Space complexity of computations

Pavel Pudlák
Mathematical Institute
Czechoslovak Academy of Sciences
Praha, 115 67, Žitná 25,
Czechoslovakia

Stanislav Žák
Institute for Computation
Techniques of the Technical
University
Praha, 128 00, Horská 3
Czechoslovakia

Abstract

Nonuniform space complexity is characterized in several ways: using models of computation based on some set operations performed with subsets of $\{0,1\}^n$, and by the size of contact schemes (two-terminal switching networks). A geometrically defined complexity of partitions of $\{0,1\}^n$ is introduced and is related with the models based on set operations. A language is exhibited such that if a machine is allowed to investigate each input cell at most once, then it requires space $\sqrt{n}$, while the bound $\log n$ can be achieved without this restriction.

Key words: Turig machine, language, nonuniform space complexity, contact scheme, treshold function.

Introduction

The object of this paper is to investigate space complexity of Turing machine computations. We shall study nonuniform complexity; here nonuniformity means that a Turing machine can use restricted additional information for free. In this paper the additional information will be called an oracle.

We restrict ourselves to the languages over the two element alphabet $B = \{0,1\}$; ω denotes the set of positive integers.

Essentially there are two kinds of oracle Turing machines which can be used to define nonuniform space complexity. Let S be a function defined on ω.

(1) An oracle is a sequence $\{a_n\}_{n\in\omega}$, $a_n \in B^*$. The input for a machine T is always a pair (x,a_n), where $x \in B^n$. Then we define: a language $L \subseteq B^*$ is accepted in nonuniform space S, or

$$L \in \text{NU-SPACE}(S),$$

if there is a machine T with an oracle $\{a_n\}_{n\in\omega}$ such that T is S - space bounded and the length of a_n is $\leq c^{S(n)}$ for $n \in \omega$ and some constant c.

(2) An oracle is an arbitrary language $A \subseteq B^*$. The machine has a special work tape - called oracle tape - and is supplied with information, whether the current word on the oracle tape belongs to A or not. Then $L \in$ NU-SPACE(S) if there is

a machine T with an oracle A such that the work tapes, including the oracle tape, are S - bounded.

It is easy to show that these definitions are equivalent. We omit the proof of it, since we need only the first definition. The non_uniform nondeterministic space classes

NU-NSPACE(S)

are defined in the same way using nondeterministic machines.

The main reason for studying nonuniform complexity is that it unifies two different approaches to the complexity: the Turing machine complexity and the Boolean cirquit complexity. There are other reasons for that; for example, it is known that the complexity classes based on random Turing machines are related to nonuniform complexity classes [1,2].

The following is a summary of the paper.

In Section 1 we introduce a model of computation based on some set operations performed with subsets of B^n. We show that the logarithm of the complexity of these algorithms corresponds to nonuniform space.

If a Turing machine is allowed to use space larger than the input length, then it can first write down the input word on the work tape and then it does not need to use the input anymore. The situation becomes quite different if the space bound is sublinear. Then the machine has to investigate the input cells repeatedly. In Section 2 we prove that there is a language such that if a machine can investigate each input cell at most once, then it requires space $\sqrt{n}$, while the bound $\log n$ can be achieved without this restriction.

Aleliunas, Karp, Lipton, Lovász and Reckoff [2] proved that directed graph reachability is in NU-SPACE(log n). This has the important consequence that nonuniform space can be characterized by the size of contact schemes. This will be discussed in Section 3.

A conservative model of computation is, roughly speaking, a model in which the algorithms are allowed to use only some restricted operations and hence such algorithms cannot be regarded as a model of a general purpose computer. In conservative models one can prove lower bounds which are not known for nonconservative ones. In Section 3 we shall extend methods of some geometrically oriented conservative model to a nonconservative case.

1. Cutting algorithms

Suppose the vertices of the n-dimensional cube B^n are of two colours Yes and No. Our task is to separate vertices of different colours using only the two following operations.

(a) We may cut a part of the cube along a coordinate i, (that is we may divide $A \subseteq B^n$ into

$$\{\underline{a} \in A;\ a_i = 0\},\ \{\underline{a} \in A;\ a_i = 1\},$$

where $\underline{a} = (a_1, \ldots, a_n)$.)

(b) We may join different parts of the cube.

If we do not require that every two vertices of the same colour belong to the same piece of the final decomposition, then this can always be done without operation (b). However we want to use as few operations as possible. Operation (b) enables us to reduce essentially the number of cuttings, since we do not have to cut many small pieces seperately. Once operation (b) is allowed, it is convenient to require that the result of cuttings and joinings is the partition {Yes,No} .

There are several possibilities how one can define cutting algorithms precisely. We give only two definitions.

Definition 1

A computational structure is a finite oriented acyclic graph, labeled by subsets of B^n such that

1. it has exactly one source - input vertex - vertex with indegree 0, which is labeled by B^n;
2. every vertex has outdegree 2 except for two sinks - output vertices - vertices with outdegree 0; the labels of the successors of every vertex with outdegree 2 are produced by operation (a) above from its label;
3. only the label of a sink may be empty.

Definition 2

A cutting algorithm is a sequence of partitions of B^n, which starts with the partition $\{B^n\}$ and at every step one operation of the form (a) or (b) is used to produce the next partition.

We say that a computation structure (resp. cutting algorithm) computes - accepts - a set $A \subseteq B^n$ iff A is a label of a sink (a block of the last partition resp.). The cutting complexity of $A \subseteq B^n$ is the minimum of the number of vertices (length of the sequence resp.) in computational structures (cutting algorithm resp.) which compute A. The two possible definitions of cutting complexity give numbers which differ only in a multiplicative factor ≤ 2; thus we shall prefer neither of them. The cutting complexity of a language $L \subseteq B^*$ is the function C, where C(n), is the cutting complexity of $L \cap B^n$.

The labels of a computational structure are quite complex, since they are subsets of B^n. Sometimes it is useful to use a different labeling. Namely, we can label vertices of the structure by coordinates allong which the cuttings are done. To get full information about the computational structure, we have to label the two edges going out of a vertex v by 0 and 1, for every v which is not a sink. Thus we get a computation graph (2-way integer computation graph) of [4]. (Here we have "2" because we use a two element alphabet).

Every input $\underline{a} \in B^n$, $\underline{a} = (a_1, \ldots, a_n)$, determines a path, in a computational structure or in a computation graph, which starts in the source and ends in a sink. In a computational structure the path is determined by the property that $\underline{a}$ is an element of the label of every vertex of the path. In

a computation graph vertex u follows vertex v in the path iff (v,u) is an edge labeled by a_i and v is labeled by i. The path will be called the computation on $\underline{a}$ and denoted by comp($\underline{a}$). The label $A \subseteq B^n$ of a vertex v is, in fact, the set of all $\underline{a} \in B^n$ such that v is in comp(a).

We are going to show that the cutting complexity characterizes the space complexity.

Theorem 1.1.

For a function $F: \omega \to \omega$, let CUT(F) denote the class of languages with cutting complexity bounded by p(F), p a polynomial. If $F(n) \geq n$, for $n \in \omega$, then

$$CUT(F) = NU\text{-}SPACE(\log F) .$$

Proof:

Both simulations are easy; so we shall only sketch them.

Let $\mathcal{C}_n$ be an optimal computational structure for $L_n = L \cap B^n$ and let v_n be the sink of $\mathcal{C}_n$ labeled by L_n. The computation graph corresponding to $\mathcal{C}_n$ and the vertex v_n can be encoded by a string S_n, the length of which is bounded by a polynomial in the number of vertices of $\mathcal{C}_n$. If a suitable code is chosen, there is a log-space Turing machine, which for S_n and $\underline{a} \in B^n$ determines, whether $\underline{a}$ belongs to the label of v_n. The machine just follows the path comp($\underline{a}$) and if it reaches v_n, then $\underline{a}$ is in L_n. If we consider $\{S_n\}_{n \in \omega}$ as an oracle, then we have an oracle Turing machine working in space $\log(n + |S_n|)$. This proves the inclusion from the left to the right.

Now let T be a Turing machine with an oracle. Suppose T accepts L in space S(n), $S(n) \geq \log n$. The description of positions of heads of T, a content of work tape of T, a state of T and the number of steps already performed by T will be called a configuration of T. The initial configuration is the same for all inputs, since the input is not contained in the configuration. Without loss of generality we may suppose that T never loops and ends in one of two special configurations. Let n be given. Then a computation structure for L_n can be defined as follows. The vertices will be the configurations of T, which can be reached during a computation on a word of length n, the label of a configuration K will be the set of all words $\underline{a}$ of length n such that T reaches K when working on $\underline{a}$. Two configurations form an oriented edge if during some computation of T they appear successively. Using contractions we can easily eliminate vertices, which have outdegree one. The number of configurations can be estimated by $k^{S(n)}$ for some constant $k > 0$, which gives the second part of the theorem. □

Proposition 1.2.

For every computational structure $\mathcal{C}$ (which accepts a nontrivial set), there is a computational structure $\mathcal{D}$ such that the number of vertices of $\mathcal{D}$ is smaller or equal to the number of different labels of $\mathcal{C}$ and $\mathcal{D}$ accepts the same set.

Proof:

For a vertex u in $\mathcal{C}$, let ℓ_u denote its label. We define two partial orderings on the set of vertices of $\mathcal{C}$:

$u \geqslant v$ if there is an oriented path from u to v in the graph of $\mathcal{C}$;

$u \geqslant v$ if $u \geqslant v$ and $\ell_u \subseteq \ell_v$.

If $u \geqslant v$ and $u \geqslant w$, then u and v are comparable in $\geqslant$, since v and w lay on the same path comp(a) for some $a \in \ell_u \subseteq \ell_v \cap \ell_w$. Thus, for every vertex u, we can define min(u) as the unique $\geqslant$-minimal vertex such that $u \geqslant \min u$.

Let C be the set of vertices of $\mathcal{C}$, put

$$C_{min} =_{df} \{ min(u) ; u \in C \} = \{ u ; u = min(u) \}.$$

Clearly, all the vertices of C_{min} have pairwise different labels. Both sinks belong to C_{min}. Let $v_o \in C_{min}$ be the vertex with label B^n.

We define a computation graph $\mathcal{D}'$ on C_{min}: If u_1, u_2 are successors of $u \in C_{min}$ in $\mathcal{C}$, then u has the successors $min(u_1)$, $min(u_2)$. The label of $(u, min(u_i))$ equals to the label of (u, u_i), $i=1,2$ in the computation graph associated with $\mathcal{C}$. If it happens that there are vertices in $\mathcal{D}'$ which are not reachable from v_o, they must be omitted.

We shall show that $\mathcal{D}'$ accepts the same sets as $\mathcal{C}$. Let a be an input. For every vertex u of comp(a) also $min(u)$ belongs to comp(a), since $a \in \ell_u \subseteq \ell_{min(u)}$. Hence there is a subsequence of comp(a) which is a computation in $\mathcal{D}'$. This sequence starts in the vertex v_o and ends in the same sink as comp(a) does. Thus the labels of sinks in the computational structure $\mathcal{D}$ associated with $\mathcal{D}'$ are the same as in $\mathcal{C}$. □

Corollary 1.3.

Let $\mathcal{C}$ be a computational structure with the minimal number of vertices such that $\mathcal{C}$ accepts given set A, $0 \neq A \neq B^n$. Then the labels of any two different vertices of $\mathcal{C}$ are different. □

Thus in a minimal computational structure we do not have to distinguish between a vertex and its label. It is not difficult to prove that a minimal computational structure is in fact completely determined by the set of its labels.

2. Algorithms asking at most once

We want to define when is some input value used during a computation. We say that a machine M asks about x_i in configuration K, if there are two inputs a, b such that M reaches K on both a and b, M scans the i-th input letter in K and the next configurations for a and b are different.

We prefer to use computational structures. In Section 1 we have shown that the vertices of a computational structure correspond to configurations of a Turing machine. Hence it is natural to call them configurations too. By Corollary 1.3 we can

assume further that

$$K = \{ a \in B^n ; K \text{ occurs in comp}(a) \},$$

for any configuration K.

Let a computational structure be given, let $a \in B^n$ be an input, $K \in$ comp(a) a configuration in the computation on a. Then we say that comp(a) asks the i-th coordinate in configuration K if the cutting associated with K is along the i-th coordinate. Computational structures, such that every computation asks each coordinate at most once, will be called 1-structures (once asking computational structures). It is a simple fact that every path in a 1-structure is a part of a computation.

Our aim is to show that "one asking" restriction sometimes causes a substantial increase of cutting complexity. Translated for machines: once asking machines require sometimes essentially more space than ordinary ones.

Now, let us present some technical definitions and lemmas. Let us have an input $\underline{a} \in B^n$ and a configuration $K \in$ comp($\underline{a}$). Then we write

$\underline{a}_K$ = { i ; before having reached K, comp($\underline{a}$) asks the i-th coordinate },

$\underline{a}^K$ = {i ; after having reached K, comp(a) asks the i-th coordinate }.

The complement of $X \subseteq \{1,\ldots,n\}$ will be denoted by $\overline{X}$. Clearly, $\underline{a}_K \subseteq \overline{\underline{a}^K}$, for any 1-structure. Let $\underline{a}, \underline{b} \in B^n$, $\underline{a}=(a_1,\ldots,a_n)$, $\underline{b} = (b_1,\ldots,b_n)$. Let S be a set of coordinates. We write

$$\underline{a} =_S \underline{b} \quad \text{iff} \quad (\forall i \in S)(a_i = b_i).$$

Lemma 2.1.

Let a 1-structure be given. Let $\underline{a}$, $\underline{b}$ be inputs and K be a configuration common to comp($\underline{a}$) and comp($\underline{b}$).

a) If $\underline{a}_K = \underline{b}_K$, $\underline{c} = \underline{a}_{\underline{a}_K}$, $\underline{c} = \underline{a}_{\overline{\underline{a}_K}}\ \underline{b}$ then $\underline{c}$ is accepted iff $\underline{b}$ is accepted.

b) $\underline{b}_K \subseteq \overline{\underline{a}^K}$ (i.e. after K, $\underline{b}_K$ is inaccessible not only for comp($\underline{b}$) but also for comp($\underline{a}$)).

c) If $\underline{c} = \underline{a}_{\overline{\underline{b}_K - \underline{a}_K}}\ \underline{a}$ then comp($\underline{c}$) = comp($\underline{a}$).

Proof:

a) comp($\underline{c}$) follows comp($\underline{a}$) until K is reached, then it follows comp($\underline{b}$).

b) Let p be the path which follows comp($\underline{b}$) to K, and which follows comp($\underline{a}$) after K. p must be a computation since we have a 1-structure, (in fact, p = comp($\underline{c}$), where

$\underline{c} = \underline{b}_{\underline{b}_K}\ \underline{b}$, $\underline{c} = \underline{a}_{\overline{\underline{b}_K}}\ \underline{a}$.). Now suppose $\underline{b}_K \cap \underline{a}^K \neq \emptyset$. Then p asks twice an $i \in \underline{b}_K$. A contradiction.

c) comp($\underline{a}$) and comp($\underline{c}$) can branch only in $\underline{b}_K - \underline{a}_K$; but by b) this is unaccessible for comp($\underline{a}$). □

A finite graph $(\{v_i\}_{i=1}^m, E)$ may be given by a 0-1 matrix $(a_{ij})_{i,j=1}^m$, where $a_{ij} = 1$ iff $(v_i, v_j) \in E$. Assuming irreflexivity and symmetry of E, such a graph can be codded by the binary string $a_{12}\ a_{13} \ldots a_{1,m}\ a_{23}\ a_{24} \ldots a_{2,m} \ldots a_{m-1,m}$.

By a half-clique we mean (the code of) any finite graph $G = (V_1 \cup V_2, E_1)$, where card(V_1) = card(V_2) and $E_1 = V_1 \times V_1 - \{(v,v) ; v \in V_1\}$. For simplicity, we shall use the same letter to denote both the code of G and V_1.

Lemma 2.2.

Let a 1-structure accepting the set of half-cliques be given. Let $\underline{a}$ be a half-clique, $\underline{b}$ be any input and K be a configuration common for comp($\underline{a}$), and comp($\underline{b}$). Then $\underline{b}_K - \underline{a}_K = \emptyset$. (If $\underline{b}$ is also a half-clique then $\underline{b}_K = \underline{a}_K$.).

Proof:

Suppose $\underline{b}_K - \underline{a}_K \neq \emptyset$. By Lemma 2.1. c) there is an input $\underline{c}$ such that comp($\underline{c}$) = comp($\underline{a}$) and such that $\underline{c}$ is not a half-clique. A contradiction. □

Theorem 2.3.

For 1-structures the language of half-cliques is of complexity at least $2^{\sqrt{n}/6}$.

Let a once asking machine be any machine which generates only 1-structures.

Corollary 2.4.

For once asking machines the language of half-cliques is of space complexity at least $\sqrt{n}$.

The proof is based on the intuitive idea that, treating the last vertices of a half-clique, the computation "has to realize" what are the other vertices of this half-clique. Generally this is possible in two ways: either to remember all these vertices, or to ask once again. For 1-structures the second possibility is forbidden.
Now, in order to remember many half-cliques it is necessary to have many configurations.

Proof of Theorem 2.3.:

Let us fix a cube B^n. Each $\underline{a} \in B^n$ can be considered as a code of a graph with m vertices, where m is maximal such that $(m^2 - m)/2 \leq n$, and therefore $m \geq \sqrt{n}$.

For a half-clique $\underline{a}$, let $F(\underline{a})$ be the first configuration in $comp(\underline{a})$ such that the edges which are asked in configurations preceeding $F(\underline{a})$ and in $F(\underline{a})$ cover at least $m/2 - 2$ vertices of the clique of $\underline{a}$.

We choose a maximal set S of half-cliques such that each two half-cliques in S differ on at least 6 vertices. The cardinality of S is at least

$$\binom{m/3}{m/6} \geq 2^{\sqrt{n}/6}.$$

(To see this, group the vertices into triads.). Now it suffices to prove that our F is 1-1 on S.

Suppose $\underline{a}, \underline{b} \in S$, $\underline{a} \neq \underline{b}$ and $F(\underline{a}) = F(\underline{b}) =$

By Lemma 2.2., $\underline{a}_K = \underline{b}_K$, and we can choose an input $\underline{c}$ such that $\underline{c} = \underline{a}_K\, \underline{a}$, $\underline{c} = \overline{\underline{a}_K}\, \underline{b}$. By Lemma 2.1. a), $\underline{c}$ is a half-clique too.

Let $v \in \underline{a} - \underline{b}$ be a vertex with at least one edge in $\underline{a}_K$; hence $v \in \underline{c}$. Let $u \in \underline{b}$ be a vertex with no edge in $\underline{b}_K$; let $w \in \underline{b} - \underline{a}$, $w \neq u$. Clearly $w \in \underline{c}$. But there is no edge between v and w in $\underline{c}$, since in $\underline{c}$ there are only edges which are in $\underline{a}$ or in $\underline{b}$. Therefore c is not a half-clique.
A contradiction. □

It is not difficult to construct a once asking machine which recognizes the language of half-cliques within the $\sqrt{n}$ space bound (and therefore it is an optimal once asking machine), and that there is a twice asking machine which recognizes this language within the log n space bound. We see that the number of asking is important, since a small change of it dramatically reduces the complexity bound. - There are many questions about hierarchies with respect to asking and space, asking-space trade-offs and so on. But we do not know even how to prove that there is a language in P or NLOG which cannot be accepted by twice asking machines within the log space bound.

3. Contact schemes

A contact scheme, or two terminal switching network, is a finite undirected loop free multigraph, the edges of which are labeled by n varibles and their negations $x_1,\dots,x_n$, $\bar{x}_1,\dots,\bar{x}_n$, with two distinguished vertices - terminals. Every word $\underline{a} \in B^n$ determines a submultigraph of the multigraph of the scheme, which consists of the same set of vertices and of the edges labeled x_i resp. $\bar{x}_i$ if $a_i = 1$ resp. $a_i = 0$. The scheme accepts the set $A \subseteq B^n$ of the words, for which the terminals are connected in the associated submultigraph. Thus a word $\underline{a} \in B^n$ is accepted iff there is a path connecting the two terminals satisfying the following condition: For every edge of the path, the edge is labeled by x_i iff $a_i = 1$.

A contact gating scheme is a generalization of a contact scheme, where the multigraph is directed and acyclic.

The size of a scheme is the number of edges. This determines in a natural way two complexities (one for contact schemes, one for contact gating schemes) of subsets of B^n and of languages in the alphabet B.

Our aim is to show that size of a minimal contact scheme characterizes nonuniform space. The following modification of computation graphs will be useful in the proof. A contact gating scheme $\mathcal{S}$ will be called deterministic contact scheme if

(i) every vertex in $\mathcal{S}$, except for one of the terminals - the output terminal, has outdegree 1 or 2;

(ii) if a vertex has outdegree 2, then the out-going edges are labeled complementarily (i.e. x_i and $\bar{x}_i$ for some $i = 1,\dots,n$).

Lemma 3.1.

Let $\mathcal{S}$ be a deterministic contact scheme, let $\mathcal{S}'$ be the contact scheme which results from $\mathcal{S}$ if the orientation of the edges is forgotten.
Then $\mathcal{S}'$ accepts the same set.

Proof:

Suppose $\mathcal{S}$ is labelled by n variables and let $\underline{a} \in B^n$. Let T be the subgraph of $\mathcal{S}$ determined by $\underline{a}$. We have to show that if there is an unoriented path connecting the terminals, then there is an oriented one too. Every vertex has outdegree ≤ 1 in T and the output terminal has outdegree 0. That is, the components of T are oriented trees and the output terminal is the root of some tree. In an oriented tree, there is an oriented path from any vertex to the root. Therefore if the terminals are in the same component, than there is an oriented path from the input terminal to the output terminal. □

Theorem 3.2.

For a function $F: \omega \to \omega$, let CS(F) (resp. DCS(F)) be the class of languages with contact scheme (resp. deterministic contact scheme) complexity bounded by p(F), p a polynomial. Let $F(n) \geq n$, $n \in \omega$, then

$$CS(F) = DCS(F) = NU\text{-}SPACE(\log F) .$$

Sketch of the proof:

We have to prove three inclusions:

$$DCS(F) \subseteq CS(F) \subseteq NU\text{-}SPACE(\log F) \subseteq DCS(F).$$

The first one is a consequence of Lemma 3.1. The proof of the second inclusion is based on a result of [2] which says that reachability in unoriented graphs is in NU-SPACE(log n).

It is easy to adapt any algorithm for reachability in such a way that it determines for a pair $(\underline{a}, \mathcal{S})$, $\underline{a} \in B^n$, $\mathcal{S}$ a (code of a) contact scheme, whether $\mathcal{S}$ accepts $\underline{a}$. Now, as in the proof of Theorem 1.1., we can take the optimal contact schemes as an oracle, (along with the universal sequences of [2]). Then the machine will work in space bounded by logarithm of the size of these schemes.

The last inclusion follows from the fact observed in Section 1. that Turing machines can be simulated by computation graphs. The transformation of a computation graph to a deterministic contact scheme is simple: In fact, we have only to omit vertices from which the output vertex cannot be reached and change the labelling. □

A similar theorem can be proved for size of contact gating schemes and nonuniform nondeterministic space.

Borodin [3] related formula size to deterministic and nondeterministic space. His proof is valid also for nonuniform complexity. If we denote by L(F) the class of languages with formula size complexity p(F), p-polynom, then the result reads

$$L(F) \subseteq \text{NU-SPACE}(\log F);$$

$$\text{NU-NSPACE}(\log F) \subseteq L(F^{\log F}),$$

for $F(n) \geq n$, $n \in \omega$. It seams to us that neither inclusion can be improved to the equality.

Nečiporuk [11] has constructed a concrete language (in fact a sequence of Boolean functions) $L \subseteq B^*$ and proved that its formula size complexity is $\geq c_1 \, n^2 \log^{-1} n$, and contact scheme complexity is $\geq c_2 \, n^2 \log^{-2} n$, c_1, c_2 - constants. Using his method one can show that the size of contact gating scheme for L is $\geq c_3 \, n^{3/2} \log^{-1} n$. For combinational complexity (= size of minimal Boolean cirquits), there is no concrete L, for which a nonlinear lower bound is known.

Nonuniform nondeterministic space can be characterized also by set operations on B^n, similarly as nonuniform space is characterized by cutting algorithms. The difference is very subtle: while in deterministic case we lose every set to which an operation is applied, in nondeterministic case we can store any set already constructed. That is, instead of a sequence of partitions in the cutting algorithm, we have a general sequence of sets of subsets of B^n. If we allow also the operation of intersection, then we obtain a kind of algorithms the complexity of which characterizes nonuniform time.

It is well-known that the minimal number of states of a two way automaton can also be used to characterize nonuniform complexity classes. Thus we have four basic approaches to nonuniform complexity; they are schematically depicted on Figure 1. For every two entries in a row a theorem of the form of Theorem 1.1. or 3.2. can be proved, cf. [2,4,6,7].

Turing machines with oracles	Automata	Cirquits	Set operations
NU-SPACE	deterministic two-way automaton	contact schemes	cutting along a coordinate, union, sets cannot be stored
NU-NSPACE	nondeterministic two-way automaton	contact gating schemes	cutting along a coordinate, union, sets can be stored
NU-TIME	alternating two-way automaton	Boolean cirquits	cutting along a coordinate, union, intersection, sets can be stored

log

Figure 1.

4. A geometrical approach

Consider the hypercube B^n as a subset of the n-dimensional Euclidean space E_n. Then the operation of cutting along the i-th coordinate corresponds to a separation by a hyperplane perpendicular to the i-th axis. This suggests a natural question, whether we can gain essentially more by using arbitrary hyperplanes. We shall show that the answer is no, and find relations to some conservative model of computation.

Definition

(1) A sequence $P_0, P_1, \ldots, P_m$ of partitions of B^n will be called an affine algorithm if

(i) $P_0 = B^n$,

(ii) for $i = 1,\ldots,n$, P_i is either coarser than P_{i-1} or there exists a block $A \in P_{i-1}$ and real numbers $a_1,\ldots,a_n$, b such that P_i results from P_{i-1} when A is split into two blocks

$$A_0 = \{ \underline{x} \in A \; ; \; \sum a_i x_i \geq b \} ,$$

$$A_1 = A \setminus A_0 .$$

(2) A deterministic affine scheme will be an oriented acyclic graph the edges of which are labelled by inequalities $a_1 x_1 + \ldots + a_n x_n \geq b$ or $a_1 x_1 + \ldots + a_n x_n < b$, one vertex (the input) has indegree 0, one vertex (the output) has outdegree 0 and

(i) every vertex, except for the output vertex has outdegree 1 or 2,

(ii) if a vertex has outdegree 2 then the outgoing edges are labelled by complementary inequalities. A word $\underline{x} \in B^n$ is accepted by such a scheme iff it satisfies all inequalities along an oriented path from the input to the output.

It is obvious that every cutting algorithm is an affine algorithm and every deterministic contact scheme is, in fact, a special kind of deterministic affine scheme.

Lemma 4.1.

(1) There exists a constant C such that every threshold function can be represented in the form

$$a_1 x_1 + \ldots + a_n x_n \geq b,$$

where a_i, b are integers and $|a_i|, |b| \leq C^n$ for $i = 1,\ldots,n$.

(2) Inequality

$$a_1 + \ldots + a_n \geq 0,$$

where a_i are integers in the binary representation, can be decided in deterministic logarithmic space.

Proof:

(1) See for example Muroga [10].

(2) If $|a_i| \leq n$ for $i = 1,\ldots,n$, then we can decide the inequality simply by computing $\sum a_i$. If this is not the case, let a_t be the number with the largest absolute value. Let $\lceil \log n \rceil + m$ be the number of digits of a_t, where $\lceil \log n \rceil$ is the smallest integer $\geq \log_2 n$. Let a_i', a_i'' be the integers such that

$$a_i = a_i' \cdot 2^m + a_i'', \quad |a_i''| < 2^m, \quad a_i' \cdot a_i'' \geq 0.$$

Then we can compute $A = \sum a_i'$ logarithmic space. Now the algorithm splits into three cases.

a) If $A \geq 2^{\lceil \log n \rceil}$, then $\sum a_i \geq 0$ since

$$\sum a_i = \sum a_i' \cdot 2^m + \sum a_i'' \geq A \cdot 2^m - \sum |a_i''| >$$

$$> A \cdot 2^m - n \cdot 2^m \geq 0.$$

b) If $A \leq -2^{\lceil \log n \rceil}$, then $\sum a_i < 0$ by a similar computation.

c) If $|A| < 2^{\lceil \log n \rceil}$, then put

$$a_1 := A \cdot 2^m + a_1'',$$

$$a_i := a_i'' \quad \text{for } i = 2,\ldots,n,$$

and repeat the procedure. We do not have to store new a_i's, all the information is given by the input, A, and m. The last two numbers require only logarithmic space, since

$$|A| < 2^{\lceil \log n \rceil}, \; m < n. \quad \square$$

Now we can prove that the size of a minimal deterministic affine scheme characterizes nonuniform space in the same manner as the contact schemes do. The minimal number of steps of an affine algorithm is at most twice larger than the size of a minimal deterministic affine scheme. Also one can easily generalize the following theorem for affine schemes in the spirit of Theorem 3.2.

Theorem 4.2.

For a function $F: \omega \to \omega$, let AS(F) denote the class of languages with deterministic affine complexity bounded by p(F), p a polynomial. If $F(n) \geq n$, for $n \in \omega$, then

AS(F) = NU-SPACE(log F).

Sketch of the proof:

(1) Every deterministic contact scheme is a deterministic affine scheme, hence AS(F) $\supseteq$ NU-SPACE(log F).

(2) To prove the converse inclusion we have only to find a suitable simulation of an affine scheme by a Turing machine. Given an affine scheme $\mathcal{A}$, the Turing machine will have a code of $\mathcal{A}$ on its oracle tape. Let n be the length of inputs and k the size of $\mathcal{A}$. By Lemma 4.1. (1), we can choose a code of $\mathcal{A}$ the length of which is bounded by a polynomial in n and k. Hence log n + log k space is enough to control the oracle head. Now, by Lemma 4.1. (2) log n space will suffice to compute the threshold functions of $\mathcal{A}$. Therefore the simulation can be performed in log n + log k space. □

Corollary 4.3.

There is a polynomial p(n) such that every n-dimensional threshold function can be computed by a contact scheme of size p(n).

Let us compare deterministic affine schemes with a conservative model of computation studied by J. Morávek. In 1967 he introduced the concept of linear separating algorithm [8], later it was called linear comparison algorithm [9], (cf. also [5]). The concept is close to affine scheme, thus we list only the differences instead of presenting the definition:

(1) trichotomic branching is used instead of dichotonic branching of ours,

(2) only trees are used instead of general acyclic graphs; this also requires more outputs,

(3) the set of inputs is E_n instead of B^n,

(4) the measure of complexity is the depth, i.e. the length of the longest oriented path of the scheme.

Clearly, the difference (1) is not important. Because of (4), neither (2) is relevant. The main difference is therefore (3). There are partitions of E_n that require arbitrarily large depth, some partitions of E_n cannot be realized by a linear comparison algorithm at all. On the other hand every partition of the discrete set B_n can be trivially realized with depth n. Therefore it is not clear how to make use of the superlinear lower bounds of Morávek in the nonconservative case.

The question about the depth makes sense for affine schemes too. By Shannon [14], most functions $f : B^n \to B$ have complexity about $2^n/n$, if the complexity is measured by the number of edges of a contact scheme realizing f. Hence, by Theorems 3.2. and 4.2., the deterministic affine complexity of most functions $f : B^n \to B$ must be at least 2^{cn} for some $c > 0$. Therefore the depth of deterministic affine schemes must be at least c.n for these functions. We do not have an example of a concrete sequence of such functions.

We shall adapt a concept of Morávek for our model of computation. We shall call a partition $\{A_1,\dots,A_m\}$ of B^n convex, if, for every $1 \le i < j \le m$, the convex hulls of A_i and A_j in E_n are disjoint.

Definition

The convex complexity of a partition P of B^n is the smallest m such that there is a convex refinement $\{A_1,\dots,A_m\}$ of P, The convex complexity of a Boolean function is the

convex complexity of the corresponding partition.

The convex complexity is an analogy of the index of convexity of [9].

Theorem 4.4.

If a Boolean function f has convex complexity m, then every affine scheme, which realizes f, has depth at least $\log_2 m$.

The proof is the same as the proof of Theorem 3.6. in [9]. □

We are going to show how one can approximate the convex complexity by the complexity of some restricted affine algorithms. Let the sequence of partitions $P_0,\dots,P_m$ be an affine algorithm $\mathcal{A}$. Then the width of $\mathcal{A}$ is

$$\max_{i=0,\dots,m} |P_i|$$

where $|P_i|$ is the number of blocks in P_i. (Cf. [12], where a similar concept is defined). We define, for a Boolean function f,

$$AA_k(f) = m,$$

where m is the smallest number such that there is an affine algorithm $P_0,\dots,P_m$ with width $\leq k$.

Theorem 4.5.

If m is convex complexity of f, then

$$\sqrt{AA_4(f)} \leq c_1 m \leq c_2 AA_3(f),$$

where c_1, c_2 are positive constants.

Proof:

(1) Let $P_0,\dots,P_m$ be an affine algorithm. Suppose each P_i, $i>0$, has at least two blocks. Let $Q_0 = P_0, Q_1, Q_2, \dots, Q_\ell = P_m$ result from $P_0,\dots,P_m$ when the partitions with three blocks are omitted. Then it holds for $i = 2,3,\dots,\ell$:

$$Q_i = \{C \setminus S,\ D \cup (C \cap S)\},$$

where $Q_{i-1} = \{C, D\}$ and S is a halfspace in E_n. With every Q_i, $i \geq 1$, we associate a convex refinement R_i with i+1 blocks as follows:

$$R_1 = Q_1;$$

if Q_{i-1} and Q_i are as above and $R_{i-1} = A_1,\dots,A_i$ then

$$R_i = \{A_1 \setminus S,\dots,A_i \setminus S,\ \{0,1\} \cap S\}.$$

Hence P_m has a convex refinement with at most m+1 blocks. This proves the second inequality.

(2) Let a partition $P = \{\bigcup_{i=1}^{j} A_i,\ \bigcup_{i=j+1}^{m} A_i\}$ be given, where $\{A_1,\dots,A_m\}$ is a convex partition. We are going to describe briefly a procedure which can be easily interpreted as the work of an affine algorithm with width ≤ 4.

Four cells C_1, C_2, C_3, C_4 are given. The first two are work cells; in the third the words which has been accepted are stored; the fourth serves for transporting words between C_1, C_2, C_3. Initially all the words of B^n are in C_1, the others being empty. At every step of computation we separate a part of the content of some cell C_i, i=1,2,3, by a hyperplane, and translate it into another one using C_4. This is quite similar as in the first part of the proof.

Since $A_1,\dots,A_m$ is convex, for every $i>1$, there is a hyperplane that separates A_1 and A_i. Thus we need only m-1 steps to translate the words of $B^n \setminus A_1$ into C_2. Then the content of C_1, which is the set A_1, is put in C_3. The same procedure is applied to A_2, $A_3,\dots,A_j$. Thus we shall get the first block of P in C_3, the second will remain in C_1 or C_2. The number of steps is bounded by m^2, which proves the first inequality. □

Unfortunately we lack an efficient method for evaluation of the convex complexity. Let $f_n : B^n \to B$ be the sum modulo 2 function that is $f_n(\underline{a}) = 1$ iff the number of ones in $\underline{a}$ is odd. Then the convex complexity of f_n is $\leq n+1$. (One can take the partition into n+1 blocks determined by the number of ones.). We conjecture that convex complexity of f_n is exactly n+1. What we can prove is however much weaker.

Proposition 4.6.

Convex complexity of the sum modulo 2 function f_n is $\geq c.\log n \,.\, (\log\log n)^{-1}$, for some $c>0$.

Proof:

Using induction over n we prove the following assertion:

(*) If $\{A_1,\dots,A_k\}$ is a convex partition of B^n such that the sum modulo 2 function f_n is constant on every block of it, then

$$k! \,.\, (1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{(k-1)!}) \geq n \,.$$

Since the lefthand side is bounded by $e.k^k$, the theorem follows from (*).

For n = 1 is (*) trivial. Suppose (*) is true for every n, $n<m$. We shall prove (*) for n = m. Let such a partition $\{A_1,\dots,A_k\}$ of B^m be given. Let $\underline{e}_1,\dots,\underline{e}_m \in B^m$ be the words with exactly one 1. There is a block that contains at least m/k of these words, and it must be different from the block that contains $\underline{0}$. We can assume that $\underline{0} \in A_1$, $\underline{e}_1,\dots,\underline{e}_j \in A_2$, $j \geq m/k$. Let k be the subcube of B^m generated by $\underline{e}_1 \dots \underline{e}_j$. Then the convex hull of $\underline{e}_1,\dots,\underline{e}_j$ separates $\underline{0}$ from the other points of k, that is $\underline{0}$ is the unique point of $K \cap A_1$. Let L be the subcube of K with dimension j-1 which contains $\underline{e}_1$, but not $\underline{0}$. Then $\{A_2, A_3,\dots,A_k\}$ induces a convex partition on L with ℓ blocks, $\ell \leq k$, and such that f_m is constant on it. Clearly we can apply the induction assumption to such a partition, hence

$$(k-1)!(1 + \frac{1}{1!} + \dots + \frac{1}{(k-2)!}) \geq$$

$$\geq \ell! \,(1 + \frac{1}{1!} + \dots + \frac{1}{(\ell-1)!}) \geq j - 1.$$

Using $j \geq m/k$ we get (*) for n = m.

This ends the proof. □

References

1 L. Adleman, Two Theorems on Random Polynomial Time. Proc. 19-th IEEE Symp. on Foundations of Computer Science, (1978), pp. 75-83.

2 R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász, C. Reckoff, Random Walks, Universal Traversal Sequences, and the Complexity of Maze. Proc. 20-th IEE Symp. on Foundations of Computer Science, (1979), pp. 218-223.

3 A. Borodin, On Relating Time and Space to Size and Depth. SIAM J. on Computing, 6 (1977),4, pp. 733-744.

4 A. Borodin, S. Cook, A Time-Space Trade-off for Sorting on a General Sequential Model of Computation. Proc. 12-th ACM Symp. on Theory of Computing, (1980), pp. 294-301.

5 D. Dobkin, R.J. Lipton, A Lower Bound of 1/2 n^2 on Linear Search Programs for the Knapsac Problem. J. Computer and Systems Sciences, 16 (1978), 3, pp. 413-417.

6 M.J. Fischer, Lectures on Network Complexity. Preprint, University of Frankfurt 1974.

7 R.M. Karp, R.J. Lipton, Some Connections Between Nonuniform and Uniform Complexity Classes. Proc. 12-th ACM Symp. on Theory of Computing, (1980), pp. 302-309.

8 J. Morávek, On the Complexity of Discrete Programming Problems. Aplikace matematiky, 14 (1969), 6, pp. 442-474.

9 J. Morávek, A. Geometrical Method in Combinatorial Complexity. Aplikace matematiky, 26 (1981), 2, pp. 82-96.

10 S. Muroga, Threshold Logic and Its Applications. John Wiley & Sons, 1971.

11 E.I. Nečiporuk, Ob odnoj bulevskoj funkcii. Doklady AN SSSR, 169 (1966), 4, pp. 765-766.

12 N. Pippenger, On Simultaneous Resource Bounds. Proc. 20-th IEEE Symp. on Foundations of Computer Science, 1979, pp. 307-311.

13 W. Ruzzo, On Uniform Cirquit Complexity. - ibidem, pp. 312-318.

14 C.E. Shannon, The Synthesis of Two-Terminal Switching Cirquits. Bell Syst. Techn. J., 28 (1949), 1, pp. 59-98.