数理解析研究所講究録
第833巻 1993年 22-30

指定された分布パラメータを満足する SATの例題生成について

宮野 英次　　岩間 一雄
Eiji Miyano and Kazuo Iwama

九州大学工学部情報工学科
Department of Computer Science and Communication Engineering
Kyushu University

1. はじめに

NP完全問題の困難さは広く認識されている．しかし，重要と思われる数多くの問題を含んでおり，実用的には効率良く動作するアルゴリズムの開発とその評価が試みられている．例えば，NP完全問題の1つである充足可能性問題（SAT）などに対して，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 [2][4]．しかし，その評価は複雑な数学を伴うことが多く，実用的立場の研究者を説得するためには，実際に計算機実験によって評価することが重要であると言われている．問題は入力例題をどのように生成するかであるが，通常行われるベンチマーク集合の利用，完全ランダム生成法は幾つかの問題点を含んでいる．この点を解消するために，答えを指定した上で例題を生成できる例題生成系を提案した [3]．答え（yes または no）の入力に対して，yes の例題と no の例題を乱数を用いてそれぞれ独立にランダムに生成して出力する．

本稿では代表的なNP完全問題であるSATの例題生成について考察する．本研究の目標は，効率が良く，安全である例題生成系を開発することである．効率とは時間的な効率を意味し，多項式時間で動作するような例題生成系を開発することを目標とする．安全性としては，(i) 複雑論的安全性，(ii) 統計的安全性という2つの安全性を考える．複雑論的安全性とは，多様な例題が生成可能で問題本来の難しさを維持しているかどうかを意味する．統計的安全性とは生成可能な個々の例題の生成確率も考慮に入れた時，都合の悪い偏りが生

じないかどうかを意味する.

本稿では特に (ii) の安全性に着目をする. 仮に, *yes*, *no* のそれぞれの例題生成について複雑論的安全性を満足しており, すべての充足可能な例題および充足不能な例題を生成することが可能であるとする. しかし, それぞれの例題集合が明らかに異なる分布に従うようであれば問題がある. 例えば, 充足可能な例題に関しては, 充足不能な例題に対して項に含まれるリテラルの数が比較的多いものばかりがほとんどいつも生成されるのであれば, 簡単な計算により (ほとんどの場合正しい) 答を推定することができる. しかし, 個々の例題の生成確率を厳密に調べることは非常に難しく, それぞれの出力集合の正確な分布を解析することも事実上不可能であろう. そこで, ある意味では非常に制限された, 比較的容易にその分布を特徴付けることが可能な例題集合を生成することを考える. 本稿ではとりあえず, 充足可能な生成例題と充足不能な生成例題が, 少なくとも "見かけ上似ている" ような例題生成系を開発する. ここで, "見かけ上" とは, (他にもあるであろうが) リテラルの出現頻度の分布と各項に含まれるリテラルの数の分布を意味する. 指定されたリテラルの出現頻度の分布パラメータを満足し, 3-SAT (各項に正確に 3 個のリテラルを含む) に対する充足可能な例題を生成する例題生成アルゴリズムと充足不能な例題を生成する例題生成アルゴリズムを示す.

2. 例題生成アルゴリズム

充足可能な例題を出力する生成系および充足不能な例題を出力する生成系は, それぞれある言語を生成するとみなすことができる. そこで, 次のような決定性チューリング機械 G を考える: G は入力列 $r \in \{0,1\}^*$ に従い CNF 式 $f(r)$ を出力する. G によって生成される言語を $L(G)$ とすると, $L(G) = \{f(r) | r \in \{0,1\}^*\}$ と定義される. 出力の長さを n とするとき, もしも G が $p(n)$ ステップで停止するならば, G は $p(n)$ 時間で動作すると言う. 入力列 r は $L(G)$ の中の 1 つを選ぶために使われ, 実際には G の内部でランダムに生成されてもかまわない. すべての充足可能な例題の集合を I_{sat}, すべての充足不能な例題の集合を $I_{\overline{sat}}$ で表す. 以下に充足可能な例題に対する最も基本的な例題生成アルゴリズムを示す.

Generator SAT-GEN:

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

ステップ 2. 和項 A をランダムに作る.

ステップ3. 和項 A が C_{ans} を包含していなければ，A を例題の1つの和項とする．そうでなければ，A の1つのリテラル x' をランダムに選び，$x' = x$ ならば $\overline{x}$ とし，$x' = \overline{x}$ ならば x とする．そのような和項 A' を例題の1つの和項とする．

ステップ4. 入力列を使い終わるまでステップ2および3を繰り返し，そのときの例題を出力する． □

次に，充足不能な例題に対する例題生成アルゴリズムを示す．

Generator $\overline{\text{SAT}}$-GEN:

ステップ1. ランダムに変数 x を1つ選び，$f_{now} = x\overline{x}$ とする．

ステップ2. (2-1) ~ (2-4) を任意に1つ選び実行する． and executed:

(2-1) f_{now} の中から和項 A と変数 x をそれぞれ1つランダムに選ぶ． A を $(A+x)(A+\overline{x})$ と分解して f_{now} を変更する．

(2-2) f_{now} の中から和項 A と A に含まれるリテラル x' をそれぞれ1つランダムに選ぶ．A から x' を削除して f_{now} を変更する．

(2-3) 和項 A が和項 B を包含するような f_{now} の2つの和項 A および B をランダムに選ぶ． B を f_{now} から削除する．

(2-4) 和項 A をランダムに作り， A を f_{now} の1つの和項とする．

ステップ3. 入力列を使い終わるまでステップ2を繰り返し，使い終わった時の f_{now} を出力する． □

定理1. (i) SAT-GEN は I_{sat} を生成する． (ii) $\overline{\text{SAT}}$-GEN は $I_{\overline{sat}}$ を生成する． □

SAT-GEN は多項式時間内で動作するのに対して，$\overline{\text{SAT}}$-GEN は多項式時間内では動作しない．しかし，仮に $\overline{\text{SAT}}$-GEN のステップ2の (2-3) の繰り返し回数が多項式回程度で抑えられれば，多項式時間での動作が可能である．ここで $d(n)$ を n の多項式とする．

Generator PRV($d(n)$)-GEN: $\overline{\text{SAT}}$-GEN と同様であるが，ステップ3を変更する．

ステップ3. ステップ2を繰り返す．ただし， (2-3) の操作の使用回数を $d(n)$ 回以下とする． □

3. 複雑論的安全性

I_{sat} がSAT-GENにより多項式時間で生成可能であるのに対して，$I_{\overline{sat}}$ を多項式時間で生成することは非常に難しい．その原理的な要因は，仮にNP=co-NPでないのであれば，多項式時間ですべての充足不能な例題を生成することは不可能であるということである．実際にPRV($d(n)$)-GENもすべての充足不能な例題を生成することはできない．出力される充足不能な例題が $I_{\overline{sat}}$ の部分集合 $L(\overline{G})$ に限られた場合，$L(\overline{G})$ がクラスPであるようであれば問題が簡単になってしまい，複雑論的安全性を満足しているとは言えない．$L(\overline{G})$ のNP完全性を示すことができれば，少なくとも $L(\overline{G})$ がクラスPに属さないことを示すことができるが，それだけではまだ充分ではない．仮に，$L(\overline{G}) \subseteq S \subseteq \overline{I_{sat}} = I_{\overline{sat}}$ を満たすような多項式時間集合 S が存在するようであれば，生成された出力例題が $L(\overline{G})$ であるのか，I_{sat} であるのかの判定が多項式時間で可能であることが知られている [1].

このような意味での安全性を満足するための自然な方法は，$L(\overline{G})$ をできるだけ大きくして $I_{\overline{sat}}$ に近づけることである．このような意味で，PRV($d(n)$)-GENは次のような望ましい性質を持っている．

定理 2. L(PRV($d_i(n)$)-GEN) はNP完全集合であり，それぞれの i について L(PRV($d_i(n)$)-GEN) $\subset L$(PRV($d_{i+1}(n)$)-GEN) を満足するような多項式の有限列 $d_0(n)$, $d_1(n), \cdots$ が存在する． □

このように，多項式時間内で時間をかけることにより L(PRV($d_i(n)$)-GEN) を $I_{\overline{sat}}$ に近づけていくことができるため，

$$\underbrace{L(\mathrm{PRV}(d_0(n))\text{-GEN}) \subset L(\mathrm{PRV}(d_1(n))\text{-GEN}) \subset \cdots}_{\text{NP-complete}} \subset \underbrace{L}_{\text{P}} \subset \underbrace{L(\mathrm{PRV}(2^{O(n)})\text{-GEN}) = I_{\overline{sat}}}_{\text{co-NP-complete}}$$

を満足するような多項式時間集合 L は存在しないのではないかと考えられる．

4. 統計的安全性

例題生成系は充足可能な例題に対するものと充足不能な例題に対するものからなる．本節の目標は，それら2つの出力例題集合が“見かけ上似ている”ような例題生成系の開発である．次のような2つのパラメータを考える．1つは，$(N(x_1), N(\overline{x_1}), \cdots, N(x_n), N(\overline{x_n}))$ で表されるリテラル分布である．ここで，$N(v)$ （v は x_i または $\overline{x_i}$）は生成例題のリテラ

ル v の出現数を表す．もう1つは，$(T(1),T(2),\cdots,T(t))$ で表される項分布である．ここで，$T(i)$ は正確に i リテラルからなる項の数を表す．以下に項分布が $(0,0,t,0,\cdots)$，つまり，いわゆる 3-SAT に対して，リテラル分布 N を満足する充足可能な例題を生成するアルゴリズム 3SAT(N)-GEN を示す．

Generator 3SAT(N)-GEN:

ステップ 1. 項数 $t=\frac{1}{3}\sum_v N(v)$ を計算する．

ステップ 2. セル $C_{ans}=v_1+v_2+v_3+\cdots+v_i+\cdots+v_n$ をランダムに作る．ただし，v_i はリテラル x_i または $\overline{x_i}$ である．C_{ans} をリテラルの集合 $\{v_1,v_2,\cdots,v_n\}$ で表すことにする．

ステップ 3. $\sum_{v\in C_{ans}} N(\overline{v})\geq t$ であれば，ステップ5へ．そうでなければ，ステップ4へ．
（$\overline{v}$ は，$v=x_i$ であれば $\overline{x_i}$，$v=\overline{x_i}$ であれば x_i を表す．）

ステップ 4. C_{ans} から $N(\overline{v})<N(v)$ を満たすリテラル v を選び，C_{ans} の v を $\overline{v}$ とする．ステップ3へ．

ステップ 5. $\{\overline{v_i}|v_i\in C_{ans}\}$ の集合の中から N を越えないように t 個のリテラル $w_1,w_2,\cdots,w_t$ をランダムに選ぶ．例題 f_{now} を $f_{now}=(w_1)(w_2)\cdots(w_t)$ とする．

ステップ 6. f_{now} から3リテラルより少ない和項 A と f_{now} の中の出現数が $N(v)$ 回より少ないリテラル v をランダムに選ぶ．A を $(A+v)$ と置き換えて f_{now} を変更する．すべての和項が3リテラルになるまでこの変更を繰り返す． □

充足不能な例題の生成は，基本的な操作に関しては $\overline{\text{SAT}}$-GEN とまったく同様であるが，リテラル分布 N に従い，すべての項が3リテラルになる例題を生成するために，それらのパラメータ値を計算しながら例題の生成を行なう必要がある．この場合，自然な方法として例題の生成過程で分布パラメータに少しずつ合わせながら例題を生成することが考えられる．例題の生成過程での項に含まれるリテラルの数に関しては次のような性質を持つ．

性質 1. 例題の生成過程で，常に各項のリテラルの数が4リテラル以下であるように限ると，すべての例題を生成することはできない． □

（証明） 次のような論理式 f を考える．

$$\begin{aligned} f \;=\; & (x_1+x_2+\overline{x_3})(x_1+\overline{x_2}+x_3)(x_1+\overline{x_2}+\overline{x_3}) \\ & (\overline{x_1}+x_2+x_3)(\overline{x_1}+x_2+\overline{x_3})(\overline{x_1}+\overline{x_2}+x_3)(\overline{x_1}+\overline{x_2}+\overline{x_3}) \\ & (x_2+x_3+\overline{x_4})(x_1+x_3+\overline{x_5})(x_1+x_2+\overline{x_6})(x_4+x_5+x_6) \end{aligned}$$

f は各項のリテラルの数を 4 以下に限定しなければ，容易に生成可能である．例題の生成過程で，常に各項のリテラルの数が 4 リテラル以下に限った場合には f が生成不可能であることを示すために，2 つの補題を示す．

補題 1. 例題 g が m 項であるとき，n 回の分解を行なうと $m+n$ 項になる． □

補題 2. $A = y_1 + y_2 + \cdots + y_k$ とするとき，項 $(A+y)$ $(y \notin A)$ が生成されるためには，少なくとも A を含む項を変数 y で分解しなければならない． □

まず，$f_{now} = x_1\overline{x_1}$ を x_2，x_3 で分解を行ない，f_{now} のすべての項を 3 リテラルにした場合を考える：

$$\begin{aligned} f_{now} \;=\; & (x_1+x_2+x_3)(x_1+x_2+\overline{x_3})(x_1+\overline{x_2}+x_3)(x_1+\overline{x_2}+\overline{x_3}) \\ & (\overline{x_1}+x_2+x_3)(\overline{x_1}+x_2+\overline{x_3})(\overline{x_1}+\overline{x_2}+x_3)(\overline{x_1}+\overline{x_2}+\overline{x_3}). \end{aligned} \tag{1}$$

f に含まれる項の変数の組合せは $\{x_1,x_2,x_3\}$，$\{x_2,x_3,x_4\}$，$\{x_1,x_3,x_5\}$，$\{x_1,x_2,x_6\}$，$\{x_4,x_5,x_6\}$ であるので，補題 2 より f_{now} のある 1 つの項を x_4，x_5，x_6 で分解を行なう必要がある．ただし，f は 11 項であるので，補題 1 より 3 回の分解により f を生成しなければならない．仮に，$(\{x_1,x_2,x_3\}+x_4)(\{x_1,x_2,x_3\}+\overline{x_4})$ と分解したとする．$(x_2+x_3+\overline{x_4})$ を生成するためには，第 2 項から $\overline{x_1}$ を削除しなければならない．このとき，第 1 項から変数の組合せ $\{x_1,x_3,x_5\}$，$\{x_1,x_2,x_6\}$，$\{x_4,x_5,x_6\}$ を同時に満足するようにリテラルを削除することができないため，f を生成することはできない．$(\{x_1,x_2,x_3\}+x_5)(\{x_1,x_2,x_3\}+\overline{x_5})$ や $(\{x_1,x_2,x_3\}+x_6)(\{x_1,x_2,x_3\}+\overline{x_6})$ としたときも同様である．

次に，最初に $f_{now} = x_4\overline{x_4}$ とした場合を考える．仮に，ある変数 y で第 1 項を分解して $(x_4+y)(x_4+\overline{y})$ としたとする．このとき，f にはリテラル x_4 は 1 回しか現れておらず，どちらかの項から x_4 を削除することができる．$\overline{x_4}$ についても同様のことが言え，例えば，$f_{now} = (x_4+y)(\overline{y})(\overline{x_4}+y)(\overline{y})$ のように変換される．さらに，変数 z で第 1 項を分解して $(x_4+z)(x_4+\overline{z})$ としたとすると，例えば，$f_{now} = (x_4+y+z)(y+\overline{z})(\overline{y})(\overline{x_4}+y+z)(y+\overline{z})(\overline{y})$ のような式になる．この場合，すべての項を 3 リテラルにするためには 8 回の分解が必要であり，その結果 14 項が生成される．従って，f を生成することができない．最初に $f_{now} = x_5\overline{x_5}$ や $f_{now} = x_6\overline{x_6}$ としたときも同様である．その他の分解やリテラルの削除の組合せを考えてみても，f を生成することができないことを証明することができる． □

以下にリテラル分布 N を満足する充足可能な 3 リテラル式を生成するアルゴリズム 3PRV$(0,N)$-GEN を示す．3PRV$(0,N)$-GEN はリテラル分布を少しずつ合わせながら例題

を生成していく．

Generator 3PRV(0, N)-GEN:

ステップ 1. ランダムに変数 x を 1 つ選び，$f_{now} = x\bar{x}$ とする．

ステップ 2. (2-1) ～ (2-3) を任意に 1 つ選び実行する．

(*2-1*) f_{now} の中から和項 A と f_{now} の中の出現数が $N(v)$ 回より少ないリテラル v をそれぞれ 1 つランダムに選ぶ．A を $(A+x)(A+\bar{x})$ と分解して f_{now} を変更する．仮に，新しい f_{now} の中のリテラル u の出現数が $N(u)$ を越えたとき（つまり，$N(u)+1$ 回）は，u を含む和項 B を選び，B から u を削除する．u のようなリテラルすべてについて繰り返す．

(*2-2*) f_{now} の中から 4 リテラル以上の和項 A を選ぶ．A からリテラルをランダムに削除することにより A を 3 リテラルにする．

(*2-3*) f_{now} のすべての和項が 3 リテラルになればステップ 4 へ．

ステップ 3. リテラル分布を満たし，すべての和項が 3 リテラルになるまでステップ 2 を繰り返す．

ステップ 4. f_{now} の中の出現数が $N(v)$ 回より少ないリテラル v により 3 リテラルの和項をランダムに作り，f_{now} の 1 つの和項とする．リテラル分布 N を満足するまでステップ 4 を繰り返す． □

定理 3. (i) 3SAT(N)-GEN はリテラル分布 N を満足するすべての充足可能な 3 リテラル式を生成する．(ii) 3PRV(0, N)-GEN は N を満足し，L(PRV(0)-GEN) に含まれる 3 リテラル式を生成する． □

3SAT(N)-GEN は入力列 r に従い 1 つの充足可能な例題を確実に生成することができる．それに対して，3PRV(0, N)-GEN は r によっては失敗（出力なしに停止）する可能性がある．3PRV(0, N)-GEN は，例題の生成過程でそれぞれのリテラル v の f_{now} の中の出現数が $N(v)$ 回を越えると v の削除を行なう．仮に，3 リテラルの項から v が削除されて，2 リテラルの項を含んでしまったとする．2 リテラルの項を 3 リテラルにするためには項の分解が必要であるが，それに伴い項の数も増えてしまい，必要な項数を超えてしまう可能性がある．一度必要な項数を超えると，項の削除は考えていないので，例題の生成に失敗をしてしまう．

失敗の可能性を下げるため，以下ように改良を加える．ここで，4リテラル以上の和項に含まれているリテラル v の個数を $N_{rmvable}(v)$ と表し，生成過程で f_{now} に含まれるリテラル v の個数を $N_{now}(v)$ と表す．

Generator REV-3PRV$(0, N)$-GEN:

ステップ1. ランダムに変数 x を1つ選び，$f_{now} = x\overline{x}$ とする．

ステップ2. f_{now} のすべての和項が3リテラルであればステップ4へ．そうでなければ，f_{now} の中から2リテラル以下の和項 A と $N(v) - N_{now}(v) \geq 1$ を満たすリテラル v をそれぞれ1つランダムに選ぶ．v のようなリテラルがないときは，例題を出力せずに停止する．A を $(A+x)(A+\overline{x})$ と分解して f_{now} を変更する．仮に，新しい f_{now} の中のリテラル u が $N(u) - N_{now}(u) < 0$ となったとき（つまり，$N_{now}(u) = N(u) + 1$ 回）は，u を含む2リテラル以上の和項 B を選び，B から u を削除する．u のようなリテラルすべてについて繰り返す．B のような和項がないときは，例題を出力せずに停止する．ステップ2を繰り返す．

ステップ3. (3-1) ～ (3-3) を任意に1つ選び実行する．

(3-1) f_{now} の中から $\sum_{v\in A}\{N_{rmvable}(v) + (N(v) - N_{now}(v))\} \geq 1$ であるような和項 A をランダムに選ぶ．A のような和項がなければステップ5へ．もしも，A が3リテラルで $N_{rmvable}(v) + (N(v) - N_{now}(v)) \geq 1$ を満たすリテラルが1つである場合，または，A が4リテラルで $\sum_{v\in A}\{N_{rmvable}(v) + (N(v) - N_{now}(v))\} = 4$ である場合は (3-1-1) へ．そうでなければ，(3-1-2) へ．

(3-1-1) $N(v) - N_{now}(v) \geq 1$ かつ $N_{rmvable}(\overline{v}) + (N(\overline{v}) - N_{now}(\overline{v})) \geq 1$ を満たすリテラル v をランダムに選ぶ．A を $(A + x)(A + \overline{x})$ と分解して f_{now} を変更する．仮に，新しい f_{now} の中のリテラル u が $N(u) - N_{now}(u) < 0$ となったとき（つまり，$N(u) + 1$ 回）は，u を含み4リテラル以上の和項 B を選び，B から u を削除する．u のようなリテラルすべてについて繰り返す．$N_{rmvable}(v)$ を計算してステップ4へ．v のようなリテラルがなければ，$\sum_{v\in A}\{N_{rmvable}(v) + (N(v) - N_{now}(v))\} \geq 2$ を満たす和項 C をランダムに選び，(3-1-2) へ．C のような和項がなければステップ5へ．

(3-1-2) $N(v) - N_{now}(v) \geq 1$ を満たすリテラル v をランダムに選ぶ．A を $(A+x)(A+\overline{x})$ と分解して f_{now} を変更する．仮に，新しい f_{now} の中のリテラル u が $N(u) - N_{now}(u) < 0$ となったときは，u を含み4リテラル以上の和項 B を選び，B から u を削除する．u のようなリテラルすべてについて繰り返す．$N_{rmvable}(v)$ を計算してステップ4へ．v のようなリテラルがなければステップ4へ．

(*3-2*) f_{now} の中から4リテラル以上の和項 A を選ぶ．A からリテラルをランダムに削除することにより A を3リテラルにする．

(*3-3*) f_{now} のすべての和項が3リテラルになればステップ5へ．

ステップ4. リテラル分布を満たし，すべての和項が3リテラルになるまでステップ3を繰り返す．

ステップ5. $N(v) - N_{now}(v) \geq 1$ を満たすリテラル v により3リテラルの和項をランダムに作り，f_{now} の1つの和項とする．リテラル分布 N を満足するまでステップ5を繰り返す． □

定理4. REV-3PRV$(0, N)$-GEN は N を満足し，L(PRV(0)-GEN) に含まれる3リテラル式を生成する． □

定理5. REV-3PRV$(0, N)$-GEN はすべてのリテラルがリテラル分布 N を越えることなく，すべての項を3リテラル以上にすることができる入力列 r に対して，1つの例題を確実に生成することができる． □

このようにREV-3PRV$(0, N)$-GEN は充足不能な例題生成の失敗の可能性を小さくすることができる．さらに，REV-3PRV$(0, N)$-GEN の出力集合 L(REV-3PRV$(0, N)$-GEN) が少なくともPでないことを示すことができる．

定理6. L(REV-3PRV$(0, N)$-GEN) はNP完全集合である． □

参考文献

[1] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, Vol. 17, No. 2, pp. 309–335, 1988.

[2] K. Iwama. Cnf satisfiability test by counting and polynomial average time. *SIAM J. Comput.*, pp. 385–391, 1989.

[3] K. Iwama, H. Abeta, and E. Miyano. Random generation of satisfiable and unsatisfiable CNF predicates. In *Proc. 12th IFIP World Computer Congress*, pp. 322–328, 1992.

[4] P. Purdom and C. Brown. The pure literal rule and polynomial average time. *SIAM J. Comput.*, pp. 943–953, 1985.