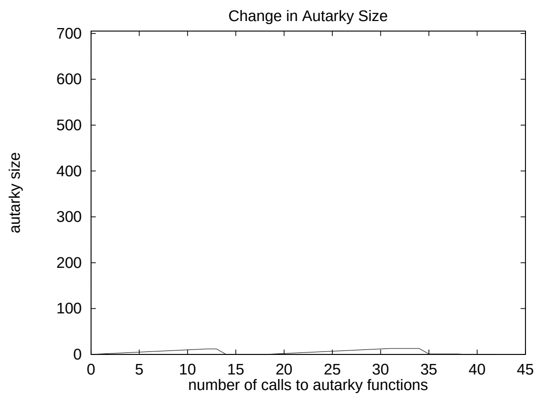


Figure 12: Change in the autarky size encountered by `modoc` on a circuit formula `bf2670-051.cnf`. This graph is similar to the graphs in Figure 5 on Page 5 in that it shows “spires”. However, it is different from them in that it shows “spires” in increasing heights.



(a) `bf2670-494.cnf`



(b) `bf2670-035.cnf`

Figure 13: Changes in the autarky size encountered by `modoc` on circuit formulas. Both graphs show what looks like "plateaus" of autarkies. Figure (b) is an exception among all circuit formulas in that very few autarkies were found.

formula name	order found	autarky size
`3.141.602.014-386071.cnf`	1	131
	2	134
	3	137
`3.141.602.026-386083.cnf`	1	134
	2	137

Figure 14: Sizes of unconditional autarkies found by `modoc`. (Formulas for which the first unconditional autarky found was a satisfying truth assignment are not shown. The last line for each formula is also the satisfying truth assignment found for that formula by `modoc`.)

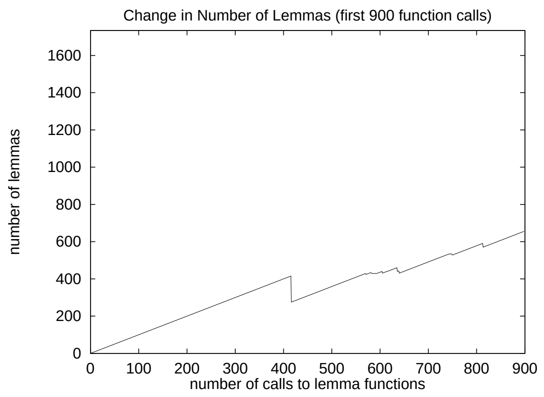
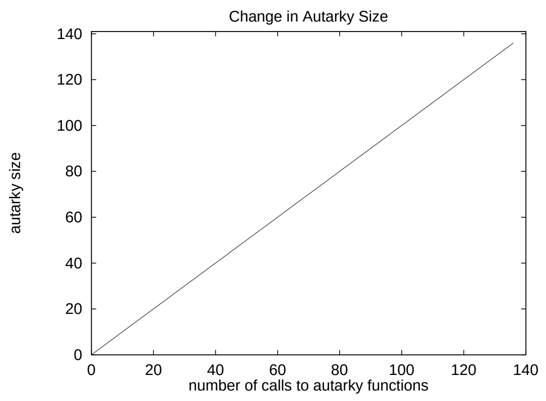
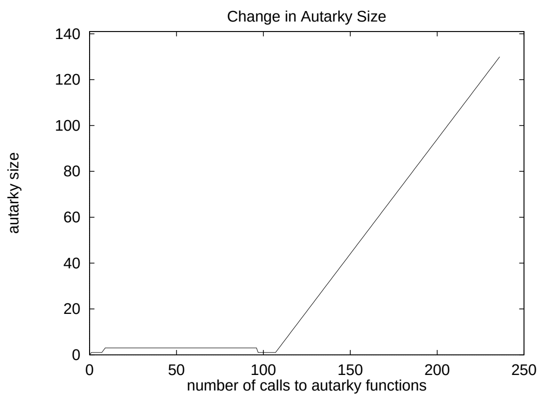


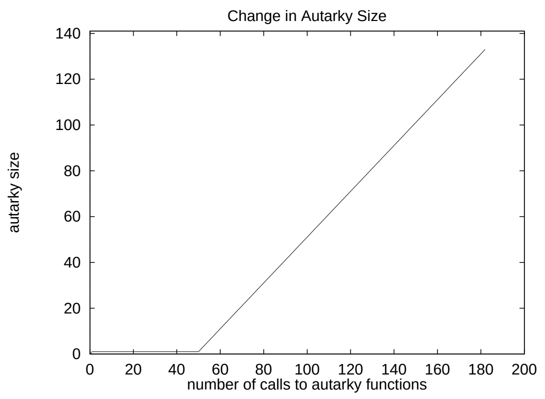
Figure 15: Change in the number of lemmas during the first 900 calls to lemma functions encountered by `modoc` on a circuit formula `bf2670-501.cnf`. This is a close-up of the first 900 function calls in Figure 8(b) on Page 7. Note that the horizontal scaling used in this graph is exactly the same as the graph in Figure 8(a). We can see that the rates of fluctuation of the two formulas are very similar.



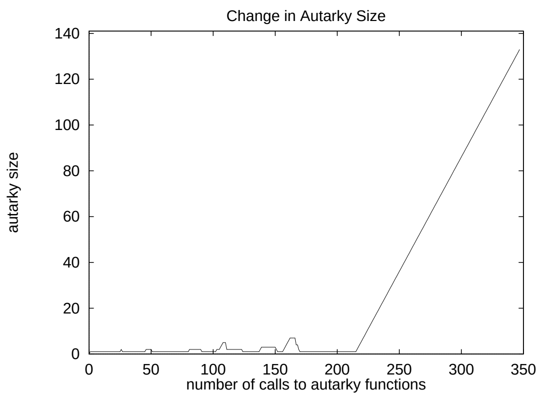
(a) `3.141.482.03-515220.cnf`



(b) `3.141.482.13-515230.cnf`



(c) `3.141.482.16-515233.cnf`



(d) `3.141.482.42-515259.cnf`

Figure 16: Changes in the autarky size encountered by `modoc` on under-constrained random formulas.

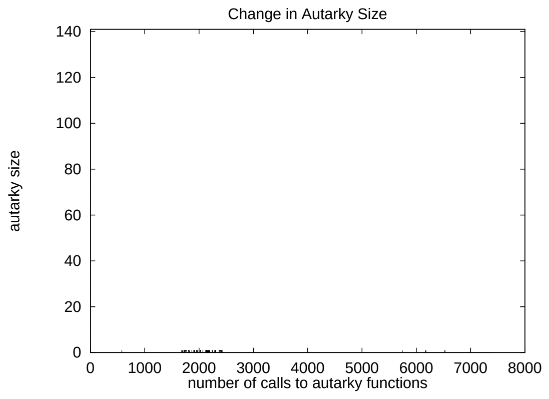


Figure 17: Typical change in autarky size encountered by `modoc` on over-constrained random formulas. (The graph is from `3.141.722.14-256911.cnf`.)