アルゴリズム 23－9
（1991．9．20）

充足可能性問題のための例題生成アルゴリズムについて†

宮野 英次　安部田 英俊　岩間 一雄

九州大学工学部

NP完全問題等の複雑さの高い問題に対して，平均時間的には効率良く解くと主張されるアルゴリズムが数多く開発されている．それらの評価は通常解析的になされるが，実際的立場の研究者からは，実際に計算機実験によっても行なわれるべきだとの指摘がある．本稿では，充足可能問題（SAT）に対するこのようなアルゴリズムの評価に用いるべき例題生成はいかにあるべきかについて考察する．さらに，その要求を満たすような例題生成アルゴリズムが存在することを示す．

Algorithms for Fairly Generating Instances of the Satisfiability Problem

Eiji Miyano, Hidetoshi Abeta and Kazuo Iwama

Department of Computer Science and Communication Engineering
Kyushu University

Fukuoka 812, Japan

For intractable problems such as NP-complete problems, a lot of algorithms, which are claimed to run fast on average, have been developed. Usually their performances have been estimated by mathematical analysis but, especially for practical purposes, it should be also important to estimate them by actually running the algorithms. In this paper we discuss how instances of the satisfiability problem, which are to be used to scale the performance of the algorithms, should be generated. We, in turn, show several instance-generation algorithms that meet the requirements considered.

†科学研究費 01302059，02302047，02650278 の補助を受けた．

1. はじめに

NP完全問題は，一般的には「手に負えない問題」として広く認識されている．しかし，実用上は重要と思われる数多くの問題を含んでおり，何らかの方法で解く必要に迫られる場合も少くない．そこで，実用的には十分有効に動作するようなアルゴリズムの開発への試みが成されている．例えば，代表的なNP完全問題の1つである充足可能性問題（SAT）等に対して，平均時間的には効率良く解くことができると主張されるアルゴリズムがいくつか開発されている[1][2]．これらのアルゴリズムの計算時間の評価は通常解析的手法によって行われるが，複雑な数学に頼る場合が多く，実用的立場の研究者に対しての説得力の点で問題がある．これに対して，計算機実験によってアルゴリズムの計算時間の評価を行なった方がデータの信頼性は高いのではないかという考え方がある．しかし，従来考えられている入力例題は幾つかの問題を含んでいる．

本稿の目的は，計算機実験での入力例題を作るための効率の良いアルゴリズムの開発技術について論じることである．以下の節では，従来の入力例題についての問題点を指摘すると供に，効率の良い例題成立について言及する．さらに，ここで述べる例題生成アルゴリズムへの要求を満たすような充足可能性問題の例題生成アルゴリズムを示し，その良さの考察を加える．

2. 生成アルゴリズムへの要求

計算機実験で入力例題をどのように決めるかについては，2つの考え方があるようである．第1は，論理設計の自動化の分野で通常行なわれているベンチマーク集合を決めるやり方である．これは，実用上意味があって，かつ様々な性質の違いを包含する形で，数10個程度の「例題」を決めてしまうというやり方である．この手法の欠点は，誰でも分かるように，それら数10個の例題に対しアルゴリズムをチューニングするという不正を防止できないことである．第2は，例題をランダムに生成するという考え方である．この手法は上のような不正は防止できるが，次のような，考えようによってはより重大な問題を含んでいる．

(1) 通常適当なパラメータ値（SATなら項当たりのリテラルの個数の期待値）に従って例題を生成するが，多くの場合答がyesの例題とnoの例題の比率が極端に異なり，この情報を悪用される恐れがある．

(2) アルゴリズムの出力が間違っている場合のチェックが容易でない．

このような問題点の解消を目指して，例題生成アルゴリズムは図1に示されるように，yes又はnoの答を入力として，その答となる例題を出力するという形をとる．yes（又はno）の答をとる例題は数多くあるので，それらの中から出力すべき1つを決めるために，$\{0,\ 1\}$上の長さnの列b_1，$\cdots$，b_nも入力として与える．あるいは，これらb_1，$\cdots$，b_nは内部的にランダムに生成しても良い．例題生成アルゴリズムの良さは直観的には次の3点によって決まると考えられる．

(1) 計算時間が短いこと．

(2) 例題の出力に「偏り」がないこと．

(3) 生成アルゴリズムの性質を利用した解法が作りづらいこと．

つまり，速い時間で，あらゆる視点から“ランダム”な例題を生成するような例題生成アルゴリズムを開発しようとしているのである．

3. 形式的枠組

この節では，より理論的な枠組を与える．以下，チューリング機械（TM）としては，オフライン型で出力テープの付いた多テープTMを使用する．最終状態を有し，そこで停止した時の出力テープの内容が出力となる．以下の条件を満たす決定性TM Mが存在する時，言語LはMによって生成されるという．

(1) 任意の入力$x \in \{0,1\}^*$に対し，Mは必ず停止し，Lに属す列yを出力する．（このxが上記のb_1，$\cdots$，b_nになる．）

(2) 任意のLに属す列yに対し，Mがyを出力するような入力$x \in \{0,1\}^*$が存在する．

ここでは，入力として答yes/noを考えていないが，yesに対する例題の集まり，noに対する例題の集まりをそれぞれ独立に扱っていることに注意されたい．さらに次の条件(3)を満たす時，Lは$T(n)$時間で生成可能であるという．

(3) Lに属す長さnの任意の列yに対し，Mがyを$T(n)$ステップ以下で出力するような入力$x \in \{0,1\}^*$が存在する．

以上の定義に関して，以下の注意を与えておく．例えば，Lとして答がyesの充足可能性問題の例題（つまり充足可能なCNF論理式）の集まりを考え，SATとおく．SATを生成するために例えば図2のようなシステムを考えたとしよう．このシステムは我々の定義を満たしているとはいえない．つまり，fが充足不能の場合に出力が出ないからである．b_1，$\cdots$，b_nをいくら長くとってもそのことが解決にならないことに注意されたい．もし，fが充足不能の場合には，あらかじめ決めておいた充足可能な式を出力するというように変更すれば形式的には定義を満たす．しかし，時間を考えてみると，少くともfの答を求める時間がかかる訳であるから，多項式時間ではできそうにない．そこで更に変更を加え，ある多項式時間$T(n)$を決め，それ以内にfの充足可能性が決定できなかった場合もあらかじめ決まった（充足可能な）式を出力するといったシステムも考えられよう．しかし，これは条件(2)に反するであろう．

このように，完全にランダムな例題を生成して，フィルターをかけるという手法は我々の意図するところではない．

4. 充足可能性問題の例題生成アルゴリズム

この節ではSAT（充足可能なCNF式の集合）と$\overline{SAT}$（充足不能なCNF式の集合）の生成アルゴリズムを具体的に与える．まず，SATを生成するためのアルゴリズムは，ランダムに和項を生成する前に論理式の値を1にするような変数の組（1つのセル）を考える．つまり，そのセルを部分項とする和項を無効にすることにして答がyesで

あることを保証する．以下にこの例題生成アルゴリズム SAT-GEN を示す．

アルゴリズム SAT-GEN .

ステップ 1 . 乱数を利用してランダムに 1 つのセル T_{ans} を（各変数を必ず 1 つずつ含んだ和項）を作る．

ステップ 2 . 入力に従って和項 T_{rand} ランダムに作る．

ステップ 3 . ステップ 1 で作ったセル T_{ans} がステップ 2 で作った和項 T_{rand} の部分項になっていなければ，この和項を論理式の 1 つの和項とし，部分項になっている時は T_{rand} を無効にしてステップ 4 へ．

ステップ 4 . 入力がある場合はステップ 2 へ．入力 b_1, $\cdots$, b_n を使い終った場合は終了し，そのときの論理式を出力する． □

定理 1 . アルゴリズム SAT-GEN は任意の充足可能な論理式を生成可能である．

（証明 ） T_{ans} は乱数を利用して生成するために等確率で全てのセルを生成する．同様に和項 T_{rand} もランダムに作られるため任意の充足可能な論理式を生成可能である（T_{ans} は答が yes であることを保証するだけであり，答となる変数の組が 1 つであるとは限らないことに注意して欲しい）． □

アルゴリズム SAT-GEN を，充足可能性問題で通常考えられている適当なパラメータ値（変数の数を v，項数を t，各項での変数の出現確率を p）に従うような集合を出力するように変更することも容易である．

以下に上記のパラメータに従い多項式時間で動作する例題生成アルゴリズム POLY-SAT-GEN を示す．

アルゴリズム POLY-SAT-GEN .

ステップ 1 . 乱数を利用してランダムに 1 つのセル T_{ans} を（各変数を必ず 1 つずつ含んだ和項）を作る．

ステップ 2 . 各変数の出現確率が p であるような和項 T_{rand} をランダムに作る．

ステップ 3 . ステップ 1 で作ったセル T_{ans} がステップ 2 で作った和項 T_{rand} の部分項になっていなければ，この和項を論理式の 1 つの和項とし，項の数が t 個になれば終了．t 個に満たなければステップ 2 へ．部分項になっている時は T_{rand} を無効にしてステップ 2 へ． □

この例題生成アルゴリズム POLY-SAT-GEN については次のような定理 2 が示される．

定理 2 . アルゴリズム POLY-SAT-GEN はパラメータ (v,t,p) に従う全ての充足可能な論理式を多項式時間で生成する．

（証明） ステップ 1 で生成されるセル T_{ans} はランダムに選ばれ，等確率で全てのセルが生成される．ステップ 2 では出現確率 p に従って和項 T_{rand} がランダムに作られるため，任意の和項が論理式の 1 つの和項として生成される．各項は多項式時間で実行可能であり，充足可能性はステップ 1 のセルにより保証されるので，アルゴリズム POLY-SAT-GEN はパラメータ (v,t,p) に従う全ての充足可能な論理式を多項式時間で生成することができる． □

任意に生成された和項が，先に生成されたセルに含まれないリテラル（変数の肯定もしくは否定）を 1 つでも含んでいれば良いので，次のような例題生成アルゴリズムも考えられる．

アルゴリズム POLY-SAT-GEN-II .

ステップ 1—ステップ 2 . アルゴリズム POLY-SAT-GEN と同じ．

ステップ 3 . ステップ 1 で作ったセル T_{ans} がステップ 2 で作った和項 T_{rand} の部分項になっていなければ，この和項を論理式の 1 つの和項とし，部分項になっている時は T_{rand} の 1 つの変数をランダムに選び，肯定であれば否定に，否定であれば肯定に変更した和項を論理式の和項とする．項の数が t 個になれば終了．t 個に満たなければステップ 2 へ． □

この例題生成アルゴリズムが多項式時間で実行可能であることも定理 2 と同様にいうことができる．

従って SAT に関しては第 2 節で述べた良さの条件 (1) 及び (3) を完全に満たした生成アルゴリズムが存在することになる．条件 (2) に関しても，形式的に証明することはできていないが，直観的には問題ないがないことを 5 節で計算機実験により示す．

次に $\overline{SAT}$ を出力する例題生成アルゴリズムを示す．

アルゴリズム $\overline{\text{SAT}}$-GEN .

ステップ 1 . ランダムに変数を 1 つ選び，その肯定の和項と否定の和項で論理式を作る．

ステップ 2 . ステップ 1 で作られた 2 項（変数が 1 つの項）のどちらか一方をランダムに選ぶ．さらに他の変数を任意に 1 つ選び，その変数で分解（ここでは和項を T とし変数を y とした時に $T=(T+y)\cdot(T+\overline{y})$ とする演算を分解と呼ぶ）を実行する．

ステップ 3 . 生成された論理式を F_{old} として，次のステップ 3 − 1, 3 − 2, 3 − 3 を任意に 1 つ実行する．

ステップ 3-1 . F_{old} の和項の中からランダムに和項を 1 つ選ぶ．さらにその和項に含まれない変数を任意に 1 つ選び，その変数で分解を実行して，新しい論理式 F_{new} を作る．

ステップ 3-2 . 2 つ以上の変数を含んだ和項及びその和項に含まれる変数を 1 つランダムに選んで，その変数を削除して，新しい論理式 F_{new} を作る．

ステップ 3-3．2つ以上の変数を含んだ和項及びその和項に含まれる変数を1つランダムに選んで，その変数を削除する．ここで，変数を削除することでできた和項を部分項とする和項があれば削除して，新しい論理式 F_{new} を作る．

ステップ 4．F_{new} を F_{old} としステップ3へ．入力 b_1，b_n を使い終ったところで終了し，そのときの F_{new} を出力する．□

定理 3．アルゴリズム $\overline{\text{SAT}}$-GEN は任意の充足不能な論理式を生成可能である．

（証明）任意にある論理式が与えられた時，その論理式の共有項を繰り返し生成することで全ての主項を求めることができる（*iterated−consensus method*）．さらに，得られた主項の中から冗長な項を取り除くことで式の簡単化を実現することが可能である．このような論理式の簡単化を施す際に利用される公理としては，次のようなものが挙げられる．

1．（巾等律） $x \cdot x = x$

2．（吸収律） $x \cdot (x + y) = x$

3．（共有項律） $(\overline{x}+y)\cdot(x+z)\cdot(y+z) = (\overline{x}+y)\cdot(x+z)$

また，次のような良く知られた演算（公理系から簡単に導くことができる）も利用される．

4． $(x + y) \cdot (x + \overline{y}) = x$

アルゴリズム $\overline{\text{SAT}}$-GEN は，ステップ3の各繰り返しの中で3−1から3−3を適当に選び組み合わせることにより，共有項や冗長な項の生成を実現可能としている．この意味でアルゴリズム $\overline{\text{SAT}}$-GEN は上記の簡単化手法を逆方向に行なっていると言える．例えば，分解の操作は上の4の操作の逆操作そのものであり，和項 T を変数 v で分解して，一方の和項から v を削除することで上記2の逆操作を実現していることになる．他の操作についても簡単に調べることができる．

充足可能でない論理式の簡単化はある変数の肯定と否定の積へと帰着することになるので，アルゴリズム $\overline{\text{SAT}}$-GEN は任意の充足不能な論理式を生成することができる．□

定理3は計算時間のことに言及していないが，多項式時間では不可能である．それでは，アルゴリズム $\overline{\text{SAT}}$-GEN を適当な多項式時間で打ち切ってみるとどうだろうか．この場合もはや充足不能な論理式をすべて生成することはできず，ある部分集合 $\overline{SAT}^*$ を生成する．問題は，この $\overline{SAT}^*$ に対して，2節の条件（3）が満たされているかどうかという点である．以下の定理は，この点に関して心配の必要がないことを示している．

与えられた和積形の論理式に対し，以下の2種類の操作のみで0（恒偽）が導けるかどうかを問う問題をCNF恒偽証明問題と呼ぶことにする．（i）A を和項，x を変数とした時，A を $(A+x)$ で置き換える．（ii）$(A+x)(A+\overline{x})$ と書ける2つの和項を1つの和項 A で置き換える．（i）は上記ステップ3-2の，また（ii）はステップ3-1の逆操作になっていることに注意されたい．つまり，CNF恒偽証明問題は，与えられた式が上記アルゴリズムの3-3を以下で置き換えたようなアルゴリズムによって生成されるかどうかを問う問題である．

ステップ 3-3 ランダムに構成された和項を追加する．

この場合，項の数は単調に増加するので，全ての式は（生成されるのであれば）多項式時間で生成される．

定理 4．CNF恒偽証明問題はNP完全である．

（略証）NPに入ることは上記の単調性より明らかである．NP困難性は3SATを本問題に帰着させることにより証明する．3SATのインスタンス A よりやはり和積形の論理式 B を構成する方法を述べる訳であるが，先ず，A には直接依存しない部分から説明する．以下のような和項を用意する．

$$\begin{aligned}&(x_1 + y_1 + U_1(1) + V(1) + z_1 + \sigma)\\ &\quad(x_1 + y_1 + U_1(2) + V(1))\cdots\\ &\quad(x_1 + y_1 + U_1(k) + V(1)) \qquad (1)\\ &(\overline{x_1} + \overline{y_1} + U_1(1) + V(1) + z_1 + \sigma)\\ &\quad(\overline{x_1} + \overline{y_1} + U_1(2) + V(1))\cdots\\ &\quad(\overline{x_1} + \overline{y_1} + U_1(k) + V(1)) \qquad (2)\end{aligned}$$

ここで，x_1，y_1，z_1 は A の第1の変数に対応し，σ は後に $\sigma\overline{\sigma}$ を導く為の重要な変数である．k としては，（余裕を見て）A の項数とする．$U_1(1)$，$U_1(2)$，$U_1(3)$，$U_1(k-1)$，$U_1(k)$，は以下のような式になる．

$$\begin{aligned}&u_{11},\ \overline{u_{11}} + u_{12},\ \overline{u_{11}} + \overline{u_{12}} + u_{13},\cdots,\\ &\overline{u_{11}} + \overline{u_{12}} + \cdots + \overline{u_{1(k-2)}} + u_{1(k-1)},\\ &\overline{u_{11}} + \overline{u_{12}} + \cdots + \overline{u_{1(k-2)}} + \overline{u_{1(k-1)}}\end{aligned}$$

つまり，

$$(A + U_1(1))(A + U_1(2))\cdots(A + U_1(k))$$

は操作（ii）のみの適用によって A にすることができる．$V(1)$，$V(2)$，$\cdots$ も同様である．A の第2の変数に対しても x_2，y_2，z_2 を使って同様の項を用意する．第3以降の変数に対しても同様．

上記の $U_1(i)$ の性質より，式（1）と $\overline{x_2}$，$\overline{y_2}$，z_2 に対する同様の式は

$$\begin{aligned}&(x_1 + y_1 + z_1 + V(1) + \sigma)\\ &(\overline{x_2} + \overline{y_2} + z_2 + V(2) + \sigma)\end{aligned}$$

のように簡単化できる．さらに，

$$(z_1 + \overline{x_1})(z_1 + x_1)(z_1 + \overline{y_1})(z_1 + y_1)(z_2 + \overline{x_2})\cdots$$

のような項を用意すれば，これらは

$$(z_1 + \sigma + V(1))(z_2 + \sigma + V(2))\cdots$$

のように簡単化され，さらに $(\sigma+\overline{z_1})(\sigma+\overline{z_2})\cdots$ を用意することにより

$$(\sigma+V(1))(\sigma+V(2))\cdots$$

となって，これは σ に簡単化される．つまり，A に現れる全ての変数に対して，(1) か (2) のどちらかの形のみをとって上記の操作をおこなうと σ が導ける．

次に以下のような項を用意する．

$$(\overline{x_1}+\overline{y_1}+x_2+y_2+\overline{x_5}+\overline{y_5}+s_1+t_1+W(1)+\overline{\sigma}) \quad (3)$$

これは，与えられた式 A の第 1 の項が第 1 の変数の否定，第 2 の変数，第 5 の変数の否定で構成されていることに対応する．以下が重要な点である．前述の (2) 式の最初を除く適当な 2 項を使うと，上記 (3) の s_1 と t_1 を“消して”次の式を導ける．

$$(\overline{x_1}+\overline{y_1}+x_2+y_2+\overline{x_5}+\overline{y_5}+U_1(i)+V(1)+U_1(j)+V(1)+W(1)+\overline{\sigma}) \quad (4)$$

次に $\overline{x_1}$，$\overline{y_1}$ 等や $U_1(i)$ 等に出現する全ての変数を消すために次のような項を用意する．

$$(\overline{s_1}+\overline{t_1}+x_1)(\overline{s_1}+\overline{t_1}+y_1)\cdots$$
$$(\overline{s_1}+\overline{t_1}+U_1\text{の変数の否定})\cdots$$
$$(\overline{s_1}+\overline{t_1}+V\text{の変数の否定})\cdots$$

これらの操作の適用は (3) の s_1 と t_1 を消しておいたから可能になったことに注意されたい．こうして (4) 式は

$$(\overline{s_1}+\overline{t_1}+W(1)+\overline{\sigma})(\overline{s_2}+\overline{t_2}+W(2)+\overline{\sigma})\cdots$$

のように変形され，さらに

$$(\overline{\sigma}+s_1)(\overline{\sigma}+t_1)(\overline{\sigma}+s_2)(\overline{\sigma}+t_2)\cdots$$

によって

$$(W(1)+\overline{\sigma})(W(2)+\overline{\sigma})\cdots$$

を経て $\overline{\sigma}$ へ変形される．こうして σ と $\overline{\sigma}$ より恒偽が導かれる．

(3) 式の s_1 と t_1 を消すために (2) 式の一部の項を利用したが，その結果 (2) 式からはもはや σ が導けなくなることが判る．つまり，全ての変数に対して，(1) 式か (2) 式のいずれかが完全な形で残ることが要求される．ここで (3) 式の s_1 と t_1 を消すことが，もとの式 A の第 1 の変数に 0 を割り当ててることに対応していることに注意して欲しい．以上証明の概略であって，意図した以外の不都合な簡単化が生じないように細かい修正が必要になる．□

次にパラメータ (v,t,p) に従い，充足不能な例題集合を多項式時間で生成するための例題生成アルゴリズムを与える．基本方針は SAT-GEN と同様である．

アルゴリズム POLY-SAT-GEN .

ステップ 1 . $x_1 \sim x_v$ まで各変数が出現確率 p に従って論理式中に何個現れるかを，乱数を利用して決める．

ステップ 2 . 出現個数が最大である変数 x_{max} とその否定 $\overline{x_{max}}$ を和項とする論理式を作る．

ステップ 3 . ランダムにどちらか一方を選び，x_{max} の次に出現個数が多い変数 x_{max-1} で分解を実行する．

ステップ 4 . 生成された論理式 F_{old} の中から和項を 1 つを選び，その和項に含まれない変数の中で出現個数が最大の変数で分解を実行する．この時ステップ 1 で得られた出現個数を既に満たしている変数については，その変数と 1 つ以上の他の変数を含む和項をランダムに選び，変数を削除して部分項を作る．

ステップ 5 . 変数を削除することで新しく作られた和項を部分項とするような和項を削除して論理式 F_{new} を作る．

ステップ 6 . 全ての変数について出現個数を満たせばステップ 7 へ．出現個数を満たしていない変数があれば，F_{new} を F_{old} としてステップ 4 へ．

ステップ 7 . 出現確率 p に従った任意の和項を作り，この和項で現れた変数については，その変数と 1 つ以上の他の変数を含む和項をランダムに選び，変数を削除する．項数が t 個になるまで繰り返す．□

5. 計算機実験

この節では，2 節で述べたような例題生成アルゴリズムに求められる条件を，4 節の多項式時間例題生成アルゴリズムが満足しているかどうかを計算機実験で調べた結果を示し，その考察を行なう．

直観的には，アルゴリズム POLY-SAT-GEN 及び POLY-SAT-GEN-II では，各変数に関して肯定か否定かがステップ 1 で設定したセル T_{ans} により影響を受け易いのではないかと考えられる．例えば 5 変数の時，$T_{ans}=\overline{x_1}+\overline{x_2}+x_3+\overline{x_4}+x_5$ であったとすると，x_1，x_2，x_4 については極端に肯定の数が多く，x_3，x_5 については極端に否定の数が多いようであれば，この情報を悪用される可能性がある．これは 2 節の条件 (2) に反するものである．

そこでアルゴリズム POLY-SAT-GEN に従い，実際に計算機で例題の出力を行ない，各変数が答えのリテラルに本当に影響を受け易いのかどうかを調べた．実験では，各変数について肯定と否定の出現回数を調べ，v 変数の中でステップ 1 で生成された T_{ans} に含まれるリテラルと同じリテラル (T_{ans} で肯定の時は肯定，否定の時は否定) が多く現れた変数の数と，異なるリテラル (T_{ans} で否定の時は肯定，肯定の時は否定) が多く現れた変数の数の平均を調べた．試行回数はいずれも 50 回である．表 1では各項目の左から，答と同じリテラルが多く現れた変数の平均，肯定と否定が同数であった変数の平均，答と異なるリテラルが多く現れた変数の平均をそれぞれ表している．

この表から，例えば 50 変数，確率 0.5 の時には，50 変数の中で 24 変数程度は答と同じリテラルの方が多く現れており，23 変数程度が異なるリテラルが多く現れ，3 変数程度が肯定，否定を同数出現していることが判る．この

表 1: T_{ans} によるリテラルへの影響

$p \setminus v$	10 変数	50 変数	100 変数
0.1	0.48:0.22:9.30	19.8:5.70:24.5	43.2:11.8:45.0
0.3	2.14:0.50:7.36	23.9:3.10:23.0	46.0:7.00:47.0
0.5	3.36:0.48:6.16	23.1:3.00:23.9	46.9:6.00:47.1
0.7	4.24:0.54:5.22	23.2:2.30:24.5	47.5:4.80:47.7
0.9	4.76:0.58:4.66	23.3:2.40:24.3	47.7:4.10:48.2

実験により，変数の数が 50 以上の場合は全ての確率で半分程度の変数については T_{ans} と同じリテラルの方が多く出現しており，10 変数の時も確率 0.5 以上では T_{ans} による影響はそれ程見られない．このことから，変数の出現状況を悪用して答を予想するようなことはないと思われる．これより，例題生成アルゴリズム POLY-SAT-GEN は生成アルゴリズムに対する条件 (1) (2) (3) 全てを満足しており，我々にとって望ましい例題を生成することができると言える．

アルゴリズム POLY-$\overline{\text{SAT}}$-GEN では，出現確率 p により各変数の出現個数を求め，各変数がその個数を満たすようにアルゴリズムの各繰り返しで分解により和項を生成している．そのため項数 t を満足しても出現個数を満たさない場合がある（つまり，各和項でリテラルの個数の期待値より小さくなっている）．これは，特に出現個数の小さな変数で見られる．この点について改良を加える．アルゴリズムの最初の v 回の繰り返しでは，それまでに生成された論理式 F_{old} の中から和項をランダムに選び，和項の分解を変数の出現個数の順に行ない，全ての変数を用いる．その後の繰り返しからステップ 4 を実行する．このような改良により全ての変数で効率良く出現個数を満たすようになると思われる．

6. 今後の課題

実際に計算機により例題を生成した時，アルゴリズム POLY-SAT-GEN 及び POLY-SAT-GEN-II に関しては，各パラメータに従い，さらにランダムに例題を生成した場合の変数の分布にも従っているのに対して，アルゴリズム POLY-$\overline{\text{SAT}}$-GEN では 1 つの項当りの変数の個数の分布に関して従っているとは言えない．そのため一見して例題の形から答が yes/no であることが判る可能性がある．この点についての改良を現在実行中である．更に重要な課題としては，例題の出力の“偏り”に関しては実験結果等により直観的には示すことができたと思われるが，各例題生成アルゴリズムについてより形式的な検証を行なうことが必要であると思われる．

参考文献

[1] K.Iwama, “CNF satisfiability test by counting and polynomial average time,” SIAM J. Comput., pp.385-391, 1989.

[2] P.Purdom and C.Brown, “The pure literal rule and polynomial average time,” SIAM J. Comput., pp.943-953, 1985.

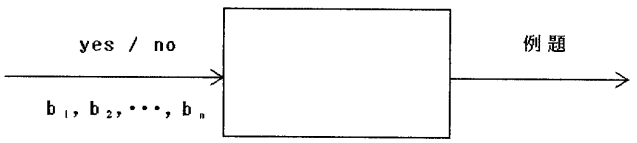

図 1

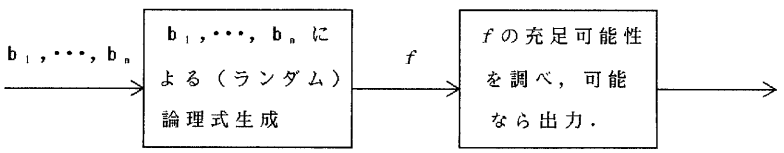

図 2