Automated Generation of Simplification Rules for SAT and MAXSAT*

Alexander S. Kulikov

St.Petersburg State University,
Department of Mathematics and Mechanics,
St.Petersburg, Russia
http://logic.pdmi.ras.ru/~kulikov

Abstract. Currently best known upper bounds for many NP-hard problems are obtained by using divide-and-conquer (splitting) algorithms. Roughly speaking, there are two ways of splitting algorithm improvement: a more involved case analysis and an introduction of a new simplification rule. It is clear that case analysis can be executed by computer, so it was considered as a machine task. Recently, several programs for automated case analysis were implemented. However, designing a new simplification rule is usually considered as a human task. In this paper we show that designing simplification rules can also be automated. We present several new (previously unknown) automatically generated simplification rules for the SAT and MAXSAT problems. The new approach allows not only to generate simplification rules, but also to find good splittings. To illustrate our technique we present a new algorithm for $(n,3)$-MAXSAT that uses both splittings and simplification rules based on our approach and has worst-case running time $O(1.2721^N L)$, where N is the number of variables and L is the length of an input formula. This bound improves the previously known bound $O(1.3248^N L)$ of Bansal and Raman.

1 Introduction

The *splitting method* (i.e., estimating the complexity of divide-and-conquer algorithms by recurrent inequalities) is a powerful tool for proving upper bounds for NP-hard problems. Currently best known upper bounds for many NP-hard problems are obtained by using exactly this method (SAT [7], MAXSAT [3], XSAT [2], MAX-CUT [4], Vertex Cover [6]). Formally, a splitting algorithm splits an input instance of a problem into several simpler instances further simplified by certain simplification rules, such that by finding the solution for all of them one can construct the solution for the initial instance in polynomial time.

* Supported in part by grant No. 1 of the 6th RAS contest-expertise of young scientist projects (1999) and Grant #NSh-2203-2003-1 of the President of Russian Federation for Leading Scientific School Support.

F. Bacchus and T. Walsh (Eds.): SAT 2005, LNCS 3569, pp. 430–436, 2005.

In general, there are two ways of splitting algorithm improvement: a more involved case analysis and an introduction of a new simplification rule . Recently, several programs for automated case analysis were implemented ([6], [5], [8]). In this paper we present a procedure for generating simplification rules for the SAT and MAXSAT problems. Our approach is based on defining a partial ordering on assignments to variables of a formula and allows not only to generate simplification rules, but also to find good splittings. By using this approach we construct a new algorithm for $(n, 3)$-MAXSAT with worst-case running time $O(1.2721^N L)$. The automated proof of the running time of this algorithm is available at `http://logic.pdmi.ras.ru/~kulikov`.

Motivation. We have already mentioned that one of the two main ways of splitting algorithm improvement is an introduction of a new simplification rule to it. This fact already motivates the need for designing new simplification rules. Also sometimes a new simplification rule does not allow to prove a better upper bound on the running time of a splitting algorithm, but it allows to significantly reduce the case analysis included in this algorithm. In general, a new simplification rule for, say, the SAT problem can improve the running time of a certain SAT solver, not necessarily based on the splitting method.

2 General Setting

2.1 The SAT and MAXSAT Problems

The SAT problem is: given a formula in CNF, check whether this formula is satisfiable, i.e., whether there exists an assignment of Boolean values to all variables of the formula that makes it *True*. For example, the formula $(xy)(\bar{x})$ is obviously satisfiable, while the formula $(xy)(\bar{x})(x\bar{y})$ is not. In the MAXSAT problem one is asked to find a maximal possible number of simultaneously satisfied clauses of a given formula. For example, this number is equal to 3 for the formula $(xy)(\bar{x}\bar{y})(\bar{x}y)(x\bar{y})$.

Both SAT and MAXSAT are very well-known NP-hard problems, that have been attacked by scientists from all over the world for a long time. The currently best known upper bounds for SAT are $O(1.238823^K)$ and $O(1.073997^L)$ [7], and for MAXSAT are $O(1.341294^K)$ [3], $O(1.105729^L)$ [1], where K denotes the number of clauses in the input formula, L denotes the length of the input formula. Note that there are no known upper bounds of the form $O(c^N)$, where $c < 2$ is a constant and N is the number of variables in the input formula, for these problems.

2.2 Simplification Rules

Throughout this paper we consider only simplification rules for SAT and MAXSAT. In case of SAT by an *optimal assignment* we mean an assignment to all variables of a formula that satisfies this formula, in case of MAXSAT we say

that an assignment is *optimal* if this assignment satisfies the maximal possible number of clauses of a formula. Note that in case of MAXSAT any CNF formula has an optimal assignment, while in case of SAT only satisfiable formulas have such an assignment.

We say that a simplification rule is applicable to a formula F if it can replace F by another formula F' in polynomial time such that both following conditions hold:

1. the complexity of F' is smaller than the complexity of F;
2. by constructing an optimal assignment for F' one can construct an optimal assignment for F in polynomial time.

2.3 Class of Formulas

Let C be a clause consisting of literals $l_1, l_2, \ldots, l_k$. We define a *clause with unfixed length* as a set of clauses that contains all these literals and probably some more literals and denote it by $(l_1 l_2 \ldots l_k \ldots)$. We call the literals $l_1, l_2, \ldots, l_k$ the *basis* of this clause. For example, $(l \ldots)$ is the set of all clauses containing the literal l. In the following we use the word "clause" to refer both to clauses in its standard definition and to clauses with unfixed length.

Similarly we define a *class of formulas*. Let $C'_1, \ldots, C'_k$ be clauses (some of them may have unfixed lengths). Then a *class of formulas* is the set of formulas represented as $F = \{C_1, \ldots, C_m\}$, such that the following conditions hold:

1. $m \geq k$,
2. for $1 \leq i \leq k$, $C'_i \subseteq C_i$ (as sets of literals), if C'_i is a clause with unfixed length, and $C_i = C'_i$ otherwise,
3. for $1 \leq i \leq k$, $k+1 \leq j \leq m$, C_j does not contain any variable from the basis of C'_i.

We denote this set by $C'_1, \ldots, C'_k \ldots$ and call the clauses $C'_1, \ldots, C'_k$ the *basis* of this class of formulas. We define two functions $CorSet$ and $CorClause$ (for corresponding set and corresponding clause) based on this definition:

$$CorSet(F, \mathcal{F}) = \{C_1, \ldots, C_k\} \ ,$$

$$CorClause(C'_i, F, \mathcal{F}) = C_i \ .$$

The notion of class of formulas allows to explain the fact that a formula contains occurrences of certain literals. For example, to show that a formula F contains a $(3,2)$-literal one can write the following:

$$F \in (x \ldots)(x \ldots)(x \ldots)(\bar{x} \ldots)(\bar{x} \ldots) \ldots \ .$$

However, if we want to express the fact that F contains two $(1,1)$-literals that occur together in a clause, we have to write the following:

$$F \in (xy \ldots)(\bar{x} \ldots)(\bar{y} \ldots) \ldots \text{ or } F \in (xy \ldots)(\bar{x}\bar{y} \ldots) \ldots \ .$$

For a formula F and a subset S of its variables we define $CorClass(F, S)$ as a class of formulas resulting from F by replacing all its variables that are not in S by "...". Clearly, $F \in CorClass(F, S)$. For example,

$$CorClass((xyz)(\bar{x})(\bar{y}zu)(\bar{u}x)(\bar{z}), \{z, u\}) = (z\ldots)(zu\ldots)(\bar{u}\ldots)(\bar{z})\ldots \ .$$

Note that in most situations we can work with a class of formulas in the same way as with a CNF formula. For example, if we eliminate from the basis of a class of formulas all clauses that contain a literal x and all occurrences of the literal $\bar{x}$ from the other clauses, we obtain the class of formulas resulting from all formulas of the initial class by setting the value of x to $True$. Also it is easy to see that if after assigning a Boolean value to a variable of a class of formulas or applying a (considered in this paper) simplification rule to it, its complexity measure decreases by Δ, then the complexity measure of each formula of this class decreases *at least* by Δ.

For a class of formulas $\mathcal{F}$ and a clause C we define a class of formulas $\mathcal{F}+\{C\}$ as a class resulting from $\mathcal{F}$ by adding the clause C to its basis. Similarly we define a clause with unfixed length $C + \{l\}$, where C is a clause with unfixed length and l is a literal.

We say that a simplification rule is applicable to a class of formulas, if this rule is applicable to every formula of this class. For example, each formula of the class $(x)(x\bar{y}\ldots)(y\ldots)\ldots$ contains a pure literal (i.e., a literal that does not occur negated in a formula).

3 New Simplification Rules

In this section we present several new simplification rules for the SAT and MAXSAT problems.

Usually to prove the correctness of a simplification rule that assigns the value $True$ to a literal x one proves that every optimal assignment that contains the literal $\bar{x}$ can be reconstructed into an optimal assignment that contains the literal x. For example, if x is a pure literal of a formula F then any assignment $A \ni \bar{x}$ satisfies not more clauses than the assignment $A\backslash\{\bar{x}\} \cup \{x\}$. This discussion motivates the following definition.

Definition 1. *Let A_1 and A_2 be assignments to all known variables of a class of formulas $\mathcal{F}$. We say that A_1 is stronger than A_2 w.r.t. $\mathcal{F}$ and write $A_1 \succ_{\mathcal{F}} A_2$, if for each formula $F \in \mathcal{F}$ and for each assignment B to all variables of F the following condition holds: if $A_2 \subset B$, then B satisfies not more clauses of F than $B\backslash A_2 \cup A_1$.*

For example, in case of MAXSAT problem if $\mathcal{F}_1 = (x\ldots)(x\ldots)\ldots$, then $\{x\} \succ_{\mathcal{F}_1} \{\bar{x}\}$ (due to the pure literal rule), and if $\mathcal{F}_2 = (xy\ldots)(\bar{x})(x\bar{y}\ldots)(\bar{x})(y\ldots)(x\ldots)\ldots$, then $\{\bar{x}\} \succ_{\mathcal{F}_2} \{x\}$ (due to the almost dominating unit clause rule). The given definition is very easy and natural. However, in general, it is

not clear how to check whether one of the given assignments is stronger than the other. Below we show that such a check is possible.

Definition 2. *Let C be a clause and A be an assignment. We define the function $mark(C, A)$ as follows:*

$$mark(C,A) = \begin{cases} \text{“}+\text{”}, & \text{if } C \text{ contains at least one literal from } A; \\ \text{“}-\text{”}, & \text{if } C \text{ has fixed length and for each literal } x \in C,\ \bar{x} \in A; \\ \text{“?”}, & \text{otherwise.} \end{cases}$$

For example, $mark((xyz\ldots),\{y\}) = \text{“}+\text{”}$, $mark((x\bar{y}),\{\bar{x},y\}) = \text{“}-\text{”}$, $mark((x\bar{y}),\{\bar{x},z\}) = \text{“?”}$, $mark((x\bar{y}\ldots),\{\bar{x},y\}) = \text{“?”}$. Informally, $mark(C, A)$ just tell us whether clauses of C are satisfied by extensions of A.

Theorem 1. *Let A_1, A_2 be assignments to all variables of a class of formulas $\mathcal{F}$ and let*

$$p_1 = |\{C \in \mathcal{F} \mid mark(C, A_1) = \text{“}+\text{”}\}|\ ,$$

$$p_2 = |\{C \in \mathcal{F} \mid mark(C, A_2) = \text{“}+\text{”}\}|\ ,$$

$$q = |\{C \in \mathcal{F} \mid mark(C, A_2) = \text{“?”}, mark(C, A_1) \neq \text{“?”}\}|\ .$$

Then, $A_1 \succ_{\mathcal{F}} A_2$ iff $p_1 \geq p_2 + q$.

Let us show how this theorem allows to generate simplification rules. We start with giving several examples. Suppose $F \in \mathcal{F} = (xy)(x\bar{y})(\bar{x}y)(\bar{x}\bar{y}\ldots)\ldots$. In such a case we can replace F by $F[\bar{x},\bar{y}]$, since $\{\bar{x},\bar{y}\} \succ_{\mathcal{F}} \{x,y\},\{\bar{x},y\},\{x,\bar{y}\}$. The assignment $\{\bar{x},\bar{y}\}$ satisfies *at least* three clauses from $CorSet(F,\mathcal{F})$, while any other assignment to variables x and y satisfies *at most* three of these clauses.

Consider a slightly more complicated example. Let $\mathcal{F} = (xy\ldots)(x\ldots)(\bar{x}\bar{y}\ldots)(\bar{y})\ldots$, then $\{\bar{x},y\} \succ_{\mathcal{F}} \{\bar{x},\bar{y}\}$ and $\{x,\bar{y}\} \succ_{\mathcal{F}} \{x,y\}$. It means that for any formula $F \in \mathcal{F}$, if there exists an optimal assignment for F, then there exists an optimal assignment B, such that either $\{\bar{x},y\} \subset B$ or $\{x,\bar{y}\} \subset B$. Thus, we can replace F by $F[y = \bar{x}]$.

In both given examples we construct a *majority set* of assignments to all variables of a class of formulas. Below we formally define this notion.

Definition 3. *Let $\mathcal{F}$ be a class of formulas, M_0 be the set of all possible assignments to all variables of $\mathcal{F}$. We say that $M \subset M_0$ is a* majority set *for $\mathcal{F}$ and write $M = MajorSet(\mathcal{F})$, if for any assignment $A_0 \in M_0$ there exists an assignment $A \in M$, such that $A \succ_{\mathcal{F}} A_0$.*

It is easy to see that a majority set is not unique and that, in particular, M_0 (from the definition) is a majority set for $\mathcal{F}$. However, we are interested in majority sets of small size. We use a greedy algorithm for construction of such majority sets, its code is given in Fig. 1.

Now we are ready to present the new simplification rule that unifies many known simplification rules that modify an input formula by assigning a value to a literal. It is illustrated in Fig. 2. It is easy to see that the running time

Procedure MSC

Input: A class of formulas $\mathcal{F}$.
Output: A majority set for $\mathcal{F}$.
Method.

1. let $M_0 = 0$
2. let M be the set of all possible assignments to all variables of $\mathcal{F}$
3. in case of SAT: remove from M all assignments A, such that $mark(C, A) =$ " $-$ " for some clause $C \in \mathcal{F}$
4. while $M \neq 0$
   (a) let A_0 be an assignment of M such that $|\{A \in M | A_0 \succ_{\mathcal{F}} A\}|$ is maximal
   (b) $M_0 = M_0 \cup \{A_0\}$
   (c) $M = M \backslash \{A \in M | A_0 \succ_{\mathcal{F}} A\}$
5. return M_0

Fig. 1. The greedy algorithm for majority set construction

Procedure GSR(integer c)

Input: A CNF formula F.
Output: A simplified CNF formula F.
Method.
for each subset S of variables of F, such that $|S| \leq c$

1. let $\mathcal{F} = CorClass(F, S)$
2. let $M = MajorSet(\mathcal{F})$
3. if there is a literal x, such that it occurs in every assignment from M, then return $F[x]$
4. if there are literals x and y, such that for any assignment $A \in M$, either $\{x, y\} \subset A$ or $\{\bar{x}, \bar{y}\} \subset A$, then return $F[x = y]$

Fig. 2. The general simplification rule

of this rule is $O(L)$, the correctness is trivial. We call this rule an *automated* procedure for generating simplification rules, as one can add this procedure with any constant parameter into an algorithm. Clearly, if $GSR(c_1)$ can simplify a formula, then $GSR(c_2)$ can simplify this formula too, for $c_1 < c_2$.

Below we give several new simplification rules, that are particular cases of GSR.

New simplification rules for MAXSAT:

1. if $F \in \mathcal{F} = (x)(\bar{x}y\ldots)(\bar{x}\bar{y}\ldots)(\bar{x}\bar{y}\ldots)(\bar{y}\ldots)(\bar{y}\ldots)\ldots$, then replace F by $F[x]$ (since $\{\{x, y\}, \{x, \bar{y}\}\} = MajorSet(\mathcal{F})$)
2. if $F \in \mathcal{F} = (xyz\ldots)(\bar{x}z\ldots)(\bar{x}z\ldots)(y\ldots)(\bar{y}\bar{z}\ldots)(z\ldots)\ldots$, then replace F by $F[x = y]$ (since $\{\{\bar{x}, y, \bar{z}\}, \{x, \bar{y}, z\}, \{x, y, z\}\} = MajorSet(\mathcal{F})$)

3. if $F \in \mathcal{F} = (x)(\bar{x}yz\ldots)(\bar{x}\bar{t}\ldots)(\bar{x}\bar{t}\ldots)(yt\ldots)(\bar{y}\bar{t}\ldots)(\bar{y}\ldots)(z\bar{t}\ldots)(\bar{z}\ldots)(\bar{z}\ldots)\ldots$, then replace F by $F[x]$ (since $\{\{x,\bar{y},\bar{t}\},\{x,y,\bar{t}\},\{x,\bar{y},t\}\} = MajorSet(\mathcal{F})$)

New simplification rules for SAT:

1. if $F \in \mathcal{F} = (xy)(\bar{x}\ldots)(\bar{y}\ldots)(\bar{x}y\ldots)(\bar{y}\ldots)\ldots$, then replace F by $F[x = \bar{y}]$ (since $\{\{x,\bar{y}\},\{\bar{x},y\}\} = MajorSet(\mathcal{F})$)
2. if $F \in \mathcal{F} = (xy\ldots)(xz\ldots)(\bar{x}\bar{z}\ldots)(\bar{x}\bar{z}\ldots)(\bar{x}\bar{z}\ldots)(y\ldots)(y\ldots)(\bar{y}z\ldots)(\bar{y}\ldots)(\bar{z}\ldots)\ldots$, then replace F by $F[x = \bar{z}]$ (since $\{\{x,\bar{y},\bar{z}\},\{x,y,\bar{z}\},\{x,\bar{y},\bar{z}\}\} = MajorSet(\mathcal{F})$)

Actually our technique allows to find not only simplifications, but also good splittings. The main idea is that for any formula $F \in \mathcal{F}$ the splitting $F[A_1]$, $F[A_2],\ldots,F[A_k]$ is correct, where $\{A_1,A_2,\ldots,A_k\} = MajorSet(\mathcal{F})$.

Acknowledgements

The author would like to express a great gratitude to his supervisor Edward A. Hirsch for valuable comments.

References

1. N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of ISAAC'99*, pages 247–258, 1999.
2. J. M. Byskov, B. A. Madsen, and B. Skjernaa. New algorithms for exact satisfiability. Technical Report RS-03-30, BRICS, 2003.
3. J. Chen and I. Kanj. Improved exact algorithms for MAX-SAT. In *Proceedings of the 5th LATIN*, volume 2286 of *LNCS*, pages 341–355, 2002.
4. S. S. Fedin and A. S. Kulikov. A $2^{|E|/4}$-time algorithm for MAX-CUT. *Zapiski nauchnykh seminarov POMI*, 293:129–138, 2002.
5. S. S. Fedin and A. S. Kulikov. Automated proofs of upper bounds on the running time of splitting algorithms. *Zapiski nauchnykh seminarov POMI*, 316:111–128, 2004.
6. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004.
7. E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.
8. S. I. Nikolenko and A. V. Sirotkin. Worst-case upper bounds for sat: automated proof. In *Proceedings of the 8th ESSLLI Student Session*, pages 225–232, 2003.