ON THE COMPLEXITY OF BRANCHING PROGRAMS AND DECISION TREES FOR CLIQUE FUNCTIONS

Ingo Wegener*
FB 20-Informatik, Johann Wolfgang Goethe-Universität,
6000 Frankfurt a.M., Fed.Rep. of Germany

Abstract

Because of the slow progress in proving lower bounds on the circuit complexity of Boolean functions one is interested in restricted models of Boolean circuits like depth restricted circuits, decision trees, branching programs, width-k branching programs and k-times-only branching programs. We prove here exponential lower bounds on the decision tree complexity of clique functions. For one-time-only branching programs we prove for k-clique functions large polynomial lower bounds if k is fixed and exponential lower bounds for k increasing with n. Finally we introduce the hierarchy of the classes $BP_k(P)$ of all sequences of Boolean functions which may be computed by k-times-only branching programs of polynomial size. We show constructively that $BP_1(P)$ is a proper subset of $BP_2(P)$.

1. INTRODUCTION

Until now one knows only a few poor methods for the proof of lower bounds on the circuit complexity of explicitly defined Boolean functions. Therefore one has considered since a long time restricted models like formulae, monotone circuits, branching programs, contact schemes (Nechiporuk [9]) and also restricted models of branching programs like width restricted branching programs (Ajtai et al. [1], Barrington [2], Borodin/Dolev/Fich/Paul [3], Chandra/Furst/Lipton [4], Pudlák [10] and Yao [15]) and (depth restricted) k-times-only branching programs (Ajtai et al. [1], Dunne [5], Kriegel/Waack [7], Masek [8], Pudlák/Zák [11] and Wegener [12], [13], [14]). We assume that the reader is familiar with Boolean circuits and formulae. A decision tree is a directed, labelled binary tree where the inner nodes are labelled by Boolean variables and the leaves by Boolean constants. One starts at the root and after reaching an inner node one tests that variable which

* Supported in part by DFG-grants No. We 1066/1-2 and Me 872/1-1

is the label of the node. If its value is 0 (or 1) one goes to the left (or right) successor. The label of the leaf one reaches is the value of the function computed by the decision tree. DT(f), the decision tree complexity of f, is the minimal number of inner nodes of a decision tree for f.A branching program is a directed acylic graph with one source where the computation starts, inner nodes of outdegree 2 and sinks of outdegree 0. The labelling and the mode of computation is similar to that of decision trees, the proper complexity measure is BP(f). Width-k branching programs are levelled and have not more than k nodes on each level. On the other hand k-times-only branching programs (BP_ks) have a depth restriction. One is allowed to test each variable on each path of computation only for k times, the complexity measure is $BP_k(f)$. Obviously all optimal decision trees are BP_1s. A computation of a BP_k needs at most kn points of time while the computation time for width-k branching programs is of the same size as the number of nodes of the program. The problem is to decide which functions may be computed efficiently even by restricted branching programs and for which functions one can prove large lower bounds in the restricted models. For width restricted branching programs this problem has a surprising solution (Barrington [2]). Sequences of Boolean functions can be computed by branching programs of polynomial size and constant width iff they can be computed by circuits of polynomial size and logarithmic depth.

We are far from similar results for depth restricted branching programs. For the motivation of BP_ks we refer to Wegener [12]. Masek [8], Pudlák/Zák [11] and Ajtai et al. [1] proved tight relations between the size of BPs (BP_ks) and the space complexity of Turing machines (so-called eraser Turing machines).

The purpose of this paper is to present methods for the proof of lower bounds on the decision tree and BP_1 complexity of Boolean functions. We apply these methods to clique functions. The clique function f_k^n where $3 \leq k \leq n-1$ is defined on $N = \binom{n}{2}$ variables corresponding to the possible edges of an n-vertex graph. f_k^n computes 1 iff the graph specified by the variables contains a k-clique.

In Chapter 2 we consider decision trees and present some general lower bound techniques.

In Chapter 3 we present our main method, a lower bound method for BP_1s. We show which computation paths cannot lead to the same computation node in a BP_1, since a BP_1 cannot separate the situations by repea-

ting an old test. This method leads to strong lower bounds for clique functions. The largest lower bounds we obtain are of size $\exp\{\theta(N^{1/2})\}$ for the number of variables N. This method has first been presented in a preliminary version of this paper [13]. Dunne [5] applied this method to other functions. Recently Ajtai et al. [1] and Kriegel/Waack [7] used this approach to prove even lower bounds of size $\exp\{\theta(N)\}$.

In Chapter 4 we consider the hierarchy problem for BP_ks. Let $BP_k(P)$ be the class of sequences of Boolean functions computable by BP_ks of polynomial size. We conjecture that these classes build a proper hierarchy, i.e. $BP_{k-1}(P)$ is a proper subclass of $BP_k(P)$. We present candidates which may separate the classes and prove that $BP_1(P)$ is a proper subclass of $BP_2(P)$.

2. ON THE DECISION TREE COMPLEXITY OF CLIQUE FUNCTIONS

Similar to Boolean formulae we count here the number of leaves instead of the number of inner nodes of decision trees. This number is only by 1 larger than the decision tree complexity.

Definition 1: Let $DT^*(f) := DT(f)+1$ $(DT_0(f), DT_1(f))$ denote the minimal number of leaves (0-leaves, 1-leaves) in a decision tree for f. Let M(f) be the minimal number of monoms in a disjunctive form for f.

M(f) is also the complexity of f in Σ_2-circuits, that means circuits of depth 2 where the last level consists of an v-gate. The following result shows that decision trees are less efficient than depth-2 circuits.

Theorem 1: i) $DT^*(f) \geq DT_0(f) + DT_1(f)$.

ii) $DT_1(f) \geq M(f)$.

iii) $DT_0(f) \geq M(\bar{f})$.

Proof: i) is obvious.

ii) Consider a decision tree for f with the minimal number of 1-leaves. Any 1-leaf L corresponds to a unique path from the root to L. Let m(L) be the monom consisting of all variables and negated variables which must be 1 if we follow this path. Then f is the disjunction of all m(L) and we obtain a disjunctive form for f with $DT_1(f)$ monoms.

iii) follows in a similar way. Q.E.D.

The reader is asked to convince himself that the number of 1-leaves of

a decision tree for f may be smaller than the number of prime implicants of f. But the following result, whose easy proof is left to the reader, shows that this cannot happen for monotone functions.

Proposition 1: If f is monotone, M(f) is equal to the number of prime implicants of f and $M(\bar{f})$ is equal to the number of prime clauses of f.

In [13] the number of prime clauses of the clique function is estimated. The number of prime implicants obviously equals $\binom{n}{k}$. These estimations combined with Theorem 1 and Proposition 1 lead to the following theorem.

Theorem 2: $DT(f_k^n) \geq \binom{n}{k} + (k-1)^{n-k+1}$, $DT(f_3^n) \geq 5^{n-5}$.

In order to obtain larger lower bounds on the decision tree complexity of clique functions we use another general approach. We have already seen that the monom corresponding to a path from the root to a 1-leaf (0-leaf) in a decision tree for f is an implicant (a clause) of f. We label the edges of a decision tree such that edges to left (right) successors get label 0 (1). Then we may identify each node v with the 0-1-sequence $(i_1,\ldots,i_m)$ consisting of the labels of the edges lying on the path from the root to v. By our considerations above we get the following result.

Theorem 3: Let f be a Boolean function and let ℓ_1 (ℓ_0) be the length of the shortest prime implicant (prime clause) of f. Any decision tree for f contains all nodes $(i_1,\ldots,i_m)$ where the number of ones is less than ℓ_1 and the number of zeros is less than ℓ_0. In particular

$$DT(f) \geq \sum_{0 \leq m \leq \ell_0+\ell_1-2} \; \sum_{m-\ell_0+1 \leq j \leq \ell_1-1} \binom{m}{j}.$$

We apply this result to clique functions. Obviously all prime implicants have length $\binom{k}{2}$ corresponding to the edges of a k-clique. Thus $\ell_1 = \binom{k}{2}$. The prime clauses have different lengths. A shortest prime clause corresponds to a minimal set of edges destroying any k-clique. The remaining edges build a maximal edge set of a graph without any k-clique. By the well-known Theorem of Turán such a graph consists of k-1 groups of vertices of nearly the same size (differing at most by 1) such that a vertex is connected to all vertices of other groups. By an easy convexity argument the number of edges is overestimated if we handle $\frac{n}{k-1}$ as an integer. We have to estimate the number of missing edges. There are $\frac{n}{k-1}-1$ missing edges incident to each node.

Thus $\ell_0 := \lceil n(\frac{n}{k-1} - 1)/2 \rceil$ is appropriate.

The lower bound of Theorem 3 gives only polynomial lower bounds for constant k. For k increasing with n the lower bound improves the bounds of Theorem 2. If for example $k(n) := \lceil n^{2/3} \rceil$, ℓ_0 as well as ℓ_1 are of size $\frac{1}{2} n^{4/3} - o(n^{4/3})$.

Corollary 1: $DT(f^n_{k(n)}) \geq 2^{n^{4/3}/2 - o(n^{4/3})}$ for $k(n) := \lceil n^{2/3} \rceil$.

In [13] also upper bounds on the decision tree complexity of clique functions are presented. The estimation of the size of a decision tree leads already for very simple algorithms to hard combinatorial problems, e.g. one may apply results on the number of clique-free graphs (Erdös/ Kleitman/Rothschild [6]).

3. ON THE BP_1-COMPLEXITY OF CLIQUE FUNCTIONS

As we have seen it is rather easy to prove large lower bounds on the decision tree complexity of Boolean functions. Decision trees are easy to describe but they are often puffed up. They may contain the same large subtree at different places. In order to decrease the size of the computation scheme it would be sufficient to describe the subtree only once and to point from many nodes to the root of the subtree. This is allowed in branching programs. Furthermore it is obvious that optimal decision trees have the property that on each path of computation each variable is tested only once. This ensures that the depth is bounded by the number of variables. In branching programs a node may have many predecessors and the subfunctions which have to be computed if one reaches this node from different predecessors need not to be equal. One may separate these situations again by repeating tests one has done before. The advantage of this procedure is the reduction of the size of the branching program at the cost of increasing the depth. For one-time-only branching programs (BP_1s) the depth is also bounded by the number of variables. The size of BP_1s for f may be polynomial even if DT(f) is exponential (see [12]).

We use the following approach for the proof of lower bounds on the BP_1-complexity of clique functions. In the discussion above we have seen that in branching programs it is sometimes necessary to repeat tests in order to separate situations one has merged before. This is not allowed

in BP_1s. There we can merge situations only if we are not forced to separate them by asking an old question again. A merging of two nodes v and w is possible if and only if the knowledge that one has reached v or w and the knowledge of all variables not tested on any path to v or w is sufficient to determine the value of the function. These considerations show that mergings are not possible near the source of the branching program. Therefore BP_1s behave near the source like decision trees. We prove that mergings are possible only after a certain number of tests.

Theorem 4: Let v and w be different nodes in a BP_1 for the clique function f_k^n. If on a path to v and also on a path to w we have tested at most $k_1 := \binom{k}{2} - 1$ variables positively and at most $k_0 := n - 2\binom{k}{2} - k + 2$ variables negatively neither v nor w may be a sink of the program. Furthermore it is not possible to merge v and w without changing the BP_1 on paths to v or w.

Proof: Neither v or w is a sink since on some path to v or w we have not tested positively all variables of a prime implicant ($k_1 < \binom{k}{2}$) and we have not tested negatively all variables of a prime clause ($k_0 < \ell_0$, the length of the shortest prime clause , see Chapter 2). Here we need the following fact which holds for BP_1s but not for BP_ks and $k \geq 2$. For each path in a BP_1 there exists some input for which we have to follow this path.

If v is a successor of w (or vice versa) we cannot merge these nodes without creating a cycle which is forbidden for branching programs. We consider in the following those paths from the source to v and w which fulfil the assumptions of the Theorem. Since v and w are not on the same path there exists an edge e which has been tested positively (negatively) if we reach v (w). The clique function is symmetric with respect to all edges, thus we may assume w.l.o.g. that e is the edge (1,2).

Each path in a BP_1 corresponds to a partial graph where some edges are existing (tested positively), some edges are forbidden (tested negatively) and the other edges are variable (not tested yet). Let G_1 (G_2) be the partial graph corresponding to the chosen path from the source to v(w). The edge (1,2) exists in G_1 and is forbidden in G_2.

We prove the claim in the following way. We show that we can fix all edges not tested on the paths to v or w in such a way that either one

of the (perhaps partial) graphs G_1^* or G_2^* arising from G_1 and G_2 by this supplement may be completed to a graph without a k-clique and to a graph with a k-clique or G_1^* contains a k-clique and G_2^* does not. In any case we cannot compute correctly f_k^n if we merge v and w and do not change some of the paths leading to v or w.

Let A be the set of vertices $v \notin \{1,2\}$ such that v is on some existing edge of G_2. Obviously $|A| \leq 2\binom{k}{2}-2$. We construct B in the following way starting with the empty set. For each forbidden edge of G_1 we add one incident vertex $w \notin \{1,2\}$ to B. This is possible since the edge (1,2) exists in G_1. Again obviously $|B| \leq n-2\binom{k}{2}-k+2$. Thus $C:=\{1,\ldots,n\}-A-B$ contains at least k vertices, in particular 1 and 2. Let D be a k-element subset of C containing 1 and 2. By our construction above there does not exist any forbidden edge on D in G_1 or any existing edge in G_2 incident with some vertex in D.

We assume that v and w are merged to the node z. Let E* be the set of edges not tested on any path to v or w, these are the only edges which may be tested on paths starting in z. For $(i,j) \in E^*$ we set $x_{ij}=1$ if $i,j \in D$ and $x_{ij}=0$ otherwise. Let G_1^* and G_2^* be the (perhaps partial) graphs arising from G_1 and G_2 by deciding the existence of the edges of E* in the described way. G_1^* has no forbidden edge on D. Either G_1^* contains a k-clique or the k-clique D is still possible and no k-clique is already completed. The computation has to stop since we are not allowed to test any variable without violating the conditions of a BP_1. But in the second case we do not know the value of the function. The value may be 1 since D is possible and it may be 0 since we have not completed any k-clique. This would be a contradiction and we may assume that G_1^* contains a k-clique.

Let us now investigate G_2^*. By the same arguments as above G_2^* has to contain a k-clique or the existence of a k-clique has to be impossible. We reach the same sink for the inputs corresponding to G_1^* and G_2^*. Thus G_2^* has to contain a k-clique. Vertices with positive degree in G_2^* belong to $D \cup A$. Since (1,2) is forbidden in G_2^* this graph does not contain the k-clique D. Since G_2 does not contain any k-clique and since we test afterwards only edges on D positively G_2^* does not contain any k-clique on A. Finally G_2^* does not contain any edge joining A and D. Thus G_2^* is free of any k-clique and the assumption that we may merge v and w leads to a contradiction.

Q.E.D.

The reader may convince himself that the parameters of Theorem 4 cannot be improved significantly. If k=3, we have $k_1=2$ and $k_0= n-7$. There is a BP_1 for f_3^n where two paths on which 2 edges have been tested positively and 4n-13 edges have been tested negatively are merged.

Theorem 4 shows that all nodes corresponding to 0-1-sequences of at most k_0 zeros and k_1 ones exist and are different. Hence we get the lower bound $BP_1(f_k^n) \geq \sum_{0\leq m\leq k_0+k_1} \sum_{m-k_0\leq j\leq k_1} \binom{m}{j}$.

At first we evaluate this bound for constant k. For $m \geq 2k_1$ the second sum is dominated by $\binom{m}{k_1}$. Thus

$$BP_1(f_k^n) \geq \sum_{0\leq m\leq k_0+k_1} \binom{m}{k_1} = \Omega\Big(\sum_{0\leq m\leq k_0+k_1} m^{k_1}\Big)$$

$$=\Omega((k_0+k_1)^{k_1+1})=\Omega(n^{k(k-1)/2}).$$

For constant k we get only polynomial lower bounds but the polynomial grows much faster than the size of the disjunctive normal form which is $\theta(n^k)$.

The bound is large if k_0 and k_1 have nearly the same size.
Let $k(n):=\lceil (2n/3)^{1/2} \rceil$. Then
$k_1=\binom{k(n)}{2}=n/3-o(n)$ and
$k_0=n-2\binom{k(n)}{2}-k+2=n/3-o(n)$.
A BP_1 for $f_{k(n)}^n$ is up to level $\min(k_0,k_1)=n/3-o(n)$ a complete binary tree and we can conclude that
$BP_1(f_{k(n)}^n)=\Omega(2^{n/3-o(n)})$.

Our bound is worse for larger k(n). For $k(n) > n^{1/2}$ the parameter k_0 is negative and Theorem 4 seems to be useless. But then we apply the obvious fact that the BP_1-complexity of some function f is not smaller than the BP_1-complexity of each subfunction f' of f.

Proposition 2: f_{k-m}^{n-m} is a subfunction of f_k^n.

Proof: Replace all variables x_{ij} where $1\leq i\leq n$ and $n-m+1\leq j\leq n$ by ones. Q.E.D.

Hence we can conclude for example that
$BP_1(f_{n/2}^n) \geq BP_1(f_{k(n)}^{n/2+k(n)})$.

Let $k(n):=\lceil (n/3)^{1/2} \rceil$. Then we get $k_1=n/6-o(n)$ and $k_0=n/6-o(n)$, thus

$BP_1(f^n_{n/2})=\Omega(2^{n/6-o(n)})$. The following theorem summarizes some of the applications of Theorem 4.

Theorem 5: i) If k is fixed,

$BP_1(f^n_k)=\Omega(n^{k(k-1)/2})$.

ii) If $k(n):=\lceil(2n/3)^{1/2}\rceil$, $BP_1(f^n_{k(n)})=\Omega(2^{n/3-o(n)})$.

iii) $BP_1(f^n_{n/2})=\Omega(2^{n/6-o(n)})$.

4. THE $BP_k(P)$-HIERARCHY

We know now that BP_1s may be much more efficient than decision trees but much less efficient than branching programs or circuits. Therefore one has to ask how much one has to increase the resources in order to get efficient branching programs. We ask whether one can compute more functions efficiently by BP_{k+1}s than by BP_ks.

Definition 2: $BP_k(P)$ is the set of all sequences of Boolean functions $f_n:\{0,1\}^n\to\{0,1\}$ such that $BP_k(f_n) \leq p(n)$ for some polynomial p. BP(P) is defined analogously.

Obviously $BP_1(P)\subseteq\ldots\subseteq BP_k(P)\subseteq BP_{k+1}(P)\subseteq\ldots\subseteq BP(P)$.
If all inclusions were proper we would obtain an interesting hierarchy of easy problems, namely those problems having branching programs of polynomial size. For a sequence f_n the minimum k such that $f_n \in BP_k(P)$ would be an interesting new measure for the complexity of functions in BP(P). On the other hand BP(P) is not too small. BP(P) contains all f_n of polynomial formula size (see Wegener [12]) and all f_n of logarithmic space complexity with respect to Turing machines (Pudlák/Zák [11]).

We can only prove that the first inclusion of the hierarchy is proper. Let g_n be the exactly-half clique function computing 1 iff the specified graph on n vertices consists of an n/2-clique and n/2 isolated vertices. Pudlák/Zák [11] proved that g_n is not in $BP_1(P)$.

Theorem 6: $BP_1(P) \subsetneq BP_2(P)$.

Proof: By the result of Pudlák/Zák [11] it is sufficient to construct for g_n a BP_2 of polynomial size.

$g_n(x) = 1$ iff in $G_n(x)$, the graph specified by x, the degree of each vertex is 0 or n/2-1, and there is some vertex i* of degree n/2-1 such that all vertices $i < i^*$ have degree 0 and all vertices of positive degree are connected to i*.

It is easy to design a BP_1 on m variables which has size $O(m^2)$ and m+1 sinks and where all inputs with exactly i ones reach the i-th sink ($0 \leq i \leq m$). Let T_ℓ be such a BP_1 on the variables $x_{\ell,j}$ ($\ell < j \leq n$). We obtain T^*_ℓ from T_ℓ by replacing the sinks $i \notin \{0,n/2-1\}$ by 0-sinks, the sink 0 by the source of $T^*_{\ell+1}$ if $\ell < n$ and a 0-sink if $\ell=n$, and the sink n/2-1 by the source of $E^*_{\ell,\ell+1}$ if $\ell < n$ and a 0-sink if $\ell=n$. Let $E_{\ell,j}$ for $\ell < j$ be a BP_1 of quadratic size counting the ones among x_{mj} ($\ell+1 \leq m \leq j-1$) and x_{jm} ($j+1 \leq m \leq n$). We obtain $E^*_{\ell,j}$ from $E_{\ell,j}$ in the following way. The sinks $i \notin \{0,n/2-2\}$ are replaced by 0-sinks, the sinks $i \in \{0,n/2-2\}$ are replaced by a test of $x_{\ell j}$. If i=0 and $x_{\ell j}=1$ or i=n/2-2 and $x_{\ell j}=0$ we reach 0-sinks. For the other two possibilities we reach the source of $E^*_{\ell,j+1}$ if $j < n$ and 1-sinks if j=n. The resulting branching program is shown in Fig.1.

This branching program computes g_n. We reach the E^*_ℓ-chain iff $deg(1)=\ldots=deg(\ell-1)=0$ and $deg(\ell)=n/2-1$. Then we reach a 1-sink of $E^*_{\ell,n}$ iff for $j > \ell$ either $deg(j)=0$ or $deg(j)=n/2-1$ and vertex j is connected to vertex ℓ.

The branching program is a BP_2. A path passes at most $T^*_1,\ldots,T^*_\ell$, $E^*_{\ell,\ell+1},\ldots,E^*_{\ell,n}$. The variable $x_{i,i+j}$ is tested in T^*_i (if $i \leq \ell$), in $E^*_{\ell,i}$ (if $\ell < i$) and in $E^*_{\ell,i+j}$ (if $\ell<i+j$), thus at most twice.

Each of the subprograms T^*_ℓ and $E^*_{\ell,j}$ has size $O(n^2)$. The total size is $O(n^4)$, since the number of subprograms in $O(n^2)$. The number of variables is $N=\binom{n}{2}$. Hence the BP_2 is of polynomial size $O(N^2)$.

Q.E.D.

Let g^n_k be a function on $\binom{n}{k}$ variables corresponding to the possible hyperedges of length k in a hypergraph on n vertices. $g^n_k(x) = 1$ iff the hypergraph specified by x consists of an n/2-hyperclique and n/2 isolated vertices. g^n_k is a candidate to separate $BP_{k-1}(P)$ from $BP_k(P)$.

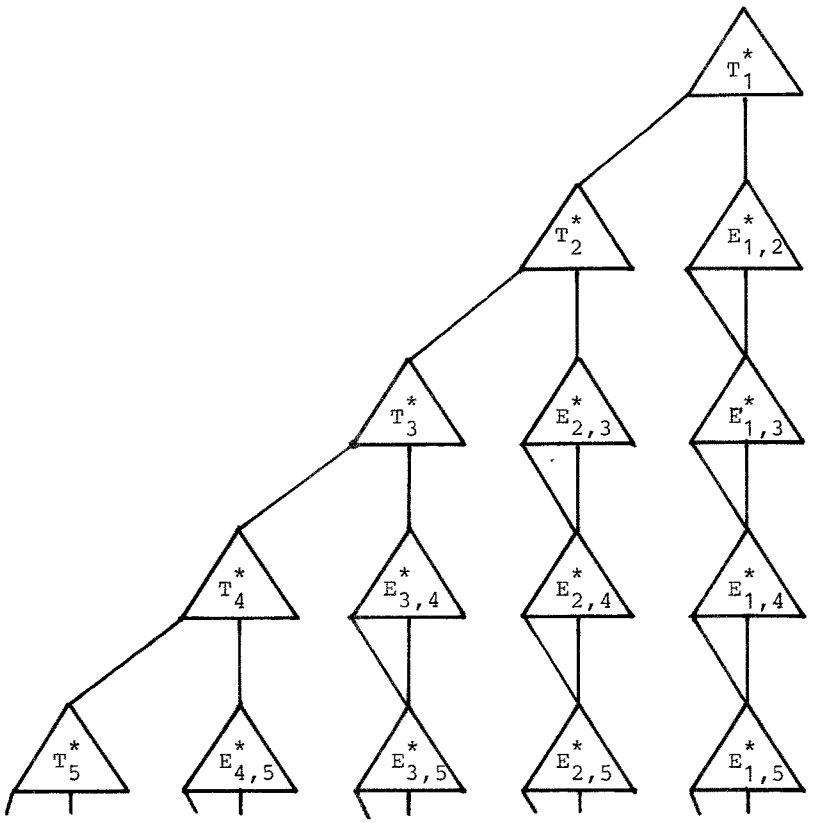

Figure 1

By the methods of Theorem 6 we can prove that g_k^n belongs to $BP_k(P)$. We conjecture that branching programs of polynomial size for g_k^n "have to look at many hyperedges from all its k vertices".

References

[1] Ajtai,M./Babai,L./Hajnal,P./Komlós,M./Pudlák,P./Rödl,V./ Szemerédi,E./Turán,G.: Two lower bounds for branching programs, 18. STOC, 30-38, 1986

[2] Barrington,D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 , 18.STOC,1-5,1986

[3] Borodin,A./Dolev,D./Fich,F.E./Paul,W.: Bounds for width two branching programs, 15.STOC, 87-93, 1983

[4] Chandra,A.K./Furst,M.L./Lipton,R.J.: Multiparty protocols, 15.STOC, 94-99, 1983

[5] Dunne,P.: Lower bounds on the complexity of 1-time only branching programs, FCT, LNCS 199, 90-99, 1985

[6] Erdös,P./Kleitman,D.J./Rothschild,B.L.: Asymptotic enumeration of K_n-free graphs. Colloq.Intern.sulle Teorie Comb.,Accad.Naz. Lincei, Rome, 19-27, 1976

[7] Kriegel,K./Waack,S.: Lower bounds on the complexity of real-time branching programs, Techn.Rep.,Akad.d.Wiss.Berlin (GDR), 1986

[8] Masek,W.: A fast algorithm for the string editing problem and decision graph complexity, M.Sc. Thesis,MIT, 1976

[9] Nechiporuk,E.I.: A Boolean function, Sov.Math.Dokl. 7, 999-1000, 1966

[10] Pudlák,P.: A lower bound on complexity of branching programs, 11.MFCS,LNCS 176, 480-489, 1984

[11] Pudlák,P./Zák,S.: Space complexity of computations, Preprint, Univ. Prague, 1983

[12] Wegener,I.: Optimal decision trees and one-time-only branching programs for symmetric Boolean functions, Information and Control 62, 129-143, 1984

[13] Wegener,I.: On the complexity of branching programs and decision trees for clique functions, Techn.Rep.,Univ.Frankfurt a.M., 1984

[14] Wegener,I.: Time-space trade-offs for branching programs, Journal of Computer and System Sciences 32, 91-96, 1986

[15] Yao, A.C.: Lower bounds by probabilistic arguments,24.FOCS, 420-428, 1983