数理解析研究所講究録
第833巻 1993年 1-10

co-NP集合のNP集合による近似について

岩間一雄[†]

九州大学工学部情報工学科
iwama@csce.kyushu-u.ac.jp

1. Introduction

Let us say that a set L_1 in a class C_1 *approximates* a set L_2 in a class C_2 if L_1 is a subset of L_2. When both A and B approximate L, A is said to be *better than* B if $A \supset B$ and $A - B$ contains infinitely many elements. A is called *optimal* if no better approximation than A exists. For example, let L_{sat} be the set of all satisfiable CNF predicates, L_{neg-af} be the set of CNF predicates in which every clause includes both negative and affirmative literals. It is well known that L_{sat} is NP-complete. One can see easily that L_{neg-af} is in P and it approximates L_{sat}. It also turns out that L_{neg}, the set of CNF predicates in which every clause includes at least one negative literal, is a better approximation than L_{neg-af}. (The approximation is an independent notion from the so-called *subproblem*: The satisfiability problem for some restricted domain (say, for Horn predicates) is clearly easier than the same problem for general CNF predicates. In the case of the approximation, the class C_1 does not have to be an easier class than C_2, namely, for example, it does not loose any sense if C_1 is the family of NP-complete sets and C_2 is P.)

An apparent research goal is to seek better approximations in a different class, and that is not new: A set A not in P has been called *P-levelable* if for any set B in P which approximates A, there exists another B' in P which also approximates A and is better than B. Most natural sets that are NP-complete, PSPACE- complete and so on are known to be P-levelable (under the assumption that those sets are not in P), i.e., they have no optimal approximation (see the summary in [2]). In this paper, we discuss the case when C_1 is the class of NP-complete sets (NPC) and C_2 is that of coNP-complete sets (coNPC). Its motivation and significance are as follows:

In [6] the present author et al. introduced the *test-case generation with known answers* for the CNF satisfiability problem. Namely, our generator consists of two generators, one for yes-instances (satisfiable predicates) and the other for no-instances (unsatisfiable predicates). This allows us (i) to prepare test problems with their answers and (ii) to mix yes- and no-instances as we wish; both are very important in testing algorithms empirically. It turns out that polynomial-time generatable sets are in NP. The set of all yes-instances, L_{sat}, is in NP and can be generated in polynomial time in a relatively easy fashion (if we do not count the distribution). However, the set of no-instances, $L_{\overline{sat}}$, is in coNPC and cannot be generated in polynomial time unless NP = co-NP. Thus we probably should try to generate not the whole

[†]Supported by Scientific Research Grant, Ministry of Education, Japan, No. 02302047 and No. 04650318.

$L_{\overline{sat}}$ but its subset S. Clearly S should be as large as possible and must not be an easy subset like one in P. The latter could be met by selecting S as an NPC set. Thus our primary goal is to seek a good NPC approximation of $L_{\overline{sat}}$.

The first result in this paper is that as with the P-levelability mentioned above, there is no optimal NPC approximation of $L_{\overline{sat}}$. This is a somewhat good news for our motivation since it appears that we can get a better and better approximation without limit. Unfortunately that is not a direct consequence of the result since its proof is nonconstructive. That is the reason why the following second result makes sense: It is shown that there are an infinite sequence of (explicit) sets $L_0, L_1, \cdots, L_i, L_{i+1}, \cdots$ such that each L_i is in NPC, each L_i approximates $L_{\overline{sat}}$ and L_{i+1} is better than L_i for each i. We will also conjecture that a slight modification of this construction results in a constructive version of the first result, namely, that for any NPC approximation of $L_{\overline{sat}}$, we can construct effectively a better approximation.

As mentioned previously, it is expected that the present results will contribute to more appropriate generation of test-cases or languages in general. Related works are as follows: Sanchis et al. [11] shows that although most P and NP languages appear to have polynomial-time generators, the question of whether all NP languages have such generators cannot be resolved without answering some major open questions in complexity theory. One interesting point is that the difficulty for NP languages is the same as that for P languages in the case of generation. There are many other papers which mainly focuses on the distribution of generated instances, see [5, 3, 1]. Note that any of those works seldom mentions the generation of co-NP languages. As for optimization problems, see [10, 8]. See [13] for generation of several kind of graphs. Also recent papers [9][12] have investigated both experimentally and theoretically that what kind of SAT instances are really hard (every SAT algorithm takes exponential time almost always). Their claim is, not surprisingly, that if the instances are generated syntactically at random then those hard ones exist around such parameter values that the probability of satisfiability is 50%.

2. Limitlessness of Better Approximation

Although most description in this section will be made by means of SAT, the statements themselves are not solely restricted to that particular problem. A *literal* is a logic variable x or its negation $\overline{x}$. A *clause* is a sum of literals. A (*CNF*) *predicate* is a product of clauses. A specific assignment of *true* and *false* to all variables is called a *cell*. A predicate f is said to be *satisfiable* (*unsatisfiable*) if there is a (no) cell that makes f *true*. It is said that a clause A *covers* a cell T if the assignment denoted by T makes A *false*. A clause A is said to *cover* a clause B if all the cells covered by B are covered by A. (Intuitively, A is larger than B.)

In proving our main theorem, we need mappings like the padding function [2]. Consider,

for example, the following δ and ψ that are defined for predicates f of n variables $x_1, \cdots, x_n$: Note that (x_{n+1}) and (x_{n+2}) below are clauses including a single literal.

$$\delta(f) = \begin{cases} f \cdot (x_{n+1}) & \text{if } n \text{ is even} \\ f \cdot (x_{n+1})(\overline{x_{n+2}}) & \text{if } n \text{ is odd} \end{cases}$$

$$\psi(f) = \begin{cases} f \cdot (x_{n+1}) & \text{if } n \text{ is odd} \\ f \cdot (x_{n+1})(\overline{x_{n+2}}) & \text{if } n \text{ is even} \end{cases}$$

Remark. δ has the following properties: As will be seen, they are important in the sense that our later theorems hold for any specific mapping with these properties. Similarly for ψ.

(i) δ is one-to-one.

(ii) δ is computable in polynomial time.

(iii) δ^{-1} is computable in polynomial time including the case that there is no predicate g such that $g = \delta^{-1}(f)$, which is denoted by $\delta^{-1}(f) = \Lambda$.

(iv) The size of $\delta(f)$ is larger than f.

(v) δ and ψ are *disjoint*, i.e., for any sets A and B of predicates, $\delta(A) \cap \psi(B) = \emptyset$.

Before presenting our main theorem (NPC approximation of coNP sets), we shall first review the result on the P-levelability (P approximation of harder sets) and make a slight extension of that.

Proposition 1. Let A be a set of predicates that is not in P and is closed under the mapping ψ. (The predicates may be simply strings and the mapping may be any one having the properties (ii) and (iv).) Then any P approximation B of A is not optimal.

Proof [2]. Suppose that B is optimal. A set C is defined as follows:

$$C = \{f \mid f \notin B \text{ but } \psi(f) \in B\}$$

As B is in P, and ψ is computable in polynomial time, C is also in P. Now consider, for each f in A, the following sequence: f, $\psi(f)$, $\psi^2(f) = \psi(\psi(f))$, $\cdots$. Since ψ is closed under A, each value is in A. Now the set

$$D_f = \{f, \psi(f), \psi^2(f), \cdots, \psi^n(f), \cdots\}$$

is infinite and is in P since ψ makes the size strictly larger. Because B was assumed to be optimal, D_f must eventually be contained in B; it must "cross" the boundary denoted by the set C.

As A is not in P, $A - B$ is infinite. For each f in $A - B$, the sequence D_f reaches some member g of C and the size of g is larger than that of f. Thus only finitely many f may reach any fixed g, and hence there must be infinitely many such g. Thus C is infinite. Since $C \cap B = \emptyset$ by the definition of C, $C \cup B$ contains infinitely more elements than B and is still in P since both C and B are in P, which contradicts the assumption that B is optimal. □

Proposition 2. Let A be a set of predicates that is not in NP and is closed under ψ. Then any NP approximation B of A is not optimal.

Proof. Almost the same as above but we define C as

$$C = \{f \mid \psi(f) \in B\}$$

Then C is in NP and so is $C \cup B$ as well. (If we would add the condition $f \notin B$ then we could no longer claim that C is in NP.) Although C and B are no longer disjoint, we can show that $C \cup B$ is infinitely larger than B by the similar argument as above. □

Thus it was an easy extension of of Proposition 1 to prove the limitlessness of the better NP approximation. As one can see, the key point in both proofs is to find the way of still expanding the optimal approximation B. The only condition in doing so is that the extension must be made within A, which is guaranteed by the mapping ψ since it is closed within A.

Now what would happen if the NP approximation is replaced by the NPC approximation? Since B is NP-complete, there are another NP-complete set X and a polynomial-time reduction α such that $B \supseteq \alpha(X)$ and $\overline{B} \supseteq \alpha(\overline{X})$. Then if we would try to expand B as above, we have to pay attention to this set $\alpha(\overline{X})$ since if the expansion of B would imply an overlap between B and $\alpha(\overline{X})$, then the NP-completeness of B would no longer be claimed. It should be noted that unlike A (i.e., particular set $L_{\overline{sat}}$), we have no specific information about this $\alpha(\overline{X})$.

Theorem 1. Suppose that A is a coNP set of predicates and B is an NPC approximation of A such that both A and $\overline{A}$ are closed under ψ and $\overline{B}$ is closed under δ. (Again ψ can be any such mapping satisfying (ii) and (iv) and similarly for δ if it satisfies (i) to (v).) Then if NP $\neq$ coNP then B is not optimal.

Proof. Since B is NP-complete, there is another NP-complete set X and a polynomial-time reduction α such that $B \supseteq \alpha(X)$ and $\overline{B} \supseteq \alpha(\overline{X})$. the first case to be considered is that there are such X and α that $\alpha(\overline{X})$ is in $\overline{A}$. (In this case, two sets B and $\overline{A}$ are P-inseparable [4]. Recall that $\overline{A}$ (=L_{sat}) can be perfectly generatable and B is probably so too since B is in NP. Then, although details are omitted, we were able to construct the so-called one-way function, which no one has ever succeeded in. Thus, this first case seems to be a little unrealistic.) This case is easy to treat since we can apply the same expansion of B as Proposition 2. Clearly we do not have to care about an overlap of B with $\alpha(\overline{X})$ because the latter is outside A.

Hence, from now on, we can assume that there are no such α and X or that any pair of the set X and the reduction α satisfies that $\alpha(\overline{X}) \cap A \neq \emptyset$. Let $C = \alpha(\overline{X}) \cap A$.

Now we show the following important property about this set C: Let

$$C_0 = \{\delta^{-\infty}(f) \mid f \in C\}$$

where $g = \delta^{-\infty}(f)$ iff $\delta^{-1}(g) = \Lambda$ and for some $i \geq 0$, $\delta^i(g) = f$. Then C_0 is infinite by the following reason: Suppose hypothetically that C_0 is finite. Then given f in $\alpha(X \cup \overline{X})$, we can

decide whether f is in C or not in polynomial time. (One can apply δ^{-1} to f repeatedly, which must end shortly since δ^{-1} makes f shorter. If f is in C then the sequence must end up in C_0. It may do so as well if f is in $\overline{A}$ but never if f is in $\alpha(X)$ because of the closure property of δ.) Now we can modify the reduction α into the following α': If $\delta^{-\infty} \cdot \alpha(x)$ is not in C_0 then $\alpha'(x) = \alpha(x)$. Otherwise $\alpha'(x)$ is some trivial element in $\overline{A}$. One can easily see that α' is also polynomial-time computable and it can act as a reduction just like α. But this time $\alpha'(\overline{X}) \subseteq \overline{A}$ or $\alpha'(\overline{X}) \cap A = \emptyset$, which contradicts our current assumption.

Now let $D = \delta(C)$ and

$$E = \{f \mid \delta^{-1}(f) \in B, \text{ or } \delta^{-1}(f) = \Lambda \text{ and } \psi(f) \in B\}$$

(i) $E \subseteq A$ by the closure property of δ and ψ.

(ii) Since B is in NP and δ^{-1} and ψ are polynomially computable functions, E is also in NP.

(iii) $E \cap D = \emptyset$. (Reason: If f is in D, $\delta^{-1}(f)$ must be in C by the definition of D. Either $\delta^{-1}(f) \in B$ or $\delta^{-1}(f) = \Lambda$ contradicts this fact because B and C are disjoint.)

(iv) The difference $C - D$ is infinite. Otherwise the above C_0 must be finite.

(v) The difference $E - B$ is infinite: Apply δ^{-1} to each f in the infinite $C - D$ repeatedly. Then it eventually reach such g_1 or g_2 that $\delta^{-1}(g_1)$ is in B for the first time and $\delta^{-1}(g_2) = \Lambda$. Since $C - D$ is infinite and δ is one-to-one, the set of those g_1 and g_2 is also infinite. If the set of g_1 is infinite then so is $E - B$ since such g_1 exists in $E - B$. Hence suppose that the set of g_2 is infinite. Then we can use the argument of Proposition 2 considering the sequence $\psi(g_2)$, $\psi^2(g_2)$, $\cdots$, to show that $E - B$ is infinite. It should be noted that this sequence never goes into D because of the property (v) of δ and ψ.

Now we obtained $E \cup B$ which is a subset of A ((i) above) and is infinitely larger than B ((v) above). It remains to show that $E \cup B$ is NPC: We modify α into $\delta \cdot \alpha$ that is clearly polynomial-time computable. $\delta \cdot \alpha(\overline{X})$ includes D and maybe some members in $\overline{A}$, which does not overlap with E by (iii) or with B by the closure property of δ. Also $\delta \cdot \alpha(X)$ must be contained in $E \cup B$ since $\alpha(X)$ is in B and E contains such f that for some g in B, $\delta(g) = f$. □

It should be noted again that the proof does not depend on any specific property of SAT, δ or ψ. For many other natural problems, it does not seem to be so hard to find δ and ψ that satisfy the condition (i) to (v) and the closure property described in the theorem.

3. Sequence of Better Approximations.

Suppose that we have an NPC approximation A of $L_{\overline{sat}}$. Then Theorem 1 says that there does exist another NPC approximation B that is better than A but says nothing about how to get B. At present we cannot find such a method, but can show an infinite sequence of better and better approximations each of which is constructed effectively.

A CNF predicate f is said to be *provable* if for some variable x, $x\bar{x}$ is implied from f by applying the following three operations repeatedly:

(i) An arbitrary literal is added to an arbitrary clause A.

(ii) Two clauses $(A + x)(A + \bar{x})$ are merged into a single clause A.

(iii) An arbitrary clause A is duplicated to AA.

Example. Let $f = (\overline{x_1} + \overline{x_2})(x_1 + \overline{x_3})(x_2 + x_3)(\overline{x_1} + x_2 + \overline{x_3})(x_1 + \overline{x_2} + x_3)$. Then f is provable as follows: By operation (iii), $(x_2 + x_3)$ is duplicated. $\overline{x_1}$ is added to this $(x_2 + x_3)$ and then merged with $(\overline{x_1} + x_2 + \overline{x_3})$ to imply $(\overline{x_1} + x_2)$. x_1 is added to the other $(x_2 + x_3)$ and then merged with $(x_1 + \overline{x_2} + x_3)$ to get $(x_1 + x_3)$. Now f became $(\overline{x_1} + \overline{x_2})(x_1 + \overline{x_3})(\overline{x_1} + x_2)(x_1 + x_3)$. By merging the 1st and 3rd and also the 2nd and 4th, we get $\overline{x_1}x_1$.

Note that among the three operations, only the third one increases the number of clauses. Let $d(n)$ be a function in an integer n. Then $PRV(d(n))$ denotes the set of predicates f such that f is provable by using operation (iii) at most $d(n)$ times where n is the number of clauses in f. Then it is not hard to see that:

(a) $PRV(d(n))$ approximates $L_{\overline{sat}}$ for any $d(n)$.

(b) $PRV(0)$ is in NP.

(c) $PRV(d(n))$ is still in NP if $d(n)$ is a polynomial.

Further results are shown in [6, 7]:

(d) $PRV(d(n))$ can be generated in polynomial time if $d(n)$ is a polynomial. (The generation is simply a reverse of the procedure of proving given above.)

(e) $PRV(2^{O(n)}) = L_{\overline{sat}}$.

Our main results in this section are:

Theorem 2. For any polynomial $d(n)$, $PRV(d(n))$ is NP-complete.

Theorem 3. If d_1 and d_2 are polynomials such that $d_1(n) < d_2(n)$ for all $n \geq 1$, then $PRV(d_2(n))$ is better than $PRV(d_1(n))$.

Thus, for example, $PRV(n)$, $PRV(n+1)$, $\cdots$, $PRV(n+i)$, $\cdots$ is an infinite sequence of better NPC approximations of $L_{\overline{sat}}$. For a set L of predicates, let $PRV(d(n), L)$ denote the set of predicates f such that f can be reduced to some predicate g in L by the three operations (at most $d(n)$ times for (iii)).

Open Question. Does there exist a polynomial $d(n)$ such that for any NPC approximation A of $L_{\overline{sat}}$, $PRV(d(n), A)$ is a better NPC approximation than A?

4. Proofs of Theorems 2 and 3 (Sketch).

We first show a key lemma for both theorems. (Note that the property (e) of the previous section is an old result in the field of switching theory. The following lemma demonstrates that there is a predicate for which we need an exponential number of operations to get only one prime implicant.)

Lemma 1. We can construct a predicate f of t clauses such that exactly $q(t) = \Theta(c^t)$ (for some $c > 1$) applications of the duplication operations are needed to show its provability.

Proof. The predicate f consists of the following (1) to (3).

$$U(n)(1,2)U(n-1)(\overline{1},\overline{2})U(n-2)(3,4)U(n-3)(\overline{3},\overline{4})\cdots U(2)(n-1,n)U(1)(\overline{n-1},\overline{n}) \tag{1}$$

$$V(n)(1,\overline{2})V(n-1)(\overline{1},2)V(n-2)(3,\overline{4})V(n-3)(\overline{3},4)\cdots V(2)(n-1,\overline{n})V(1)(\overline{n-1},n) \tag{2}$$

$$\begin{aligned}
&W_1(n)(1,\overline{2})W_1(n-1)(\overline{1},2)W_1(n-2)(3,4)W_1(n-3)(\overline{3},\overline{4})\cdots W_1(2)(n-1,n)W_1(1)(\overline{n-1},\overline{n})\\
&W_2(n)(1,2)W_2(n-1)(\overline{1},\overline{2})W_2(n-2)(3,\overline{4})W_2(n-3)(\overline{3},4)\cdots W_2(2)(n-1,\overline{n})W_2(1)(\overline{n-1},n)\\
&W_3(n-2)(3,\overline{4})W_3(n-3)(\overline{3},4)W_3(n-4)(5,6)W_3(n-5)(\overline{5},\overline{6})\cdots W_3(2)(n-1,n)W_3(1)(\overline{n-1},\overline{n})\\
&W_4(n-2)(3,4)W_4(n-3)(\overline{3},\overline{4})W_4(n-4)(5,\overline{6})W_4(n-5)(\overline{5},6)\cdots W_4(2)(n-1,\overline{n})W_4(1)(\overline{n-1},n)\\
&\qquad\vdots\\
&W_{n-3}(4)(n-3,\overline{n-2})W_{n-3}(3)(\overline{n-3},n-2)W_{n-3}(2)(n-1,n)W_{n-3}(1)(\overline{n-1},\overline{n})\\
&W_{n-2}(4)(n-3,n-2)W_{n-2}(3)(\overline{n-3},\overline{n-2})W_{n-2}(2)(n-1,\overline{n})W_{n-2}(1)(\overline{n-1},n)
\end{aligned} \tag{3}$$

where $U(i)(k,k+1) = (U(i)+z_k)(U(i)+z_{k+1}), U(i)(k,\overline{k+1}) = (U(i)+z_k)(U(i)+\overline{z_{k+1}})$, and similarly for $U(i)(\overline{k},k+1)$ and $U(i)(\overline{k},\overline{z_{k+1}})$. Furthermore, $U(1), U(2), \cdots, U(n-1), U(n)$ are u_1, $\overline{u_1}+u_2$, $\cdots$, $\overline{u_1}+\overline{u_2}+\cdots+\overline{u_{(n-2)}}+u_{(n-1)}$, $\overline{u_1}+\overline{u_2}+\cdots+\overline{u_{(n-2)}}+\overline{u_{(n-1)}}$, respectively. Namely,

$$(F+U(1))(F+U(2))\cdots(F+U(n)) \tag{4}$$

can be modified to F (a sum of literals) by repeated merge operations. Similarly for $V(i)$ and $W_j(i)$. Variables $z_1, \cdots, z_n$ are called *main-variables* and those in $U(i)$, $V(i)$ and $W_j(i)$ are called *sub-variables.*

We first show that f is provable. Removing the sub-variables of (1) as above, we get $2^{n/2}$ clauses of the following form

$$x_1+x_2+x_3+x_4+\cdots+x_{2i-1}+x_{2i}+\cdots+x_{n-1}+x_n, \tag{5}$$

where $(x_{2i-1}, x_{2i}) = (\overline{z_{2i-1}}, z_{2i})$ or $(z_{2i-1}, \overline{z_{2i}})$. It should be noted that we need an exponential number of duplication operations in this process. Similarly we get $2^{n/2}$ ones from (2) like

$$y_1+y_2+y_3+y_4+\cdots+y_{2i-1}+y_{2i}+\cdots+y_{n-1}+y_n, \tag{6}$$

where $(y_{2i-1}, y_{2i}) = (z_{2i-1}, z_{2i})$ or $(\overline{z_{2i-1}}, \overline{z_{2i}})$. Now, from the first line of (3), we can imply $2^{n/2}$ clauses like

$$z_1 + z_2 + x_3 + x_4 + \cdots + x_{n-1} + x_n, \quad \text{and}$$
$$\overline{z_1} + \overline{z_2} + x_3 + x_4 + \cdots + x_{n-1} + x_n, \tag{7}$$

where $x_3 + x_4 + \cdots + x_{n-1} + x_n$ is exactly the same as (5). Thus we can combine (5) and (7) to imply all the predicates denoted by

$$x_3 + x_4 + \cdots + x_{n-1} + x_n. \tag{8}$$

Similarly, we can get

$$y_3 + y_4 + \cdots + y_{n-1} + y_n \tag{9}$$

from (6) and the second line of (3). Then using the third line of (3), x_3 and x_4 are removed from (8). Also y_3 and y_4 are removed from (9) using the forth line of (3). We can continue this procedure until both

$$x_{n-1} + x_n \quad \text{and} \quad y_{n-1} + y_n$$

are implied, from which, as one can easily see, $z_n\overline{z_n}$ can be implied.

We shall next show that what we did above is the only possibility of proving f. As an easier case, suppose that (1) is replaced by $2^{n/2}-1$ clauses, say (1'), that consist of all the clauses denoted by $x_1+x_2+\cdots+x_n$ except a single (any) one, which we denote by $z'_1+z'_2+\cdots+z'_n$. Then we can show that the set of clauses (1'), (2) and (3) is satisfiable: Set values to $z_1, z_2, \cdots, z_n$ as follows: If $(z'_{2i-1}, z'_{2i}) = (\overline{z_{2i-1}}, z_{2i})$ then $z_{2i-1} = 1$, $z_{2i} = 0$. If $(z'_{2i-1}, z'_{2i}) = (z_{2i-1}, \overline{z_{2i}})$ then $z_{2i-1} = 0$, $z_{2i} = 1$. Then all the other $2^{n/2} - 1$ clauses then this $z'_1 + z'_2 + \cdots + z'_n$ clearly become 1. In terms of (2), either $V(n)(1, \bar{2}) = (V(n) + z_1)(V(n) + \overline{z_2})$ or $V(n-1)(\bar{1}, 2) = (V(n-1) + \overline{z_1})(V(n-1) + z_2)$ becomes 1 by the above assignment regardless of the values of the sub-variables in $V(n)$ and $V(n-1)$. Therefore, if for example the former is 1, then all the other clauses, $V(n-1)(\bar{1}, 2), V(n-2)(3, \bar{4})$, and so on can be made 1 by assigning proper values only to the sub-variables in $V(n-1), V(n-2)$, and so on (namely, $v_{n-1} = v_{n-2} = \cdots = v_1 = 1$). As for the first line of (3), either $W_1(n)(1, \bar{2})$ or $W_1(n-1)(\bar{1}, 2)$ becomes 1 regardless of $W_1(n)$ or $W_1(n-1)$, and all the other clauses can be made 1 only by properly setting $W_1(n-2), \cdots$. Similarly for the second line (either $W_2(n-2)(3, \bar{4})$ or $W_2(n-3)(\bar{3}, 4)$ becomes 1) and for the rest of (3).

This argument also holds for (2) and for each line of (3). Namely, if we try to delete sub-variables before main-variables, we have to do so by expanding each line into all the possible clauses of z_i's. On the other hand, considering that sub-variables are able to be deleted only by the standard expansion, it turns out that to try to combine, say $U(n-2)(3,4)$ and $V(n-2)(3, \bar{4})$ before deleting the sub-variables does not help to our goal. Therefore, if we could prove the

predicate using only a polynomial number of the duplication operation, it would be only by: (i) We first expand (completely) the last several (not too many) lines of (3). (ii) Then using those clauses not including sub-variables we try to delete main-variables of the rest of the lines before deleting sub-variables. This, however, is impossible since the process intuitively means that we try to delete main-variables from z_n towards z_1, which is the opposite direction compared to the successful proof (but using exponential delete operations) described before. We omit the details which need rather tedious case-analysis. □

Lemma 2. There is a predicates of 5 clauses that is provable if we can use the duplication operation once but is not provable only by (i) and (ii).

Proof. The predicate shown in the example of Section 3 meets the condition. □

From these two lemmas, the following lemma is then implied:

Lemma 3. For any polynomial $d(t)$, we can construct a predicate f of $O(t^{1/2})$ clauses such that we need exactly $d(t)$ applications of the duplication operation to show its provability. (Proof is omitted.)

Lemma 4 [7]. There is a polynomial-time reduction α from 3-CNF predicates into CNF predicates such that $\alpha(f)$ is in $PRV(0)$ iff f is satisfiable and $\alpha(f)$ includes $O(t^2)$ clauses if f has t clauses. (Namely $PRV(0)$ is NP-complete.)

Now it is not hard to complete the proof for Theorem 2. We design a similar reductoin as Lemma 4: For a 3-CNF predicate g_0 of t clauses, we first set a sufficiently large number , say $n = t^3$. By Lemma 3, we then a predicate g_1 such that exactly $d(n)$ duplication operations are needed to imply a single literal, say σ. (This can be done by replacing each clause A in f of lemma 3 by $(A = \sigma)$.) g_1 contains $O(t^{1.5})$ clauses. We then construct g_2 such that we can imply $\overline{\sigma}$ fromg_2 only by operations (i) and (ii) iff the 3-CNF predicate g_0 is satisfiable. Now the target predicate g is $g_1 g_2 g_3$ where g_3 is a trivial (satisfiable) predicate just to adjast the number of clauses of g to n. One can see that g is in $PRV(d(n))$ iff g_0 is satisfiable.

Theorem 3 is almost trivial by Lemma 3.

Acknowledgement. The author would like to thank Jose Balcazar for his suggestion of possible extensions of P-levelability to the current form.

References

[1] M. Abadi, E. Allender, A. Broader, J. Feigenbaum, and A. Hemachandra. On generating solved instances of computational problems. In *Proc. CRIPT88*, 1988.

[2] J. Balcazar, J. Diaz, and J. Gabarro. *Structural Complexity II.* Springer, 1989.

[3] S. Ben-David, B. Chor, O Goldreich, and M. Luby. On the theory of average case complexity. In *Proc. 21st ACM Symp. on Theory of Computing*, pp. 204–216, 1989.

[4] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with application to public-key cryptography. *Information and Control*, Vol. 61,, 1984.

[5] R. Impagliazzo and L. Levin. No better ways to generate hard np instances than picking uniformly at random. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pp. 812–821, 1990.

[6] 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.

[7] K. Iwama and E. Miyano. Security of test-case generation with known answers. In *Proc. AAAI Spring Symposium Series*, 1993.

[8] B. Krishnamurthy. Constructing test cases for partitioning heuristics. *IEEE Trans. Comput.*, Vol. 36, pp. 1112–1114, 1987.

[9] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. Tenth National Conference on Artificial Intellegence*, pp. 459–465, 1992.

[10] L. Sanchis. On the complexity of test case generation for NP-hard problems. *Inform. Process. Lett.*, Vol. 36, pp. 135–140, 1990.

[11] L. Sanchis and M. Fulk. On the efficient generation of language instances. *SIAM J. Comput.*, Vol. 19, pp. 281–296, 1990.

[12] V. Shvatal and E. Szemeredi. Many hard examples for resolution. *J. Assoc. Comput. Mach.*, Vol. 35, pp. 759–768, 1988.

[13] G. Tinhofer. Generating graphs uniformly at random. In *Computational graph theory*, pp. 235–255. Springer, 1990.