数理解析研究所講究録
第 871 巻 1994 年 37-44

木状 Hajós Calculus の非多項式時間限定性

九州大学工学部　岩間　一雄 (Kazuo Iwama)
iwama@csce.kyushu-u.ac.jp

1 Introduction

Frege systems are a very powerful class of proof systems for the propositional calculus, and (hence) it is believed that to prove their non-polynomial lower bounds is hard. Recently there was a breakthrough; Pitassi and Urquhart [PU92] proved that Frege systems can be simulated (efficiently) by a *simpler* graph calculus called the Hajós calculus, in such a way that the simulation guarantees that if the latter is not polynomially–bounded then the former is not either. Unfortunately the calculus is still not sufficiently simple; it has long been an open problem if the Hajós calculus is polynomially–bounded. In this paper, we cannot prove that the Hajós calculus is poly-bounded but we prove its important subclass is poly-bounded. We call the subclass the tree-like Hajós calculus.

Suppose that our final goal is to prove that NP≠co–NP. Then a reasonable subgoal is to consider some proof or generation system, S, for co-NP languages, L, for example, the set of unsatisfiable predicates or the set of non–3–colorable graphs, and to prove that S is not polynomially–bounded. By ***polynomially-bounded***, we mean that any element in L can be generated in a polynomial number of steps. Obviously the system S should be as powerful as possible; if it were as powerful as nondeterministic Turing machines, then we would have achieved the final goal. In this sense, Frege systems have drawn a lot of researchers' interests and several non-polynomial lower bounds have been obtained for bounded cases [Coo75, CR79, Ajt88, BIK$^+$92, for instance]. However, for unbounded cases, their great power did not allow to develop useful techniques for proving the lower bounds.

It is quite natural then to seek for simpler (seemingly less powerful) generation systems that can simulate Frege systems with polynomial overhead. [PU92] found out the Hajós calculus to be such a generation system. The Hajós calculus [Haj61] (HC for short) is a procedure for generating non–3–colorable graphs. (In this paper we only consider the 3–color case although HC can treat k–color cases in general.) It starts with a K_4 as its set of graphs and get new (more complicated) graphs by applying one in some set of generation rules repeatedly. A little surprisingly, this relatively simple generation system is ***complete***, namely, it can generate any non–3–colorable graphs.

The tree-like HC (TLHC) is different from the general HC in that copying graphs is not allowed. Namely, every new graph must be generated from K_4's using a tree-structured proof. Therefore if we try to simulate a generation procedure of the original HC by TLHC in the trivial way, then the latter can blow up to an exponential size even though the former is poly-bounded. However, the current result still seems to have a certain merit: (i) TLHC is still complete and hence the result seems to be a nice progress to the final goal. (ii) Although details have not yet been examined, we conjecture that HLHC is p-equivalent to resolution, which appears to be a nice comparison to that the general Hajos Calculus is p-equivalent to Extended Frege.

Iwama, Abeta and Miyano proposed another very simple generation system for unsatisfiable CNF predicates [IAM92], which we shall call USG. Similarly as HC, USG begins with an initial predicate like $x\overline{x}$ and applies one of the four rules (see Sec. 3) step by step. (Our motivation of introducing USG was the generation of test problems to evaluate experimentally the performance of SAT algorithms. See [IAM92, IM93] for that aspect and also [Kuc91, CR92, Sanar] for other projects of related test-case generation.)

2 Hajós Calculus

All graphs in this paper are simple, undirected graphs without self–loops. A graph G is (V, E), where V is a set of *vertices* and E a set of unordered pairs of vertices, called *edges*. A graph G may be a collection of mutually disconnected (disjoint) *components*. A (*proper*) *k-coloring* of G is an assignment of one of k different colors to each of the vertices such that no two adjacent vertices receive the same color. If such a coloring exists, then G is said to be *k–colorable* and *non–k–colorable* otherwise. K_4 is the complete graph, i.e., every pair of vertices is adjacent, of four vertices.

TLHC starts with the initial graph K_4 and changes it by applying one of the following rules repeatedly. This definition is taken from [PU92] although slightly modified.

(i) (K_4 introduction) Add a K_4 as a new component of the current graph G_{now}.

(ii) (Vertex/edge introduction) Add any number of vertices and edges to G_{now} unless those added vertices and edges constitute new disjoint components. In other words, newly added vertices must be connected to some existing component.

(iii) (Join) Let G_1 and G_2 be disjoint components of G_{now}, a and b adjacent vertices in G_1 and c and d adjacent vertices in G_2. Then remove edges (a, b) and (c, d); then add an edge (b, d); finally contract vertices a and c into a single vertex. See Fig. 1.

(iv) (Contraction) Contract two nonadjacent vertices into a single vertex, and remove any resulting duplicated edges.

An essential difference between TLHC and HC is in the join rule: In TLHC, both G_1 and G_2 must be existing components but in the general HC, we can copy G_1 andor G_2 from the existing components. It is known [Haj61, MW] that any component of graph G generated by TLHC is non–3–colorable and any non–3–colorable graph can be generated by TLHC. If a graph G of size n can be generated in $p(n)$ applications of the rules for some polynomial $p(n)$, then it is said that G is *generated in polynomial time*. If every non–3–colorable graph is generated in polynomial time, TLHC is said to be *polynomially–bounded.*

3 Unsatisfiable Predicate Generator

A *literal* is a logic variable x or its negation $\overline{x}$. A *clause* is a sum of (one or more) literals. In this paper, a single clause cannot contain two or more same literals or both positive and negative literals of the same variable. A (*CNF*) *predicate* is a product of clauses. A specific assignment of *true* and *false* to all variables is called a *cell*. It is said that a clause A *covers* a cell T if the assignment denoted by T makes A *false*. A predicate F is said to be *satisfiable* (*unsatisfiable*) if there is at least one (no) cell which is not covered by any clause in F. A clause A is said to *cover* a clause B if all the cells covered by B are covered by A. It is straightforward to see that A covers B iff B includes all the literals of A and possibly more.

USG has the same structure as TLHC, which begins with the initial predicate $x_1\overline{x_1}$ and has the following rules.

(i) (Clause introduction) Add any clause to the current predicate F_{now}.

(ii) (Split) Replace a clause A by $(A + x_i)(A + \overline{x_i})$ for some variable x_i not appearing in A.

(iii) (Literal deletion) Delete any (single) literal in a clause which includes two or more literals.

(iv) (Clause deletion) Delete a clause A if A is covered by some other clause.

Lemma 1. If a predicate F is generated by USG then F is unsatisfiable.

Lemma 2. Any unsatisfiable predicate can be generated by USG.

Expressions such as "generated in polynomial time" and "polynomially–bounded" are also used for USG and have exactly the same meaning as before.

4 Simulation of TLHC by USG

4.1 Outline

We first define two transformations, P_r from graphs into predicates and G_r from predicates into graphs, as in [PU92]. For a graph G of n vertices $\{v_1, \cdots, v_n\}$ and m edges, $P_r(G)$ is the following predicate F consisting of $5n + 3m$ clauses.

(i) F uses $3n$ variables $R_{v_1}, \cdots, R_{v_n}, B_{v_1}, \cdots, B_{v_n}, G_{v_1}, \cdots, G_{v_n}$. ($R$, B and G stands for red, blue and green, respectively.)

(ii) For each vertex v, we introduce five clauses $(\overline{R_v} + \overline{B_v} + \overline{G_v})$ $(R_v + B_v + G_v)$ $(\overline{R_v} + \overline{B_v} + G_v)$ $(\overline{R_v} + B_v + \overline{G_v})$ $(R_v + \overline{B_v} + \overline{G_v})$.

(iii) For each edge (u, v), we introduce three clauses $(\overline{R_u} + \overline{R_v})$ $(\overline{B_u} + \overline{B_v})$ $(\overline{G_u} + \overline{G_v})$.

Conversely, for a predicate F of n variables $\{x_1, \cdots, x_n\}$ and t clauses, $G_r(F)$ is the graph G as illustrated in Fig. 2. Namely G consists of a single triangle of v_1, v_2 and v_3, n triangles of v_3, x_i and $\overline{x_i}$ for $i = 1, \cdots, n$, and t subgraphs each of which is associated with each clause. In the figure, only one such subgraph associated with a clause $(l_1 + l_2 + \cdots + l_q)$ (each l_i is x_j or $\overline{x_j}$) is drawn. Note that F is unsatisfiable iff $G_r(F)$ is non–3–colorable [GJS76].

Recall that we wish to prove the following: If any non–3–colorable graph can be generated in polynomial time by TLHC then any unsatisfiable predicate can also be generated by USG in polynomial time. Let F be an unsatisfiable predicate we wish to generate. The outline of our generation procedure is as follows: (1) Obtain $G_r(F)$. (2) Since $G_r(F)$ is non–3–colorable, there is a polynomially long sequence G_0, G_1, $\cdots$, $G_p = G_r(F)$ of generation by TLHC. (3) Simulate this by USG, namely, construct the following sequence of generation by USG:

$$F_0 = x_1\overline{x_1},\ S_{init},\ P_r(G_0),\ S_{0,1},\ P_r(G_1),\ S_{1,2},\ \cdots,$$
$$P_r(G_i),\ S_{i,i+1},\ P_r(G_{i+1}),\ \cdots,\ S_{p-1,p},\ P_r(G_p) = P_r(G_r(F)),\ S_{fin},\ F$$

where S_{init} is not a single predicate but a sequence of predicates gradually changing from F_0 to $P_r(G_0)$. Since $P_r(G_0)$ is a fixed predicate (determined by the initial graph of TLHC) and USG is complete, this sequence must be finite. $S_{i,i+1}$ is also a sequence of predicates from $P_r(G_i)$ to $P_r(G_{i+1})$. Recall that G_{i+1} is obtained from G_i by applying one of the four TLHC's rules. What we have to prove is that the length of this sequence $S_{i,i+1}$ is not too long, namely, that a single application of the TLHC's rules can be simulated by a polynomially long sequence of applications of USG's rules in the transformed predicates. Finally we also have to show that S_{fin}, which is needed to change $P_r(G_r(F))$ to F for any unsatisfiable F in general, is polynomially long.

However, we shall not try to change, say, $F_i = P_r(G_i)$ to $F_{i+1} = P_r(G_{i+1})$, in a straightforward manner (that means applying some USG rule to F_i, getting $F_{i,1}$, then applying a rule again to $F_{i,1}$, getting $F_{i,2}$ and so on). Instead, we shall try to get F_{i+1} directly from the initial predicate $x_1\overline{x_1}$ by making full use of the fact that F_i is generated from $x_1\overline{x_1}$ in polynomial time. The next subsection gives us several convenient lemmas for this strategy.

4.2 Useful Properties of USG

Let F be a predicate. Then $F[x_i = 1]$ is a predicate obtained by the following *substitution* operation (that is independent from USG). (1) Delete any clause that includes literal x_i. (2) Delete all the occurrence of literal $\overline{x_i}$. If some clause becomes empty as a result, then $F[x_i = 1]$ is undefined. (It turns out that such undefined cases never occur in this paper, so that we can assume that $F[x_i = 1]$ is always defined.) $F[x_i = 0]$ is defined similarly. Moreover $F[x_i = x_j]$ is a predicate defined by the following: (1) Replace all x_i by x_j and all $\overline{x_i}$ by $\overline{x_j}$. (2) Delete any clause that includes both x_j and $\overline{x_j}$. (3) If a clause includes two x_j's (or $\overline{x_j}$'s) then delete one of them. $F[x_i = \overline{x_j}]$ is similar. In these substitutions, x_j may or may not appear in F.

Lemma 3. Suppose that a predicate F can be generated in polynomial time. Then (1) $F[x_k = 1]$, (2) $F[x_k = 0]$, (3) $F[x_k = x_j]$ and (4) $F[x_k = \overline{x_j}]$ can also be generated in polynomial time. (More formally: If F can be generated in m steps, then $F[x_k = 1]$ and so on can be generated in $m +$ [poly in the length of $F[x_k = 1]$] steps. Similarly for the following lemmas and theorems.)

Proof. We shall only give a proof for (1). Without loss of generality, we can assume that $x_k \neq x_1$. Suppose that F is generated by the sequence σ_0 of $F_0 = x_1\overline{x_1}$, $F_1, \cdots, F_i, F_{i+1}, \cdots, F_q = F$. Below we shall get a new generation sequence σ_1 of $H_0 = x_1\overline{x_1}$, $H_1, \cdots, H_i, H_{i+1}, \cdots, H_q = F[x_k = 1]$.

For a clause A, define a mapping μ as $\mu(A) = A$ if A does not include x_k or $\overline{x_k}$, $\mu(A + x_k) = \phi$ and $\mu(A + \overline{x_k}) = A$. Also we introduce the following assertion Q_i: H_i includes clauses $\mu(A)$ such that A is included in F_i and $\mu(A) \neq \phi$. When $i = 0$, Q_0 is certainly true since both F_0 and H_0 are $x_1\overline{x_1}$.

Now suppose that Q_i is true and some rule is applied to F_i to get F_{i+1}. There are several cases:

Case 1. The rule is $A \rightarrow (A + x_j)(A + \overline{x_j})$ where A includes neither x_k nor $\overline{x_k}$ and $x_j \neq x_k$. Then by the assumption (Q_i is true), A also appears in H_i and exactly the same rule can be applied to get H_{i+1}. Now Q_{i+1} is obviously true.

Case 2. The rule is $A \rightarrow (A + x_k)(A + \overline{x_k})$. Again A is also included in H_i (A must not include x_k or $\overline{x_k}$). In this case we set $H_{i+1} = H_i$, namely, the application of the rule in σ_0 is completely skipped in σ_1. Note that the assertion Q_{i+1} is again true.

Case 3. The rule is the same as Case 1 but A includes $\overline{x_k}$. Let A' be the clause obtained by deleting $\overline{x_k}$ in A. By the assumption, $\mu(A) = A'$ exists in H_i. So, we can apply the same rule to A' (i.e., $A' \rightarrow (A' + x_j)(A' + \overline{x_j})$) to get H_{i+1}.

Case 4. The same as Case 1 but A includes x_k. $\mu(A)$ does not exist in H_i. Set $H_{i+1} = H_i$.

Case 5. Literal $\overline{x_k}$ is deleted from $(A + \overline{x_k})$. $\mu(A + \overline{x_k}) = A$ exists in H_i. Set $H_{i+1} = H_i$.

Case 6. Literal x_k is deleted from $(A + x_k)$. Recall that $\mu(A + x_k) = \phi$ and thus neither $(A + x_k)$ nor A exists in H_i. Add clause A to H_i to get H_{i+1}.

Case 7. Clause A, such that neither x_k nor $\overline{x_k}$ exists in it, is deleted since it is covered by some other B. Note that B must not include x_k or $\overline{x_k}$. So both $\mu(A) = A$ and $\mu(B) = B$ exist in H_i. Apply the same rule to H_i (delete A) to get H_{i+1}.

Case 8. Clause $(A + \overline{x_k})$ is deleted since it is covered by B. Since B must not include x_k, $\mu(B) \neq \phi$. One can then see that $\mu(A + \overline{x_k})$ is covered by $\mu(B)$ whether or not B includes $\overline{x_k}$. Delete $\mu(A + \overline{x_k})$ to get H_{i+1}.

Case 9. $(A + x_k)$ is deleted. Set $H_{i+1} = H_i$.

There remain some other cases, but they are easy and are omitted. Although the sequence σ_1 contains unchanged portions as mentioned above, one can just remove those portions to get a proper sequence of generation. Also one can assure that Q_{i+1} is true in any case, which claims the lemma. □

Lemma 4. Suppose that two predicates $AA_1 \cdots A_k$ and $B_1B_2 \cdots B_l$ can be both generated in polynomial time. Then so can be done $(A + B_1)(A + B_2) \cdots (A + B_l)A_1 \cdots A_k$, where $(A + B_i)$ is deleted if it includes both positive and negative forms of the same variable. Repeated literals are also removed.

Proof. We first generate $AA_1 \cdots A_k$ from $x_1\overline{x_1}$. Then change it to $(A + x_1)(A + \overline{x_1})A_1 \cdots A_k$ by splitting. Now consider the other generation from $x_1\overline{x_1}$ to $B_1B_2 \cdots B_l$. Note that if we add A to all the clauses appearing in this generation sequence, then it can be regarded as a generation from $(A + x_1)(A + \overline{x_1})$ to $(A + B_1)(A + B_2) \cdots (A + B_l)$. Now one can continue the first generation from $(A + x_1)(A + \overline{x_1})A_1 \cdots A_k$ to get $(A + B_1)(A + B_2) \cdots (A + B_l)A_1 \cdots A_k$. Apply the same technique as

Lemma 3 to delete the improper clauses and repeated literals. □

Lemma 5. Suppose that two predicates $AA_1 \cdots A_k$ and $BA_1 \cdots A_k$ can be both generated in polynomial time. Then so can be done $(A + B)A_1 \cdots A_k$ under the same condition as the previous lemma.

Proof. Apply Lemma 4. Then $(A + B)(A + A_1) \cdots (A + A_k)A_1 \cdots A_k$ can be generated. All $(A + A_i)$ can be deleted since it is covered by A_i. □

4.3 Efficiency of the Simulation

We shall now prove that there is an efficient simulation of TLHC by USG.

Lemma 6. Suppose that a graph G consists of some number of components G_i, each of which is non–3–colorable, and also that we get G', consisting of components G'_i, by applying one of the four rules of TLHC. Then if each $P_r(G_i)$, for all i, can be generated in polynomial time then so can be done $P_r(G'_i)$ for all i.

Proof. There are four cases.

Case 1. The rule is the addition of K_4. Then the lemma is trivial since G' consists of exactly the same components as G plus the new K_4. $P_r(K_4)$ can be generated in constant time.

Case 2. The rule is addition of vertices and/or edges. Recall that new vertices must be connected to some component. Suppose, for example, components G_1 and G_2 of G become connected (the resulting graph to be G'_1) by means of the newly introduced vertices and edges. Then $P_r(G'_1)$ can be obtained by simply adding polynomially many clauses to $P_r(G_1)P_r(G_2)$. (If $P_r(G_1)$ and $P_r(G_2)$ share the same variable then regenerate $P_r(G_2)$ in advance using completely different variables.)

Case 3. Components G_1 and G_2 are joined into G'_1. See Fig. 1 again. Let

$$P_r(G_1) = A_1 \cdots A_n(\overline{R_a} + \overline{R_b})(\overline{B_a} + \overline{B_b})(\overline{G_a} + \overline{G_b}),$$

$$P_r(G_2) = B_1 \cdots B_m(\overline{R_c} + \overline{R_d})(\overline{B_c} + \overline{B_d})(\overline{G_c} + \overline{G_d}).$$

(i) We first rename (by substitution) variables R_c, B_c and G_c of $P_r(G_2)$ into R_a, B_a and G_a, respectively, and get

$$B_1 \cdots B_m(\overline{R_a} + \overline{R_d})(\overline{B_a} + \overline{B_d})(\overline{G_a} + \overline{G_d}),$$

which can be generated in polynomial time by Lemma 3.

(ii) Apply Lemma 4 to $P_r(G_1)$ and the predicate in (i) above to get

$$A_1 \cdots A_n(\overline{B_a} + \overline{B_b})(\overline{G_a} + \overline{G_b})(\overline{R_a} + \overline{R_b} + B_1) \cdots (\overline{R_a} + \overline{R_b} + B_m)$$
$$\cdot(\overline{R_a} + \overline{R_b} + \overline{R_d})(\overline{R_a} + \overline{R_b} + \overline{B_a} + \overline{B_d})(\overline{R_a} + \overline{R_b} + \overline{G_a} + \overline{G_d}).$$

(iii) Change $(\overline{R_a} + \overline{R_b} + \overline{B_a} + \overline{B_d})$ into $(\overline{R_a} + \overline{B_a})$ by the literal deletion rule and split it into $(\overline{R_a} + \overline{B_a} + \overline{G_a})(\overline{R_a} + \overline{B_a} + G_a)$. Then both clauses can be deleted since they are the same as some of the clauses introduced for vertex a. $(\overline{R_a} + \overline{R_b} + \overline{G_a} + \overline{G_d})$ is also deleted similarly. Furthermore delete literals $\overline{R_a}$ and $\overline{R_b}$ from $(\overline{R_a} + \overline{R_b} + B_1) \cdots (\overline{R_a} + \overline{R_b} + B_m)$. Then add clauses associated with the new edge between vertices b and d. Now we get

$$A_1 \cdots A_n B_1 \cdots B_m(\overline{R_a} + \overline{R_b} + \overline{R_d})(\overline{B_a} + \overline{B_b})(\overline{G_a} + \overline{G_b})(\overline{R_b} + \overline{R_d})(\overline{B_b} + \overline{B_d})(\overline{G_b} + \overline{G_d}),$$

where we can delete $(\overline{R_a} + \overline{R_b} + \overline{R_d})$ because of $(\overline{R_b} + \overline{R_d})$.

(iv) Note that steps (ii) and (iii) can be regarded as playing the role of deleting $(\overline{R_a} + \overline{R_b})$. Hence we can repeat similar steps to delete $(\overline{B_a} + \overline{B_b})$ and then to delete $(\overline{G_a} + \overline{G_b})$. Now we get

$$A_1 \cdots A_n B_1 \cdots B_m (\overline{R_b} + \overline{R_d})(\overline{B_b} + \overline{B_d})(\overline{G_b} + \overline{G_d}),$$

which is what we wanted for $P_r(G'_1)$.

Case 4. Contraction rule. $P_r(G'_1)$ can be obtained by a simple renaming of variables (omitted).
□

Lemma 7. If $P_r(G_r(F))$ can be generated in polynomial time then so can be done F.

Proof. Fig. 3 shows a portion of $G_r(F)$ where the subgraph of the lower part is associated with a clause $(x_k + \overline{x_j} + x_i)$. (The following discussion does not differ much if the clause contains four or more literals.) We prepare variables R_a, B_a, G_a for v_a, R_0, B_0, G_0 for v_0, R_i, B_i, G_i for x_i, and R'_i, B'_i, G'_i for $\overline{x_i}$. Similarly for v_b, v_c and $v_1, \cdots, v_5$.

Let $H = P_r(G_r(F))$. H can be obtained by preparing the clauses as described in Sec. 4.1. As described below, we shall modify (simplify) this H mainly using the substitution operation mentioned in Sec. 4.2, in such a way that if H can be generated in polynomial time then so can be done the simplified predicate.

(i) We first fix the value of the variables associated with v_a, v_b, v_c and v_0 as follows: $G_a = 1$, $R_a = B_a = 0$, $R_b = 1$, $B_b = G_b = 0$, $B_c = 1$, $R_c = G_c = 0$, and $G_0 = 1$, $R_0 = B_0 = 0$. This substitution means that we fixed the color of v_a, v_b, v_c and v_0 to green, red, blue and green, respectively. H is simplified to H_1 by this substitution. By Lemma 3, if H can be generated in polynomial time then so can be done H_1.

(ii) Further simplify H_1 by substitution $G_1 = G_2 = 0$, $B_i = B'_i = B_j = B'_j = B_k = B'_k = 0$. Moreover carry out the following substitution: $B_2 = \overline{R_2}$, $R_1 = \overline{R_2}$, and $B_1 = R_2$. For vertices x_k and $\overline{x_k}$, $R_k = G'_k = \overline{G_k}$ and $R'_k = G_k$, and similarly for x_i, $\overline{x_i}$, x_j and $\overline{x_j}$. Then the resulting predicate, say, H_2, becomes much simpler than the original one; we have already no clauses for such vertices as v_a, v_b, v_c, v_0, v_1, v_2, x_i, $\overline{x_i}$, x_j, $\overline{x_j}$, x_k and $\overline{x_k}$. All of the original clauses for v_3, v_4 and v_5 remain as they were. As for the clauses associated with edges, there remain

$$(G_k + \overline{R_2}) \text{ for } (x_k, v_2),$$

$$(\overline{G_j} + \overline{R_5}) \text{ for } (\overline{x_j}, v_5),$$

$$(G_i + \overline{R_4}) \text{ for } (x_i, v_4),$$

$$(R_2 + \overline{R_3})(\overline{R_2} + \overline{B_3}) \text{ for } (v_3, v_1),$$

and all of the original clauses for (v_3, v_4), (v_4, v_5) and (v_5, v_3).

(iii) We execute a very similar procedure for other subgraphs associated with other clauses of F. Let $H_2 = H_{20}H_{21}$ where H_{21} is the remaining clauses described in (ii) above and H_{20} is all the other clauses for the other subgraphs.

(vi) Now carry out further substitution: $R_2 = 1$, $R_3 = 1$, $B_4 = 1$, $G_5 = 1$ and all other variables except G_i, G_j and G_k are set to 0. Then one can see that H_2 becomes $H_{20}(G_k)$ ((G_k) is a clause of a single literal). It should be noted that variables $R_1 \cdots R_5$, $B_1 \cdots B_5$ and $G_1 \cdots G_5$ appearing in H_{21} never appear in H_{20}. Hence the above substitution does not change H_{20} at all.

(v) If we make a different substitution for $H_2 = H_{20}H_{21}$: $B_3 = 1$, $G_4 = 1$, $R_5 = 1$ and all other variables except G_i, G_j and G_k are set to 0, then H_2 becomes $H_{20}(\overline{G_j})$. Therefore, by Lemma 4, $H_{20}(G_k + \overline{G_j})$ can be generated in polynomial time.

(vi) If we execute one more different substitution for $H_2 = H_{20}H_{21}$ (details are omitted), then H_2 becomes $H_{20}(G_i)$ and again by Lemma 4, $H_{20}(G_k + \overline{G_j} + G_i)$ can be generated in polynomial time.

(vii) Now we simplify $H_{20}(G_k+\overline{G_j}+G_i)$ further by applying the same procedure to other subgraphs. Finally one can get the predicate F' in polynomial time where F' is such a predicate that each x_i ($\overline{x_i}$) of the original predicate F is changed to G_i ($\overline{G_i}$). To modify F' to F is easy. □

Theorem 1. If TLHC is polynomially-bounded then so is USG.

Proof. Outline of the simulation was given in Sec. 4.1 and we can get rid of the unsettled matters there by Lemmas 6 and 7. □

Corollary 1. TLHC is not polynomially-bounded.

Proof. It is known [Hak85] that there are predicates which cannot be generated in poly-time by USG. □

References

[Ajt88] M. Ajtai. The complexity of the pigeonhole principle. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 346–355, 1988.

[BIK+92] P. Beame, R. Impagliazzo, J. Krajicek, T. Pitassi, P. Pudlák, and W. Woods. Exponential lower bounds for the pigeonhole principle. In *Proc. 24th ACM Symposium on Theory of Computing*, pages 200–220, 1992.

[Coo75] S. Cook. Feasibly constructive proofs and the propositional calculus. In *Proc. 16th IEEE Symp. on Foundations of Computer Science*, pages 83–97, 1975.

[CR79] S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *J. Symbolic Logic*, 44, 1979.

[CR92] V. Chvátal and B. Reed. Mick gets some (the odds are on his side). In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 620–627, 1992.

[GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1:237–267, 1976.

[Haj61] G. Hajós. Über eine Konstruktion nicht n–färbbarer Graphen. *Wiss. Zeitschr. Martin Luther Univ. Halle–Wittenberg*, A 10, 1961.

[Hak85] A. Haken. The intractability of resolution. *Theor. Compout. Sci.*, pages 297–308, 1985.

[IAM92] K. Iwama, H. Abeta, and E. Miyano. Random generation of satisfiable and unsatisfiable CNF predicates. In *Proc. 12th IFIP World Computer Congress*, pages 322–328, 1992.

[IM93] K. Iwama and E. Miyano. Security of test-case generation with known answers. In *Proc. AAAI Spring Symposium Series*, 1993.

[Kuc91] L. Kucéra. A generalized encription scheme based on random graphs. In *Proc. 17th Ann. Workshop on Graph-Theoretic Concepts in Computer Science (Lecture Notes in Computer Science 570)*, pages 180–186, 1991.

[MW] A. Mansfield and D. Welsh. Some coloring problems and their complexity. *Annals of Discrete Math.*, 13:159–170.

[PU92] T. Pitassi and A. Urquhart. The complexity of Hajós calculus. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 187–196, 1992.

[Sanar] L. Sanchis. Generating hard and diverse test sets for hp-hard graph problems. *Discrete Applied Math.*, to appear.

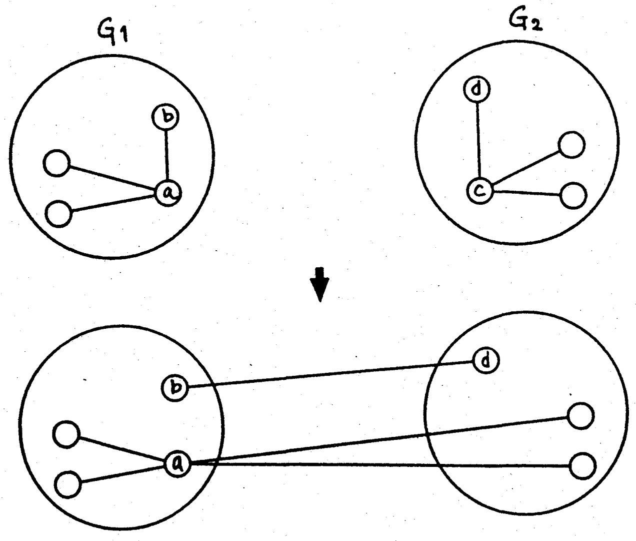

Fig. 1

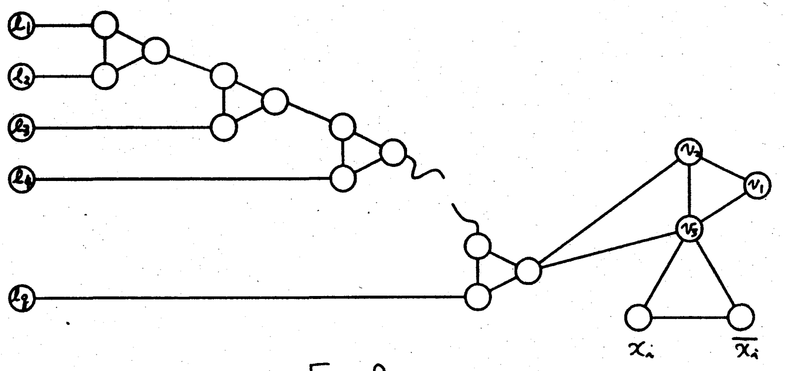

Fig. 2

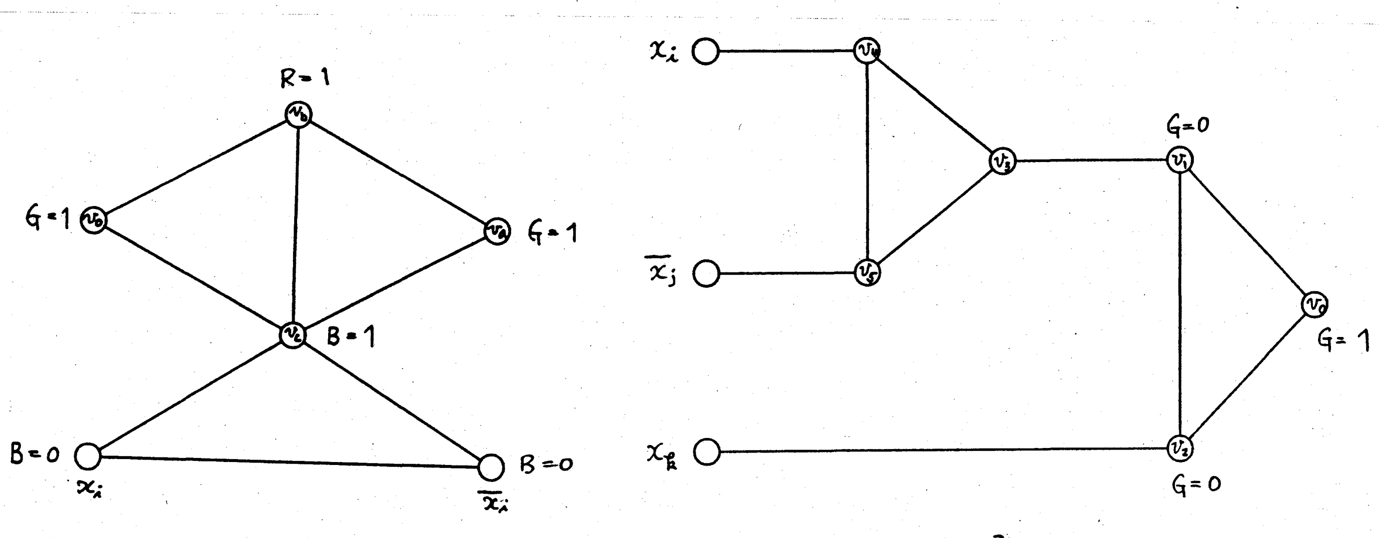

Fig. 3