Solving SAT by Algorithm Transform of Wu's Method

HE Simin (贺思敏) and ZHANG Bo (张钹)

Department of Computer Science and Technology

National Key Laboratory of Intelligent Technology and Systems

Tsinghua University, Beijing 100084, P. R. China

Received July 6, 1999

Abstract Recently algorithms for solving propositional satisfiability problem, or SAT, have aroused great interest, and more attention is paid to transformation problem solving. The commonly used transformation is representation transform, but since its intermediate computing procedure is a black box from the viewpoint of the original problem, this approach has many limitations. In this paper, a new approach called algorithm transform is proposed and applied to solving SAT by Wu's method, a general algorithm for solving polynomial equations. By establishing the correspondence between the primitive operation in Wu's method and clause resolution in SAT, it is shown that Wu's method, when used for solving SAT, is primarily a restricted clause resolution procedure. While Wu's method introduces entirely new concepts, *e.g.*, characteristic set of clauses, to resolution procedure, the complexity result of resolution procedure suggests an exponential lower bound to Wu's method for solving general polynomial equations. Moreover, this algorithm transform can help achieve a more efficient implementation of Wu's method since it can avoid the complex manipulation of polynomials and can make the best use of domain specific knowledge.

Keywords algorithm design, satisfiability problem, Wu's method, automated reasoning

1 Introduction

In recent years, algorithms for solving the propositional satisfiability problem, or SAT, have attracted the attention of many researchers. Since SAT originated from automated theorem proving (ATP), we can directly use ATP methods to solve SAT. There are mainly two such algorithms: one is DP procedure [1], whose refinements are the most efficient complete algorithms at present; the other is resolution procedure [2], which has beautiful properties in ATP of first-order predicate logic and can, of course, be used to solve satisfiability problem in propositional logic. In particular, resolution operation has deep relations with almost all algorithms for SAT.

Another approach to solving SAT is transformation problem solving, that is, by appropriate transform, we can make use of some relatively mature methods, techniques or strategies developed from other problem domains to solve the current problem indirectly. Transformation problem solving can provide us with wider field of vision to reveal the essence of the problem and the intrinsic connections between various problems and between their solving methods. Currently, transformation algorithms for SAT are mainly related to the following four classes of problem solving methods: discrete local search, linear and integer programming, techniques for constraint satisfaction problems, and continuous optimization algorithms.

The common way of transformation is representation transform, which is to convert the representation of the original problem to the form of a new problem, then use the algorithm for the new problem to get the result, and finally convert the result back to the form of the original problem. Although the inputs and the outputs under the two problem representations correspond with each other, respectively, the intermediate computing procedure under the

new problem representation can not be understood in terms of concepts and operations under the original problem representation. In short, representation transform is not readable.

Another way of transformation is algorithm transform, which we highly advocate in this paper. Algorithm transform is to rewrite an algorithm for a new problem with concepts and operations under the original problem representation and produce a new algorithm to solve the original problem. Although we are still using methods from new problem domains to solve the original problem, the intermediate computing procedure is no longer a black box: we are clear about every step.

Compared with representation transform, algorithm transform has its obvious superiority. First, representation transform often produces a huge and inefficient transformed input representation. Under the new problem representation, many implicit properties of the original problem must be explicitly expressed now, while some important features of the original problem can not be neatly and completely expressed. However, there is no such burden in algorithm transform. Second, representation transform can not closely combine various methods from different problem domains to solve one problem, since different methods originally depend heavily on the corresponding different representations. Algorithm transform, however, can produce efficient hybrid algorithms because all transformed methods can be expressed under the common problem representation. This is a very important advantage since we believe that no single method can efficiently solve so difficult a problem like SAT. Finally, algorithm transform can help reveal the intrinsic connections between problems and between their solving methods, but representation transform helps little.

In this paper, we will discuss how to solve SAT by algorithm transform of Wu's method. Wu's method [3] is a general method to solve polynomial equations, and its great success in mechanical theorem proving in plane geometry inspires us to apply it to solving SAT. Although it is novel to use Wu's method to solve SAT, what we want to emphasize in this paper is the importance of algorithm transform. In Section 2, we briefly introduce Wu's method. In Section 3, we show step by step how to use Wu's method to solve SAT, especially how algorithm transform is established, and how we benefit from algorithm transform both in efficiency of implementation and in depth of understanding of Wu's method. In Section 4, we give detailed experimental results, including the performance of Wu's method and its comparison with those of resolution and DP procedures. Some discussions and concluding remarks are given in Section 5.

2 Introduction to Wu's Method

In this section, we will introduce some main concepts and theorems of Wu's method, which may be cited in later sections [3, 4].

2.1 Basic Concepts

For any number field K and any set of variables $x_1, x_2,\ldots, x_n$, $K[x_1, x_2,\ldots, x_n]$ is a polynomial ring over K and is denoted by $K[x]$, K^n is the n-dimensional linear space over K. Let $PS = \{P_1, P_2,\ldots, P_m\}$ be any set of polynomials in $K[x]$, let x^0 be any point in K^n, if $P_j(x^0) = 0$ for all $j = 1, 2,\ldots, m$, then x^0 is called a solution to the system of polynomial equations $PS = 0$. We also call x^0 a *zero* of the set of polynomials PS, and the zero set of PS is denoted by $Zero(PS)$. Solving the system of polynomial equations $PS = 0$ is in fact equivalent to determining $Zero(PS)$.

2.2 Variable Ordering

The first step of Wu's method is variable ordering, which makes great impact on the performance of Wu's method. However, no general rule has been found to determine a good variable order. Therefore, we often determine a variable order heuristically with the help of domain specific experience.

After variable ordering, we define the *leading variable of a polynomial P* as the highest variable that actually appears in P. If x_i is the leading variable of P, then P can be written as

$$P = I \cdot x_i^{d_i} + \text{lower terms with respect to } x_i$$

where d_i is the highest degree of x_i in P and is written as $d_i = \deg_{x_i}(P)$, and I is the coefficient of the leading term and is called the *initial* of P. A polynomial Q is said to be *reduced* with respect to P if $\deg_{x_i}(Q) < \deg_{x_i}(P)$.

We introduce an order relation '≤' among the polynomials: $P \le Q$ if and only if (1) the leading variable of P is lower than that of Q, or (2) P and Q have the same leading variable x_i, but $\deg_{x_i}(P) \le \deg_{x_i}(Q)$. We further define $P \cong Q$ if and only if $P \le Q$ and $Q \le P$, and define $P < Q$ if and only if $P \le Q$ but $P \cong Q$ does not hold.

2.3 Pseudo-Division between Two Polynomials

Pseudo-division between two polynomials is the primitive operation of Wu's method. Given two polynomials P and Q, let x_i and I be the leading variable and the initial of P respectively, generally we have

$$I^s \cdot Q = H \cdot P + R, \quad \deg_{x_i}(R) < \deg_{x_i}(P)$$

where s is the least possible non-negative integer, and R is called the *remainder* of Q with respect to P and is written as $Rem(Q/P)$. We further have

$$Zero(P, Q) - Zero(I) = Zero(P, R) - Zero(I)$$

2.4 Ascending Set and the Remainder Formula

Ascending set is the basic concept of Wu's method.

Definition 1 (Ascending Set). *A set of polynomials $AS = \{A_1, A_2, \ldots, A_r\}$ is called an ascending set if it satisfies* (1) *AS is of triangular form, that is, for any i, j, $1 \le i < j \le r$, the leading variable of A_i is lower than that of A_j, and* (2) *for any A_i and A_j, if $i < j$, then A_j is reduced with respect to A_i. If $r = 1$ and A_1 is a non-zero constant in K, then the ascending set in this case is called a trivial contradictory ascending set.*

A polynomial is called *reduced with respect to an ascending set* if this polynomial is reduced with respect to every polynomial in the ascending set.

Let P be any polynomial, by successively pseudo-dividing P by A_i in reverse order from r to 1, we get the remainder formula of the form

$$J \cdot P = \sum_i H_i \cdot A_i + R$$

where J is a power-product of initials I_i of A_i, and H_i and R are polynomials with R reduced with respect to AS. We can choose exponents of I_i in J to be as small as possible. Then R is uniquely determined and will be called the *remainder of P with respect to AS*, and is denoted by $R = Rem(P/AS)$.

2.5 Characteristic Set and Zero Theorems

Characteristic set is the soul concept of Wu's method.

Definition 2 (Characteristic Set). *An ascending set $CS = \{C_1, C_2, \ldots, C_r\}$ is called a characteristic set of a set of polynomials $PS = \{P_1, P_2, \ldots, P_m\}$ if CS satisfies*
(1) *for all $j = 1, 2, \ldots, m$, $Rem(P_j / CS) = 0$, and*
(2) $Zero(PS) \subseteq Zero(CS)$.

Lemma 1. *Given a set of polynomials $PS = \{P_1, P_2, \ldots, P_m\}$, there is an algorithmic procedure that can, within finite steps, produce a characteristic set of PS, or prove that there is no solution to $PS = 0$.*

Often there are two strategies to compute the characteristic set, depth-first and breadth-first, and the former is generally more efficient [4].

Lemma 2. *Given a set of polynomials PS = {P_1, P_2,…, P_m} and a characteristic set CS of PS, CS = {C_1, C_2,…, C_r}, let I_i be the initial of C_i for each i = 1, 2,…, r and I be the product of all these I_i, then the zero set of PS has the following structure*

$$Zero(PS) = Zero(CS / I) + \sum_i Zero(PS, I_i)$$

where $Zero(CS / I) = Zero(CS) - Zero(I)$.

Lemma 3. *There is an algorithmic procedure that permits to give for any polynomial set PS a finite decomposition of the form*

$$Zero(PS) = \sum_i Zero(CS_i / J_i)$$

where CS_i is an ascending set and J_i is the product of all initials of polynomials in CS_i.

3 Algorithm transform of Wu's Method

In this section, we will explicate every essential step for solving SAT by Wu's method. Please refer to [5] for detailed proofs. First we give the formal definition of SAT problem in Conjunctive Normal Form (CNF).

Definition 3 (The Propositional Satisfiability Problem). *Let x_1, x_2,…, x_n be n Boolean variables* and *C_1, C_2,…, C_m be m clauses. For each variable x_i, x_i and its negation $\overline{x}_i$ are called complement literals with each other with respect to x_i, and are denoted by $\tilde{x}_i$. For each clause $C_j = \tilde{x}_{j_1} \vee \tilde{x}_{j_2} \vee \Lambda \vee \tilde{x}_{j_{k_j}}$, k_j is the number of literals in C_j, and all literals in C_j correspond to different variables. The question is: is there any consistent assignment of truth value* 0 *or* 1 *to all the n variables that can make all the m clauses be evaluated true? In other words, is the set of clauses satisfiable?*

Example. Given five Boolean variables x_1, x_2, x_3, x_4, x_5, and a set of seven clauses:

$C_1 = \overline{x}_3 \vee \overline{x}_4 \vee x_5$, $C_2 = \overline{x}_1 \vee \overline{x}_2 \vee x_3$, $C_3 = \overline{x}_1 \vee x_3 \vee \overline{x}_4$,

$C_4 = x_1 \vee \overline{x}_2 \vee \overline{x}_4$, $C_5 = x_2 \vee x_3 \vee \overline{x}_5$, $C_6 = x_1 \vee \overline{x}_3 \vee \overline{x}_5$, $C_7 = x_1 \vee \overline{x}_2 \vee x_4$,

is this set of clauses satisfiable?

This example will be used for illustration throughout this section.

3.1 Representation Transform

Representation transform is a natural way to use Wu's method to solve SAT, which requires rewriting SAT in polynomial terms and establishing correspondence between solutions under two forms of representations. Appropriate representation transform is also a springboard to algorithm transform of Wu's method.

One way to accomplish representation transform is to make use of the equivalence between Boolean algebra and Boolean ring with unit element. Starting from general well-formed formula (wff) representation of SAT, first use Boolean *exclusive-or*, *and* and constants 1, 0 to express all Boolean connectives, then take *exclusive-or* as '+', *and* as '×'. Thus each wff is converted to a polynomial in $B[x_1, x_2,..., x_n]$ satisfying $x_i^2 = x_i$, where $B=(\{0,1\},+,\times)$ is a Boolean ring. Solution correspondence is set up accordingly. This transform is adopted in [6], where Groebner basis method instead of Wu's method is used to solve the system of polynomial equations.

However, we will not adopt this representation transform because it would destroy the further possibility of establishing algorithm transform and lose any benefit from algorithm transform, which will be explained later. Instead, we start from CNF or clause form representation of SAT and adopt a natural representation transform *f* defined below.

Definition 4. *Given a SAT instance of n Boolean variables x_i and m clauses $C_j = \tilde{x}_{j_1} \vee \tilde{x}_{j_2} \vee \Lambda \vee \tilde{x}_{j_{k_j}}$, a transform f is defined as follows*:

(1) $f(1) = 0$, $f(0) = 1$

(2) $f(\tilde{x}_i) = \begin{cases} 1 - x_i, & \text{if } \tilde{x}_i = x_i \\ x_i, & \text{if } \tilde{x}_i = \bar{x}_i \end{cases}$

(3) $f(\tilde{x}_{j_1} \vee \tilde{x}_{j_2} \vee \Lambda \vee \tilde{x}_{j_{k_j}}) = f(\tilde{x}_{j_1}) f(\tilde{x}_{j_2}) \Lambda\ f(\tilde{x}_{j_{k_j}})$

(4) *clause set* $\{C_1, C_2, \Lambda, C_m\}$ *corresponds to a set of Boolean polynomials PS*

$$PS = \{ f(C_j) \mid j = 1, 2, \ldots, m\} \cup \{x_i(1-x_i) \mid i = 1, 2, \ldots, n\}$$

where $f(C_j)$ *is called a clause polynomial, and* $x_i(1-x_i)$ *is called a constraint polynomial.*

Theorem 1 [5]**.** *There is a bijection between satisfying truth assignments of clause set* $\{C_1, C_2, \Lambda, C_m\}$ *and zeros of the set of polynomials PS.*

Example (*continued*). The polynomials corresponding to the seven clauses are:
$f(C_1) = x_3\, x_4\, (1-x_5)$, $f(C_2) = x_1\, x_2\, (1-x_3)$, $f(C_3) = x_1\, (1-x_3)\, x_4$, $f(C_4) = (1-x_1)\, x_2\, x_4$,
$f(C_5) = (1-x_2)(1-x_3)\, x_5$, $f(C_6) = (1-x_1)\, x_3\, x_5$, $f(C_7)=(1-x_1)\, x_2\, (1-x_4)$,
$x_1\, (1-x_1)$, $x_2\, (1-x_2)$, $x_3\, (1-x_3)$, $x_4\, (1-x_4)$, $x_5\, (1-x_5)$.

Actually, we will not directly use Wu's method to determine *Zero*(*PS*), which is a trivial idea. The above mentioned representation transform is only used to induce algorithm transform and will be discarded after that.

3.2 Algorithm transform

The essential step of our algorithm transform of Wu's method is establishing the correspondence between the primitive operation in Wu's method and clause resolution in SAT. Pseudo-division is the primitive operation of Wu's method, and from the following theorem we can discover what is special about it when solving SAT.

Theorem 2 [5]**.** *Let C and C′ be two clauses, and* x_i *be the leading variable of* $f(C)$*. Let* $x_k(1-x_k) = 0$ *for all* $k = 1, 2, \ldots, n$ *be simplification rules, then Rem*($f(C')$ / $f(C)$) *can only have four cases*:

(1) *if C′ has no literal with respect to* x_i ,
$Rem(f(C') / f(C)) = f(C')$;

(2) *if C and C′ have same literals with respect to* x_i ,
$Rem(f(C') / f(C)) = 0$;

(3) *if C and C′ have complement literals between them both with respect to* x_i *and with respect to some other variable,*
$Rem(f(C') / f(C)) = 0$;

(4) *if C and C′ have complement literals between them only with respect to* x_i ,
$Rem(f(C') / f(C)) = f(Res(C, C', x_i))$,
where $Res(C, C', x_i)$ *is the resolvent of C and C′ with respect to* x_i.

Example (*continued*). Assume the variable order is $x_1 < x_2 < x_3 < x_4 < x_5$, then we have

$Rem(f(C_1) / f(C_2)) = x_1x_2x_4(1-x_5) = Rem(f(Res(C_1, C_2, x_3)))$, <case 4>
$Rem(f(C_1) / f(C_3)) = 0$, <case 2>
$Rem(f(C_1) / f(C_5)) = 0$, <case 3>
$Rem(f(C_2) / f(C_3)) = f(C_2)$. <case 1>

From Theorem 2, we know that when using Wu's method to solve SAT, the remainder of pseudo-division between clause polynomials still corresponds to a clause, and furthermore, only in case 4 can we get something new, which corresponds to nothing but the clause resolvent! Therefore, the whole computing procedure corresponds to a clause resolution procedure, and to be precise, a restricted resolution procedure because resolution is only done with respect to some leading variable, not wherever there is a complement pair of literals!

Therefore, when solving SAT, the computing procedure of Wu's method can now be represented by our familiar language of clause and clause resolution, and it is in this sense that we say algorithm transform has been established. Now polynomial language is no longer needed, concepts and operations defined on clause polynomials can now be defined directly on clause, for example, *leading variable of clause, initial of clause, ascending set of clauses, characteristic set of clauses, and pseudo-division between clauses*. All of these bring entirely new ideas to SAT problem solving.

What can we benefit from this algorithm transform?

First, we now obtain a new strategy to control resolution procedure in propositional logic. Resolution procedure, though simple in principle, has great uncertainty in implementation: even using control strategies like unit-preference and set-of-support can not avoid producing a lot of clauses contributing nothing to proof, which greatly decreases the practical efficiency [7]. Although Wu's method in clause form is also a resolution procedure, this resolution is rather focused: resolution is only done with respect to the leading variable of clause, which is similar to semantic resolution [8]. What's more important, the computing procedure of Wu's method is concentrating on computation of characteristic set, which is completely new to resolution principle and is more effective!

Second, by algorithm transform, not only does resolution procedure benefit from Wu's method, Wu's method also benefits from resolution procedure. Haken [9] proves that any resolution proof of pigeonhole problem must have at least exponential number of different clauses, which means an exponential lower bound to resolution proof of SAT problem. Wu's method is primarily a resolution procedure when solving SAT, especially on unsatisfiable instances like pigeonhole problem, only resolution operation (case 4 of Theorem 2) can help produce a contradiction, therefore, in the worst case, Wu's method may have the same fate as general resolution. Theorems 1 and 2 combined with [9] probably give an exponential lower bound to Wu's method when solving general polynomial equations, which is really a non-trivial result. We do not state this proposition strictly because the problem decomposition step of Wu's method should be dealt with specially, although we do not think it will influence the result of complexity. We only want to indicate that algorithm transform can really deepen our understanding of Wu's method while representation transform can not.

Finally, algorithm transform not only helps understand but also helps implement Wu's method in computer for solving SAT. All clauses or clause polynomials can be neatly represented by one-dimensional array where array length is equal to the number of variables. Positive, negative and no occurrence of the related variable can be represented by 1, –1 and 0 respectively. And clause resolution or clause pseudo-division is also easy to be implemented on this data structure. Furthermore, we can make use of our experience of solving SAT to help adapt Wu's method to SAT, which will be explained later. It goes without saying that implementation is crucial to evaluation of a method. If we had used polynomial representation of clauses, which would be inevitable in representation transform, we could not imagine how complex and inefficient it would be, let alone make use of the past experience!

Now we can use clause language to represent and implement Wu's method in order to solve SAT. However, we must adhere to the main steps of Wu's method because it is an entirely new resolution procedure, and its correctness and termination depend on the property of general Wu's method. Next we explain how to adapt some major steps of Wu's method to SAT, including characteristic set computing, problem decomposition, and variable ordering, which will help us further grasp algorithm transform.

3.3 Characteristic Set Computing

Characteristic set is the kernel concept of Wu's method, whose computation is the main procedure. We will explain what is special about characteristic set computing, characteristic set form, and determination of zero set of characteristic set on SAT problem.

We use depth-first strategy [4] to compute characteristic set of polynomial set *PS* defined in Subsection 3.1, but add some modifications. We know that *PS* consists of clause polynomials

$f(C_j)$ for all $j = 1, 2, \Lambda, m$ and constraint polynomials $x_i(1-x_i)$ for all $i = 1, 2, \Lambda, n$. When computing characteristic set, we only do pseudo-division between clause polynomials according to Theorem 2; constraint polynomials will only work implicitly as simplification rules in Theorem 2. We have the following theorem.

Theorem 3 [5]**.** *For computing the characteristic set of PS, the general procedure with above modifications will terminate. If any contradiction is produced, then PS has no solution. Otherwise, we will get an ascending set of clause polynomials* $AS = \{ f(C_k') \mid k = 1, 2, \ldots, r\}$ $(1 \le r \le n)$ *such that every clause polynomial* $f(C_j)$ *of PS has remainder* 0 *with respect to AS according to Theorem* 2. *Let* $CS = AS \cup \{x_t(1-x_t) \mid 1 \le t \le n$, x_t is not a leading variable of any polynomial in $AS\}$, *then CS is a characteristic set of PS.*

Example (*continued*). Here we give the computing procedures of a characteristic set of *PS* under two different variable orders.

Case 1: the variable order is $x_1 < x_3 < x_2 < x_4 < x_5$, under which the clause polynomials can be ordered as $f(C_2) < f(C_3) \cong f(C_4) \cong f(C_7) < f(C_1) \cong f(C_5) \cong f(C_6)$. Let *AS* denote an ascending set.

(1) Initially, add $f(C_2)$ to *AS* for initialization, and $AS = \{ x_1(1-x_3)x_2 \}$;
(2) Compute $Rem(f(C_3) / AS)$, and add the remainder $x_1(1-x_3)x_4$ to *AS*, no further reduction is needed. Now $AS = \{ x_1(1-x_3)x_4, x_1(1-x_3)x_2 \}$;
(3) Compute $Rem(f(C_4) / AS)$, which is 0;
(4) Compute $Rem(f(C_7) / AS)$, which is 0;
(5) Compute $Rem(f(C_1) / AS)$, which is 0;
(6) Compute $Rem(f(C_5) / AS)$, and add the remainder $x_1(1-x_3)x_5$ to *AS*, no further reduction. Now $AS = \{ x_1(1-x_3)x_5, x_1(1-x_3)x_4, x_1(1-x_3)x_2 \}$;
(7) Now compute $Rem(f(C_j) / AS)$ for all $j = 1, 2, \ldots, 7$, and they are all 0;
(8) A characteristic set *CS* of *PS* is $AS \cup \{ x_3(1-x_3), x_1(1-x_1) \}$. That is, under the variable order $x_1 < x_3 < x_2 < x_4 < x_5$, a characteristic set of *PS* is $CS = \{ x_1(1-x_3)x_5, x_1(1-x_3)x_4, x_1(1-x_3)x_2, x_3(1-x_3), x_1(1-x_1) \}$.

Case 2: the variable order is $x_1 < x_2 < x_3 < x_4 < x_5$, under which the clause polynomials can be ordered as $f(C_2) < f(C_3) \cong f(C_4) \cong f(C_7) < f(C_1) \cong f(C_5) \cong f(C_6)$. Let *AS* denote an ascending set.

(1) Initially, add $f(C_2)$ to *AS* for initialization, and $AS = \{ x_1x_2(1-x_3) \}$;
(2) Compute $Rem(f(C_3) / AS)$, which is 0;
(3) Compute $Rem(f(C_4) / AS)$, and add the remainder $(1-x_1)x_2x_4$ to *AS*, no further reduction is needed. Now $AS = \{ (1-x_1)x_2x_4, x_1x_2(1-x_3) \}$;
(4) Compute $Rem(f(C_7) / AS)$, and add the remainder $(1-x_1)x_2$ to *AS*, then reduce all other clause polynomials in *AS* with respect to $(1-x_1)x_2$. Now $AS = \{ (1-x_1)x_2 \}$;
(5) Compute $Rem(f(C_1) / AS)$, and add the remainder $x_3x_4(1-x_5)$ to *AS*, no further reduction. Now $AS = \{ x_3x_4(1-x_5), (1-x_1)x_2 \}$;
(6) Compute $Rem(f(C_5) / AS)$, which is 0;
(7) Compute $Rem(f(C_6) / AS)$, and add the remainder $(1-x_1)x_3x_4$ to *AS*, then $x_3x_4(1-x_5)$ in *AS* will be reduced to 0 with respect to $(1-x_1)x_3x_4$. Now $AS = \{ (1-x_1)x_3x_4, (1-x_1)x_2 \}$;
(8) Compute $Rem(f(C_2) / AS)$, which is 0;
(9) Compute $Rem(f(C_3) / AS)$, which is 0;
(10) Compute $Rem(f(C_4) / AS)$, which is 0;
(11) Compute $Rem(f(C_7) / AS)$, which is 0;
(12) Compute $Rem(f(C_1) / AS)$, which is 0;
(13) Compute $Rem(f(C_5) / AS)$, and add the remainder $(1-x_1)(1-x_3)x_5$ to *AS*, no further reduction. $AS = \{ (1-x_1)(1-x_3)x_5, (1-x_1)x_3x_4, (1-x_1)x_2 \}$;
(14) Now compute $Rem(f(C_j) / AS)$ for all $j = 1, 2, \ldots, 7$, and they are all 0;
(15) A characteristic set *CS* of *PS* is $AS \cup \{ x_3(1-x_3), x_1(1-x_1) \}$. That is, under the variable order $x_1 < x_2 < x_3 < x_4 < x_5$, a characteristic set of *PS* is $CS=\{(1-x_1)(1-x_3)x_5, (1-x_1)x_3x_4, x_3(1-x_3), (1-x_1)x_2, x_1(1-x_1)\}$.

From above, it is shown that under different variable order, the complexity of characteristic set computing, and the characteristic set itself, can be quite different.

Theorem 3 assures us that the above modifications, while improving efficiency, will not destroy termination or correctness of characteristic set computing in original sense of Wu's method. Thus, the characteristic set computing procedure becomes completely a clause resolution procedure.

Let I be the product of initials of polynomials in the characteristic set CS. The following theorem tells us that it is trivial to decide $Zero(CS/I)$.

Theorem 4 [5]**.** *$Zero(CS/I) \neq \varnothing$ if and only if there is no complement pair of literals between the clauses corresponding to the initials of CS. If $Zero(CS/I) \neq \varnothing$, the element of $Zero(CS/I)$ can thus be determined: literals corresponding to the leading variables of clause polynomials in CS should be true, literals contained in I should be false, the other variables can have either value.*

We call a variable *conflicting* if there is a complement pair of literals with respect to this variable between initials of CS. Obviously Theorem 4 means $Zero(CS/I) \neq \varnothing$ if and only if there is no conflicting variable in CS.

Example (*continued*). Under the variable order $x_1 < x_3 < x_2 < x_4 < x_5$, a characteristic set of PS is $CS = \{x_1(1-x_3)x_5,\ x_1(1-x_3)x_4,\ x_1(1-x_3)\ x_2,\ x_3(1-x_3),\ x_1(1-x_1)\}$. Since there is no conflicting variable in CS, then $Zero(CS/I)$ is not empty. In fact, $Zero(CS/I) = \{(x_1=1,\ x_2=0,\ x_3=0,\ x_4=0,\ x_5=0)\}$. But under the variable order $x_1 < x_2 < x_3 < x_4 < x_5$, a characteristic set of PS is $CS = \{(1-x_1)(1-x_3)x_5,\ (1-x_1)x_3x_4,\ x_3(1-x_3),\ (1-x_1)x_2,\ x_1(1-x_1)\}$, and there is a conflicting variable x_3, thus $Zero(CS/I) = \varnothing$.

Now it is more clear why we decide to start from clause form instead of general well-formed formula form: if starting from general well-formed formula form, first it would be difficult to gain readability, secondly computation of characteristic set using general polynomial representation would be rather inefficient, finally it would be much more difficult to decide $Zero(CS/I)$, compared with Theorem 4.

Characteristic set is the soul of Wu's method. Although when solving SAT, Wu's method has close relations with resolution procedure, but the concept of characteristic set is unique, which makes Wu's method an entirely new algorithm to solve SAT. In fact, the computation of the characteristic set can be extracted from Wu's method and be integrated into other algorithms, in order to make the most of Wu's method. Without the help of algorithm transform, however, we would hesitate to do so because we could not understand Wu's method in such depth.

3.4 Problem decomposition

If no contradiction is produced but $Zero(CS/I) = \varnothing$, then according to Wu's method it is necessary to decompose the problem: we must solve $\{PS, I_i\}$ respectively, where each I_i is the initial of each polynomial of CS.

Since SAT can be naturally decomposed into two branches $x_i = 0$ and $x_i = 1$, and shorter clause is more useful than longer one, we decide to decompose the problem in this way to make use of this special feature of SAT. It is rational to select branching variable from conflicting variables in CS. If there are more than one conflicting variable, we experiment with two strategies: highest first or lowest first.

Now we can view the computing procedure of Wu's method as expanding a search tree, and each node of the tree represents a characteristic set, where an inner node means $Zero(CS/I) = \varnothing$, a leaf node either means a contradiction has been produced and we should backtrack, or means $Zero(CS/I) \neq \varnothing$ and we have found a solution.

3.5 Variable Ordering

Variable ordering is in fact the first step of Wu's method. After representation transform, we should first set a variable order so as to decide the leading variable of each clause polynomial and start computing the characteristic set, *etc*. Variable order has great effect on the efficiency of Wu's method.

However, since there is no general effective rule to decide a good variable order, we can only make use of domain specific heuristics to do it. In fact, when we design DP or backtracking algorithm to solve SAT, variable ordering is also an essential step, which is used for choosing the next branch variable. The commonly used heuristic is to order the variables by the strength of constraints on them and the most constrained first. Of course, the strength of constraints on variable can only be approximately estimated. In our experience, on the commonly used fixed clause-length random instance model of SAT [10], the *product* of the number of positive occurrences and the number of negative occurrences of the variable is a good estimator. Here we try to apply this variable ordering heuristic in DP to the variable ordering in Wu's method, and thus we have at least two variable orders: the more constrained the higher or just the reverse. To further test the effects of these two variable orders, we use random variable order for comparison. We experiment with all these three variable orders in the next section.

Example (*continued*). According to the above heuristic variable ordering rule, we can decide upon a variable order $x_1 < x_3 < x_2 < x_4 < x_5$, corresponding to which a characteristic set of *PS* is $CS = \{x_1(1-x_3)x_5, x_1(1-x_3)x_4, x_1(1-x_3)x_2, x_3(1-x_3), x_1(1-x_1)\}$, as shown before. Since there is no conflicting variable in *CS*, then $Zero(CS/I)$ is not empty, and the clause set is satisfiable. In fact, $Zero(CS/I)=\{(x_1=1, x_2=0, x_3=0, x_4=0, x_5=0)\}$, which gives a satisfiable variable assignment. But, under the natural variable order $x_1 < x_2 < x_3 < x_4 < x_5$, the computing of a characteristic set is much more complicated as shown before; and what's worse, there is a conflicting variable x_3 in the computed characteristic set $CS = \{(1-x_1)(1-x_3)x_5, (1-x_1)x_3x_4, x_3(1-x_3), (1-x_1)x_2, x_1(1-x_1)\}$, thus $Zero(CS/I) = \varnothing$, further decomposition or branching is unavoidable in order to decide whether the clause set is satisfiable.

4 Experiments

In this section, we report detailed experimental results of Wu's method for solving SAT. We should have implemented in computer both representation transform and algorithm transform of Wu's method to demonstrate the superiority of the latter, but representation transform is really too complex to be implemented in computer and its bad performance can be anticipated. Therefore we only implement algorithm transform of Wu's method and mainly want to indicate that, after algorithm transform, Wu's method can easily utilize experience of, or be utilized by, other algorithms. We also want to know the efficiency of our implementation of Wu's method.

The experiments include three parts. First, we test which variable order and branch order is the best, and find out the most efficient implementation of Wu's method for solving SAT. Secondly, we compare Wu's method with two resolution procedures, the semantic resolution and the set-of-support resolution, which are both relatively efficient and closely related to Wu's method. Finally, we compare Wu's method with DP procedure, which is currently the most efficient algorithm for solving SAT. From these experiments, we can have a comprehensive understanding of Wu's method and its algorithm transform..

We use the fixed clause-length random 3-SAT instance model [10] as benchmark because this model is commonly used and can easily generate difficult instances. Most experiments are done on instances with $m/n \approx 4.3$, for this is the most difficult point. All programs are written in C and compiled with -O3 option on SGI Indigo-Elan-4000 workstation.

4.1 Efficient Implementation of Wu's method

Under the algorithm transform of Wu's method defined in Section 3, it is easy to implement Wu's method. We only need to select a better combination of variable order and branch order. We tested 3 variable orders: the more constrained the higher (MCH), the more constrained the lower (MCL), and random variable order (RAN) (see Subsection 3.5) and 2 branch orders: the highest conflicting variable branching first and the lowest conflicting variable branching first (see Subsection 3.4).

Table 1. Implementation of Wu's method

Experiment 1: Implementation of Wu's method 50 variables, 215 clauses, 50 instances(27 satisfiable)							
		lowest branching first			highest branching first		
		time	node	operation	time	node	operation
MCL	sat	0	17	35778	0	65	67684
	unsat	1	28	70909	2	141	156358
	total	0	23	51938	1	100	108474
RAN	sat	1	36	121738	3	144	241661
	unsat	6	113	421027	10	433	716021
	total	4	72	259411	6	277	459867
MCH	sat	29	328	1811622	22	364	1413003
	unsat	109	999	6617467	61	904	3813508
	total	66	637	4022311	40	612	2517235

In Table 1, time unit is second, time = 0 means time < 0.5s, 'node' means the average number of characteristic set computed (see Subsection 3.4), 'operation' means the average number of primitive operations which include Cases 2, 3 and 4 of Theorem 2 since there is little to do in Case 1.

From the data of Table 1, we can know that variable order can significantly influence the performance of the algorithm. Specifically, MCL order is uniformly better than random order, and random order is uniformly better than MCH order. This means that our experience in estimating strength of constraints on variable when designing DP algorithm is really effective when adapting Wu's method to SAT. The success of MCL order also suggests some general rule of variable ordering when using Wu's method to solve general polynomial equations, that is, to place the more constrained variable in lower order may reduce the number of primitive operations when computing characteristic set.

As indicated in Table 1, we will use more-constrained-lower as variable order and lowest-conflicting-first as branching strategy, which is the best combination to implement Wu's method. Table 2 reports the performance of Wu's method on larger scale instances.

Table 2. Performance of Wu's Method

Experiment 2: Performance of Wu's Method variables=100, clauses=400,430,500, instances=100 each					
clauses		instance	time	node	operation
400	sat	99	25	144	952470
	unsat	1	125	591	4703445
	total	100	26	149	989980
430	sat	54	38	165	1394844
	unsat	46	83	313	2997091
	total	100	59	233	2131877
500	sat	1	4	15	150987
	unsat	99	40	84	1430169
	total	100	40	83	1417377

4.2 Comparison with Resolution Procedure

Since Wu's method has intrinsic connections with resolution procedure, we experimentally compared Wu's method with two resolution procedures: semantic resolution and set-of-support resolution.

Semantic resolution [8] has some similarities with Wu's method in that they both need a variable order. Set-of-support resolution is one of the most efficient resolution procedures [7]. Semantic resolution and set-of-support resolution also have deep connections with each other, and they are both complete strategies. This is why we select these two for the comparison with Wu's method.

To implement set-of-support resolution, we need first to determine the set of support. In theorem proving we often use refutation of conclusion as the set of support, yet it does not work in SAT. Therefore, we decide to use an interpretation of propositional variables to produce a set of unsatisfied clauses and treat this set as the set of support. We tried two methods: one is to use a random interpretation, the other is to use local search to find an interpretation under which the set of unsatisfied clauses can be smaller.

To implement semantic resolution, we need determine an interpretation of all variables and a variable order. We tested the same two methods of interpretation selection as used in implementing set-of-support resolution and the same three variable orders as used in implementing Wu's method.

Applying local search to resolution procedure is a new idea. Recently local search has been successfully used in solving SAT [11]. Local search can quickly find a local minimum, *i.e.*, an interpretation of all variables under which there are only a few unsatisfied clauses. In set-of-support resolution, if the initial set of support can be made smaller, the growth of the number of resolvents can be greatly slowed down. This is one of the 33 basic problems proposed by Wos [7]. Here we apply local search to resolution procedure in propositional logic for two purposes: one is to test its feasibility before applied to first-order predicate logic, the other is to speed up set-of-support and semantic resolution in order to do a relatively fair comparison with Wu's method.

We have tested 6 implementations of semantic resolution and 2 implementations of set-of-support resolution in order to find the best. We have used complex indexing techniques and unit-preference strategy to speed up resolution procedure. The maximum capacity of clause list is 10000 clauses. Please refer to [5] for detailed experiment data. On 50 instances of 30 variables and 129 clauses, semantic resolution is better than set-of-support resolution, and MCL variable order is much better than both MCH and random order. In particular, with the interpretation found by local search, the performance of all 6 implementations of semantic resolution has been improved by 100% compared with random interpretation, and by 20% in set-of-support resolution. Therefore we suggest that local search should be introduced to ATP in first-order predicate logic. Local search in (maybe restricted) Herbrand interpretation universe may find a better set of support, and/or a better interpretation for semantic resolution, and/or a better top clause for linear resolution, *etc*. All these steps are essential to the performance of the corresponding algorithms.

We compared Wu's method with semantic resolution using MCL variable order. Although the number of primitive operations in unit propagation is not included in the statistics of semantic resolution, and local search helps improve its performance by 100%, Wu's method is still dominant. In Table 3, 'S.R.1' means semantic resolution with random interpretation, 'S.R.2' means semantic resolution with local search interpretation, 'subsume-1' means number of subsumption test callings by new resolvents, and 'subsume-2' means number of basic subsumption test between two clauses. Note that in 'S.R.2', local search has already solved the satisfiable instances and resolution is no longer needed.

Table 3. Comparison with Semantic Resolution

Experiment 3: Comparison of Wu's Method with Semantic Resolution 30 variables, 129 clauses, 50 instances (25 satisfiable)				
	SAT		UNSAT	
	time	primitive operation	time	primitive operation
S.R.1	20	resolution=23602 subsume-1=20103 subsume-2=5.3M	20	resolution=23708 subsume-1=20249 subsume-2=4.9M
Wu	0	6701	0	8980
S.R.2			8	resolution=10804 subsume-1=9124 subsume-2=1.8M

4.3 Comparison with DP procedure

DP procedure is currently the most efficient complete algorithm for SAT, so we think it is necessary to compare Wu's method with DP procedure.

To implement DP procedure, branching strategy is the essential step. We adopt the commonly used strategy of most constrained variable branching first, and use dynamic variable ordering at each node, which is different from the static one in Wu's method. We also test the idea of embedding characteristic set computing in the node of DP search tree, *i.e.*, treating characteristic set computing as another node operation besides unit resolution. Thus, at least in principle, we achieve a non-trivial combination of DP and the more complex Wu's method. Table 4 gives the experiment data.

From Table 4, we can know that Wu's method is not so fast as DP procedure, although the size of the search tree is smaller. When characteristic set computing is embedded in DP, although it is still worse than pure DP in time, the new algorithm is better than Wu's method both in time and in the number of nodes.

Table 4. Comparison with DP algorithm

Experiment 4: Comparison and Combination with DP algorithm 100 variables, 430 clauses, 100 instances (54 satisfiable)						
	sat		unsat		total	
	time	node	time	node	time	node
DP	0	155	0	471	0	301
Wu	38	165	83	313	59	233
DP+Wu	6	50	19	124	12	84

5 Discussions and Conclusions

In the above sections, we have discussed how to solve SAT by algorithm transform of Wu's method. Here we discuss two further problems.

The first problem is whether algorithm transform always exists. In our opinion, at least in NP-complete problem set, which includes almost all problems with great potential in applications, since representation transforms have been established among the problems and all these problems are *equivalent* from the viewpoint of the worst case complexity, we believe that algorithm transform does exist, but may not be obvious. Therefore the algorithm transform can be discovered only if you have this thought in mind, and design an *appropriate* representation transform as illustrated in this paper, or by some other ways such as abstraction. For example, the local search algorithm for solving SAT, though innovative, can be considered as the abstraction of the old technique for solving Traveling Salesman Problem, with some modifications to the problem representation, that is, transforming a decision problem SAT to an optimization problem MAX-SAT. In recent years, great progress has been

made in designing new algorithms to solve various NP-complete problems, which provides us with a wider space to explore algorithm transform.

The second problem is about the performance of Wu's method. It seems in Section 4 that the performance of Wu's method on random SAT instances is not so attractive compared with DP algorithm. But first, as we have mentioned at the beginning of Section 4, if we had used representation transform of Wu's method to solve SAT, the performance would have been much worse than that of algorithm transform, which is the essence of the problem. Second, Wu's method really brings entirely new concepts and techniques to SAT algorithms, and because of the readability, all of these can be easily exploited by other SAT algorithms to produce more efficient hybrid ones. We should not expect Wu's method to solve all the problem. We only provide a way to make use of such a relatively complex tool. Third, performance is very sensitive to instance model, which is a common phenomenon in experimental comparison of SAT algorithms. More instance models should be tested in the future, which is another problem.

In conclusion, we highly advocate in this paper a new approach to transformation problem solving, namely algorithm transform, and under the guidance of this strategy, we discuss the problem of solving SAT by Wu's method and show that Wu's method, when used to solve SAT, is primarily a restricted resolution procedure. While Wu's method introduces entirely new concepts, *e.g.*, characteristic set of clauses, to resolution procedure, the complexity result of resolution procedure suggests an exponential lower bound to Wu's method when solving general polynomial equations. Moreover, this algorithm transform can help achieve a more efficient implementation of Wu's method since it can avoid the complex manipulation of polynomials and can make the best use of domain specific experience.

Acknowledgements

The first author would like to thank Drs./Profs. Wu Jinzhao (吴尽昭), Wang Jue (王珏), Liu Zhuojun (刘卓军), Zou Hongxing (邹红星), Zhang Ling (张铃), Lu Xuguang (卢旭光) and the editors for their help and encouragement during the preparation of this paper.

References

[1] Davis M, Logemann G, Loveland D. A machine program for theorem proving. *Communications of the ACM*, 1962, 5: 394-397.

[2] Robinson J A. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 1965, 12(1): 23-41.

[3] Wu Wenjun. Basic principles of mechanical theorem proving in geometries. *Journal of Automated Reasoning*, 1986, 2: 221-252.

[4] Kapur D, Lakshman Y N. Elimination theory: an introduction. Chapter 2 in *Symbolic and Numerical Computation for Artificial Intelligence*, Academic Press, 1992.

[5] He Simin. The Design and Analysis of Algorithms for Satisfiability Problem. *Ph.D. dissertation*, Department of Computer Science and Technology, Tsinghua University, 1997.

[6] Kapur D, Narendran P. An equational approach to theorem proving in first-order predicate calculus. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (*IJCAI-85*), Los Angeles, 1985, Vol. 2, pp. 1146-1153.

[7] Wos L. *Automated Reasoning: 33 Basic Research Problems*. Prentice Hall, 1988.

[8] Slagle, J. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 1967, 14: 687-697.

[9] Haken A. The intractability of resolution. *Theoretical Computer Science*, 1985, 39: 297-308.

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

[11] Selman B, Levesque H, and Mitchell D. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (*AAAI-92*), San Jose, CA, July 1992, pp. 440-446.

This work is supported by National Natural Sciences Foundation of China.

He Simin was born in 1968 and had been studying in the Department of Computer Science and Technology, Tsinghua University since 1986 and received Ph.D. degree in 1997. His research interests include algorithmics, especially experimental algorithmics, and their applications.

Zhang Bo was born in 1935 and is a Professor in the Department of Computer Science and Technology, Tsinghua University. He is a member of the Chinese Academy of Sciences and a Foreign Fellow of Russian Academy of Sciences. His research interests are foundations of artificial intelligence, including the theory and application of problem solving, artificial neural networks.