FAST ALGORITHMS FOR N-DIMENSIONAL RESTRICTIONS OF HARD PROBLEMS

by

Friedhelm Meyer auf der Heide

IBM Research Laboratory, San Jose, CA 95193

Abstract : Let M be a parallel RAM with p processors and arithmetic operations addition and subtraction recognizing $L \subset N^n$ in t steps. Then L can be recognized by a (sequential!) linear search algorithm (LSA) in $O(n^4(\log(n) + t + \log(p)))$ steps. Thus many n-dimensional restrictions of NP-complete problems (binary programming, traveling salesman problem, etc.) and even that of the uniquely optimum traveling salesman problem, which is Δ_2^p-complete, can be solved in polynomial time by an LSA. This result generalizes the construction of a polynomial LSA for the n-dimensional restriction of the knapsack problem previously shown by the author, and destroys the hope of proving nonpolynomial lower bounds for any problem which can be recognized by a PRAM as above with $2^{\text{poly(n)}}$ processors in poly(n) time.

INTRODUCTION

Linear search algorithms (LSA's) have turned out to be a realistic and comfortable computation model for proving lower time bounds for a large variety of interesting problems. Such an algorithm is an abstraction of a random access machine (RAM).

These RAM's have the capability of executing direct or indirect storage access, if-questions, addition, or subtraction in one step (see [1] or, tailored to our use, [2], [3] or [4]). They recognize languages $L \subset N^n$ or $L \subset N^*$ and read the input integer by integer (not bit by bit).

When dealing with LSA's one usually allows inputs consisting of n real numbers , only counts if-questions as computation steps , and assumes that they all are of the form "If $f(\bar{x}) > 0$ then goto if-question1 else goto if-question2" , where $\bar{x} \epsilon R^n$ is the input and $f:R^n \to R$ is an affine function, i.e. $f(\bar{x}) = \bar{a}\bar{x} - b$ for some $\bar{a} \epsilon R^n, b \epsilon R$. An if-question as above is called to be defined by f. We represent LSA's by rooted, binary trees whose root and inner nodes are labelled with predicates of the form "$f(\bar{x}) > 0$" for some f as above. The leaves are labelled with "accept" or "reject". A computation started with some input $\bar{x} \epsilon R^n$ consists of traversing the tree from the root to a leaf, always choosing the left or right branch of a node according to whether its predicate is fulfilled or not. The LSA accepts the language of all inputs arriving at an accepting leaf.

The reason why LSA's are comfortable for proving lower bounds is their nice geometrical structure. As each if-question defined by some affine function $f:R^n \to R$ subdivides the input set in the two halfspaces $\{\bar{x} \epsilon R^n \mid f(\bar{x}) > (resp.\ \leq) 0\}$ of the hyperplane $\{\bar{x} \epsilon R^n \mid f(\bar{x}) = 0\}$, the inputs arriving at some node of an LSA form a (convex) polytope. Lower bounds for LSA's can for example be found in [4],[5], or [6].

On the other hand, LSA's provide a realistic computation model in the sense that many lower bounds for LSA's can be carried over to RAM's (see [2],[3], and, for a general lower bound for RAM's, [4, *theorem*3]).

In [7] it is shown by the author that LSA's are surprisingly strong. They can solve the NP-complete (compare [8]) knapsack problem (input: $\bar{x} \varepsilon R_+^n$, query: $\exists \bar{\alpha} \varepsilon \{0,1\}^n$ *s.t.* $\bar{\alpha}\bar{x} = 1$) in polynomial time. The reason for this strength of LSA's is that they only deal with inputs consisting of a fixed number n of variables, i.e. they only handle n-dimensional restrictions of given problems. Thus, while a program solving e.g. the knapsack problem must have constant length (independent of the number of input variables), the length of an LSA for its n-dimensional restriction may depend on n. In fact, the length of the LSA shown in [7] is exponential in n. Its main importance is to pinpoint the limits of proving large lower bounds on LSA's.

In this paper we will take a closer look at these limits. For this purpose we consider parallel RAM's (PRAM's). Such a PRAM consists of p RAM's as described above, its processors, and an additional common memory consisting of infinitely many registers, which can be accessed directly or indirectly by the processors.

The main result of this paper is the following:

If $L \subset N^n$ can be recognized by a PRAM with p processors in t steps, then a (sequential!) LSA can recognize L in time polynomial in n, t, and log(p).

Now one can show the existence of polynomial LSA's for given problems by simply designing polynomial parallel algorithms for them using $2^{poly(n)}$ processors. This method yields for example polynomial LSA's for NP-complete problems as the traveling salesman problem, binary programming, and integer programming with bounded solution size. Furthermore we obtain a polynomial LSA for the problem of deciding whether an instance of the traveling salesman problem has a unique optimal solution. This problem is shown to be complete in Δ_2^P, the class of all problems that can be solved in polynomial time using oracles from NP (see [9]).

The paper is organized as follows. In the first section we define PRAM's in more detail, state our theorem, and show applications. In the second section we outline the proof, state the two basic lemmas, and conclude the theorem from them. The sections three and four contain the proofs of the two lemmas mentioned above. In the last section, some open problems are discussed.

SECTION 1

A PRAM M consists of a finite number p of processors $P_1,...,P_p$ and a common memory. Each P_i has two private registers a_i and b_i. The common memory consists of infinitely many registers $m_j, j \varepsilon N$. Each register can store one integer. $\#a_i$, $\#b_i$, $\#m_j$ denote the current content of the respective register. M is assumed to be synchronized. In the beginning of a computation, the input $\bar{x} = (x_1,...,x_n) \varepsilon N^n$ is stored in the first n common registers. The computation of M started with $\bar{x}$ proceeds in phases each consisting of four steps, namely a read step, an arithmetic step, a write step, and a Boolean step. In a read step, each P_i stores $\#m_{\#b_i}$ into a_i. In an arithmetic step, each P_i stores $\#a_i + \#b_i$, $\#a_i - \#b_i$, or a constant $\leq i$ into a_i or b_i. (Note that we allow arbitrary constants $\leq i$ whereas usually one only allows 0, 1, and its address i to be used by P_i.) In a write step, each processor P_i tries to write $\#a_i$ into $m_{\#b_i}$. Hereby a write conflict may arise when several processors want to write into the same register. We use the most general method to solve this conflict considered in literature by assuming that the processor with largest index succeeds. In a Boolean step, each processor asks an if-question "If $\#a_i > 0$ then execute instruction

I, else instruction J". M stops if P_1 stops, and accepts, if finally $\#a_1=1$. The complexity of M is the maximum number of steps, M started with some input $\bar{x}\epsilon N^n$ executes.

We will prove the following theorem in this paper.

THEOREM: Let $L \subset N^n$ be recognized by a PRAM with p processors in t steps. Then L can be recognized by an LSA in $6n^4(\log(n)+t+\log(p))+O(n^3)$ steps.

We now give some applications of this theorem. All algorithms for PRAM's mentioned below are trivial, and are not explained in this paper.

First we consider some NP-complete problems (compare [8]).

Integer Programming with Solutions $\leq k$ (n variables, m inequalities) (Input: m linear inequalities with variables $x_1,...,x_n$. Query: $\exists(x_1,...,x_n)\epsilon\{0,...,k\}^n$ which fulfills all the inequalities.)

(Note that for k=1 we have defined the ***Binary Programming*** problem.)

This problem can be solved by a PRAM with $(k+1)^n$ processors in O(nm) steps, thus by an LSA (for inputs from $R^{(n+1)m}$) in $O((nm)^4(nm+n\log(k+1)))$ steps.

We note here that in [4], an $\Omega(n^2\log(k+1))$ lower bound for m=1 on LSA's and RAM's is shown generalizing the bounds from [6] and [3] for m=1, k=1.

Traveling Salesman Problem (n towns) (Input: nxn-matrix of distances between all pairs of towns, and a number k. Query: There is a roundtrip visiting each town exactly once with total length at most k.)

This problem can be solved by a PRAM with n! processors in O(nlog(n)) steps, thus by an LSA (for inputs from R^{n^2}) in $O(n^9\log(n))$ steps.

We now consider the following generalization of the traveling salesman problem, namely the

Uniquely Optimum Traveling Salesman Problem (n towns) (Input: nxn-matrix of distances between all pairs of towns. Query: There is a unique shortest roundtrip visiting each town exactly once.)

This problem is shown in [9] to be Δ_2^P-complete. Also this problem can obviously be solved by a PRAM with n! processors in O(nlog(n)) steps, thus by an LSA in $O(n^9\log(n))$ steps.

SECTION 2

The proof of the theorem is not done by simulating a PRAM step by step by an LSA. Instead we do the following. We first show that languages recognized by a PRAM with p processors in t steps have a certain structure; we call them q-languages where q denotes a parameter dependent on t and p. Then we show that q-languages can be recognized fast by LSA's.

In order to define q-languages let $F=\{f_1,...,f_m\}$ be a set of affine functions $f_i:R^n \to R$. Then the following languages are called F-languages.

1) $\{\bar{x}\epsilon R^n \mid f_i(\bar{x}) : 0\}$, i=1,...,m, ":"$\epsilon\{=,<,>\}$, are F-languages.
2) Unions and intersections of F-languages are F-languages.

In order to interpret F-languages geometrically, and for later use, we now give some geometrical definitions. Let $f:R^n \to R$ be an affine function. The hyperplane $\{\bar{x}\epsilon R^n \mid f(\bar{x})=0\}$ and its halfspaces are called to be defined by f. The intersection of a finite number of halfspaces and

hyperplanes is a (convex) polytope. Let F be the set of affine functions described above, and let $H_1,...,H_m$ be the hyperplanes defined by $f_1,...f_m$. Then for each $A \subset \{1,...,m\}$, each polytope $\bigcap_{i\in A} H_i - \bigcup_{i\in\{1,..,m\}-A} H_i$ consists of is called a face of F, or a face of the language $\bigcup_{i=1}^{m} H_i$.

With these definitions it is easily seen that each F-language is a union of some faces of F.

Now let F_q denote the set of all affine functions $f:R^n \to R$ with coefficients from {-q,...,q}. An F_q-language is called a *q-language.*

The proof of the theorem is based on the following two lemmas.

***LEMMA 1** : Let M be a PRAM with p processors recognizing $L \subset N^n$ in t steps. Then there is a $p2^t$-language L' with $L = L' \cap N^n$.*

***LEMMA 2** : Each q-language can be recognized by an LSA in $6n^4(\log(n)+\log(q))+O(n^3)$ steps.*

The ***proof of the theorem*** now is done by inserting the bound for q from lemma 1 into lemma 2. Therefore it remains to prove these two lemmas.

SECTION 3

In order to prove lemma 1 we will "unroll" the above PRAM M to a certain kind of parallel computation tree (compare [10], resp. [2] or [3] for the sequential case). In [10] a similar construction is shown. But it has the disadvantage that the resulting computation tree does not simulate M for all inputs, which is not sufficient for our purpose. Therefore we will define a generalized computation tree T with p processors (p-GCT).

A p-GCT is a finite, rooted tree levelled according to the four phases of the PRAM. For $s \in N$ the levels 4s-3, 4s-2, 4s-1, and 4s contain read, arithmetic, write, and Boolean nodes, resp.. Read and write nodes have 2^{n+4ps}, arithmetic nodes 1, and Boolean nodes 2^p sons. The 0'th level contains the root. It has one son.

Each node v represents a configuration of the simulated PRAM. It is represented by an instruction list, a storage list, and a computation list. The instruction list $(I_1^v,...,I_p^v)$ contains the instructions to be executed by the processors. The storage list $(L_i^v,R_i^v,i = 1,...,s_v)$, $L_i^v,R_i^v:N^n \to N$, describes the state of the common memory in the following way : If, started with some input $\bar{x} \in N^n$, M chooses the computation given by the path to v, then $m_{L_i^v(\bar{x})}$ contains $R_i^v(\bar{x})$ for $i = 1,...,s_v$, and all other m_j's contain 0. The computation list $(A_i^v,B_i^v,i = 1,...,p)$, $A_i^v,B_i^v:N^n \to N$, describes the contents of the private memories: $\#a_i = A_i^v(\bar{x})$, $\#b_i = B_i^v(\bar{x})$. Each edge e is labelled with its restriction set $C_e \subset N^n$ of inputs $\bar{x}$, such that M started with $\bar{x}$ executes the computation described by the path to e.

Such a p-GCT is said to simulate M. We may w.l.o.g. assume that each leaf v either fulfills $A_1^v \equiv 1$ or $A_1^v \equiv 0$. In the first case , v is accepting, in the latter case it is rejecting. Thus the language L recognized by M is the union of the restriction sets belonging to edges which are incident to accepting leaves. The following claim will show that the restriction sets are q-languages for a certain q.

***CLAIM 1** : There is a p-GCT T simulating M, such that for each node v and each edge e in depth s, the storage and the computation list of v consist of functions from F_{p2^s}, and C_e is a $p2^s$-language.*

Now we can easily conclude lemma 1. As by claim 1, each set C_e for an edge incident to an accepting leaf is a $p2^t$-language, L, the union of all these sets, is also a $p2^t$-language, as claimed in lemma 1. Thus it remains to prove claim 1.

Proof of claim 1 : Suppose we are given a - still unlabelled - tree T as described in the above definition of a p-GCT. We first attach instruction lists to the nodes. The root represents the state of M before the computation starts. Therefore its instruction list is empty. That of the son of the root consists of the instructions to be executed first by the processors. Now suppose this attachment is done up to depth s. Let v be a node in depth s and v' a son of v. If v is no Boolean node (i.e. it does not represent a Boolean step of M), then $I_i^{v'}$ is the (uniquely determined) instruction to be executed by P_i after I_i^v, i=1,...,p. If v is a Boolean node, let v_d, $d = (d_1,...,d_p) \varepsilon \{>, \leq\}^p$, be the 2^p sons of v. If now I_i^v means "If $\#a_i > 0$,then execute I, else execute J", then $I_i^{v_d}$=I, if d_i=1, else $I_i^{v_d}$=J.

Now we attach the storage lists and the computation lists to the nodes and the restriction sets to the edges. Thereby we prove the properties demanded in claim 1 inductively on the depth s.

The root gets the storage list (i,x_i), i=1,...,n), and its computation list consists of 0-functions. The restriction set of its outgoing edge is N^n. Hereby, obviously the configuration before the start of a computation of M is described correctly, and the properties from claim 1 hold for s=0.

Now let s>0 and suppose that all nodes up to depth s-1 and all outgoing edges are labelled and fulfill the properties demanded in the claim. Let v be a node in depth s-1.

If v is a read node, then there are at most s_v+1 many different values to be read by some processor P_i when M started with $\bar{x}$ has executed the computation up to v, namely $R_j^v(\bar{x})$, if $B_i^v(\bar{x}) = L_j^v(\bar{x})$, $j = 1,...,s_v$, or 0, if none of these equations holds. Now consider some tuple $D=(d_{ij},\ i=1,..,p;\ j=1,..,s_v) \varepsilon \{=, \neq\}^{ps_v}$. For each such tuple D choose a son v_D of v. (Note that $s_v \leq n + p(s-1)$, because in the beginning n registers are occupied and in each step at most p further registers can be accessed. Thus v has a son for each tuple D.) Let e be the edge from v to v_D and e' the incoming edge of v. Then $C_e = \{\bar{x} \varepsilon N^n \mid B_i^v(\bar{x}) - L_j^v(\bar{x})\ d_{ij}\ 0, i = 1,..,p, j = 1,...,s_v\} \cap C_{e'}$. The storage list of v_D is that of v, and the computation list is defined according to the description above about the different values to be read. This attachment clearly ensures that the simulation goes on correctly. Furthermore, all functions appearing in the storage or computation list of v_D belong to those of v, thus to $F_{p2^{s-1}} \subset F_{p2^s}$, as demanded in the claim. As differences of two functions from F_q belong to F_{2q}, C_e is defined by functions from F_{p2^s}, which proves claim 1 for this case.

If v is an arithmetic node, and v' is its son, then the storage list of v' is that of v, the restriction set of the edge from v to v' is that of the incoming edge of v, and $A_i^{v'}$ or $B_i^{v'}$ equal $A_i^v \mp B_i^v$ or some constant $\leq i$ dependent on the arithmetic operation to be executed by P_i. But anyway we simulate correctly, and obtain functions from F_{p2^s}, namely constants $\leq p$ or sums or differences of functions from $F_{p2^{s-1}}$. Thus the claim also holds in this case.

If v is a write node, we consider the same set of sons v_D and attach the same restriction sets to the edges as in the above description for read nodes. The computation lists of these sons are those of v, and their storage lists are derived from those of v by the following modifications. Let $D \varepsilon \{=, \neq\}^{n+p(s-1)}$ and $j \varepsilon \{1,...,s_v\}$ be fixed. If there is $i' \varepsilon \{1,...,p\}$ such that $d_{i'j}$ means "=", then let i be minimal with this property (compare our rule of solving write conflicts), and replace R_j^v by A_i^v. If such an i' does not exist, then add a new pair (L,R) to the list with $L \equiv B_i^v$, $R \equiv A_i^v$. It is immediate that this attachment fulfills the properties demanded in the claim.

If v is a Boolean node, then each of the 2^p sons of v represents a possible outcome of the p if-questions asked by the p processors. Their storage and computation lists are those of v. Let v_d be a son of v representing the outcomes $d=(d_1,...,d_p)\varepsilon\{>,\leq\}^p$ of the if-questions. Let e be the edge from v to v_d. Then $C_e = C_{e'} \cap \bigcap_{i=1}^{p} \{\bar{x}\varepsilon N^n \mid A_i^v(\bar{x})\ d_i\ 0\}$. This attachment also fulfills all demanded properties, and therefore claim 1 is proved. q.e.d.

SECTION 4

In this section we prove lemma 2. This proof is based on theorem 2 from [7].

THEOREM *([7]) : Let $H_1,...,H_m$ be hyperplanes in R^n, $H_i=\{\bar{x}\varepsilon R^n \mid f_i(\bar{x})=0\}$ for some $f_i\varepsilon F_q$, and $L=\bigcup_{i=1}^{m} H_i$. Then $[-1,1]^n \cap L$ can be recognized by an LSA in $3n^4(\log(n)+\log(q))+O(n^3)$ steps.*

In order to prove lemma 2 we first show

CLAIM 2 *: The above theorem also holds, if L instead of $[-1,1]^n \cap L$ has to be recognized.*

Proof : For an affine function $f:R^n \to R$ with $f(\bar{x})=\bar{a}\bar{x}-b$ for some $\bar{a}\varepsilon R^n, b\varepsilon R$, let $\tilde{f}:R^{n+1}\to R$ be defined by $\tilde{f}(\bar{x},x_{n+1})=\bar{a}\bar{x}-bx_{n+1}$. Now let $f_1,...,f_m, H_1,...,H_m$ be as in the above theorem from [7], $\tilde{H}_1,...,\tilde{H}_m$ be the linear hyperplanes in R^{n+1} defined by $\tilde{f}_1,...,\tilde{f}_m$, and $\tilde{L}=\bigcup_{i=1}^{m}\tilde{H}_i$. It suffices to prove the theorem for $\tilde{L}$, because an LSA recognizing $\tilde{L}$ recognizes L if we substitute 1 for x_{n+1}. $\tilde{L}$ consists of linear hyperplanes, i.e. all $\tilde{H}_i$'s contain $\bar{0}$.

Now let $P_j^+ (P_j^-)=\{\bar{x}\varepsilon R^{n+1} \mid x_j > (<) 0$ and $|x_j| = \max\{|x_1|,...,|x_{n+1}|\}\}$ and $\tilde{L}_j^{\pm} = \tilde{L} \cap P_j^{\pm}$. Then by lemma 1 from [7], $\tilde{L}_j^{\pm}$ can be recognized as fast as $\tilde{L}_j^{\pm} \cap \{\bar{x}\varepsilon R^n \mid x_j = \pm 1\} = L_j^{\pm} \cap [-1,1]^n$, where $L_j^{\pm}$ is the union of the hyperplanes in $R^n (\cong R^{n+1}|_{x_j=\pm 1})$ defined by $f_{ij}^{\pm} = \tilde{f}_i|_{x_j=\pm 1}$. As f_i belongs to F_q, $f_{ij}^{\pm}$ does, too. Thus $L_j^{\pm}$, and therefore $\tilde{L}_j^{\pm}$, too, can be recognized in $3n^4(\log(n)+\log(q))+O(n^3)$ steps because of the theorem from [7]. Therefore the following LSA recognizes L as fast as desired in claim 2.

Decide in which $P_j^{\pm}$ the input $\bar{x}$ lies. Suppose $\bar{x}\varepsilon P_j^{\pm}$.
Use the above LSA for recognizing $L \cap P_j^{\pm}$.

The first part of this algorithm needs $2(n+1)$ steps, the second part needs $3n^4(\log(n)+\log(q))+O(n^3)$ steps as shown above. q.e.d.

Claim 2 already proves lemma 2 for special q-languages, namely those which consist of hyperplanes defined by functions from F_q. We now shall construct LSA's for arbitrary q-languages from the above LSA's for q-languages consisting of hyperplanes. For this purpose let $H_1,...,H_m$ be arbitrary hyperplanes in R^n and $L=\bigcup_{i=1}^{m} H_i$. We say, an LSA partitions R^n according to L, if for each leaf v of the LSA, the set of inputs arriving at v is a subset of a face of L (compare section 2).

CLAIM 3 *: If L can be recognized by an LSA in t steps, then R^n can be partitioned according to L in 2t steps.*

The claims 2 and 3 imply lemma 2 as follows. Let L be a q- language and L_q the language consisting of all hyperplanes defined by functions

from F_q. Then, by claim 2, L_q can be recognized by an LSA in t= $3n^4(\log(n) + \log(q)) + O(n^3)$ steps. Thus, by claim 3, R^n can be partitioned according to L_q in 2t steps by some LSA T. By the definition of a q-language, L is the union of faces of L_q. Therefore we obtain an LSA of depth 2t for L by attaching "accept" to all leaves v of T for which the set of inputs arriving at v is a subset of a face of L_q belonging to L, and attaching "reject" to the other leaves of T. Thus it remains to prove claim 3.

Proof of claim 3 : A 3-way LSA T is a 3-ary tree whose nodes are labelled with affine functions $f:R^n \to R$. An input $\bar{x} \epsilon R^n$ arriving at such a node chooses the left (middle, right) branch, if $f(\bar{x})>(=,<)0$. The leaves are labelled with "accept"or "reject". The set of inputs arriving at a node v is called c(v). T accepts the union of all sets c(v) for accepting leaves v of T.

SUBCLAIM : *Let T be a 3-way LSA accepting L in t steps. Then there is a 3-way LSA T' with depth t and the following property : For each leaf v of T' with c(v)⊂L there is a leaf v' of T' for which c(v')∩L is empty, such that c(v) is a face of c(v').*

Proof : Let T be as above. We may assume w.l.o.g. that v is a leaf whenever c(v)⊂L. Then an accepting leaf is always reached via the middle branch of its father. The following algorithm constructs T' from T.

1)Mark all fathers of accepting leafs.
2)While there is a marked node v whose right son v" is no leaf, replace its middle son by a copy of the subtree with root v".
3)Remove all labels (accept or reject) from the leaves.

This algorithm stops, because it neither changes the maximum degree nor the depth of the tree but adds at least one node to it in each step of 2).

Now let v be a leaf of T' with c(v)⊂L. Traverse the path from the root to v until the first time it chooses the middle branch of a marked node. At this point choose the right branch instead, and go on in the copy of the path to v in the subtree of this branch in the same way. Let this path lead to the leaf v'. Then c(v) and c(v') differ exactly by the fact that restrictions from marked nodes of the form $f(\bar{x}) = 0$ defining c(v) are replaced by $f(\bar{x}) < 0$ in the definition of c(v'). But this just means that c(v) is a face of c(v'). Furthermore c(v')∩L is empty , because inputs from L choose a path in T' on which at least at one marked node the middle branch is chosen. By construction this is not the case in the path to v'. q.e.d.

In order to prove claim 3 we now show that the above 3-way LSA partitions R^n according to L. As obviously T' can be simulated by an LSA in 2t steps, this implies claim 3.

Let v be a leaf of T'. If c(v)∩L is empty, then c(v) belongs to a connected component of $R^n - L$, which is by definition a face of L. If c(v)⊂L, then by the subclaim c(v) is a face of c(v') for some leaf v' for which c(v')∩L is empty. As each face of a polytope contained in one of the polytopes P, $R^n - L$ consists of, is a subset of a face of P, claim 3 follows. q.e.d.

SECTION 5

We have shown in this paper that the n-dimensional restriction of many languages can be recognized surprisingly fast. This result motivates the following questions.

1) The set F_q consists of $m_q = O(q^n)$ functions, i.e. L_q, the union of all the hyperplanes defined by functions from F_q, consists of m_q hyperplanes. Our result shows that L_q can be recognized in poly(m_q) steps. In order to understand the power of LSA's it is interesting to find out whether LSA's can recognize every union of m

hyperplanes in R^n in *poly*(log(*m*),*n*) steps, or whether there exist 'hard' such languages.

2) By Ben Or's result from [11], most known lower bounds for LSA's also hold for algebraic computation trees (ACT's) in which multiplication and division are allowed. Also ACT's only deal with n-dimensional restrictions of problems. Does this also imply surprisingly fast ACT's? A weaker result than that in this paper can be shown which says shows that at least deterministic and probabilistic ACT's are polynomially related.

3) The reason why LSA's are so fast is the fact that the length of LSA's may depend on n, the number of input variables, whereas the length of 'usual' programs is bounded independent of the input. This means for a 'bounded program length' version of LSA's that both the tree and the set of functions attached to its nodes have a lot of structure. It would be of greatest interest to explore this structure and to derive lower bound arguments from it.

REFERENCES

[1] *A. V. Aho, J. E. Hopcroft, J. D. Ullman :* The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading Mass., 1974.

[2] *W. J. Paul, J. Simon :* Decision Trees and Random Access Machines, Symposium ueber Logik und Algorithmik, Zuerich, 1980.

[3] *P. Klein, F. Meyer auf der Heide :* A Lower Time Bound For the Knapsack Problem on Random Access Machines, Acta Informatica 19, 385-395, 1983.

[4] *F. Meyer auf der Heide :* Lower Bounds for Solving Linear Diophantine Equations on Random Access Machines, Interner Bericht 6.84, Fb Informatik, Universitaet Frankfurt, 1984.

[5] *E. Reingold :* On the Optimality of some Set Algorithms, J. ACM 19, 649-659, 1972.

[6] *D. Dobkin, R. J. Lipton :* A Lower Bound of $\frac{1}{2}n^2$ on Linear Search Programs for the Knapsack Problem, J.C.S.S. 16, 413-416, 1978.

[7] *F. Meyer auf der Heide :* A Polynomial Linear Search Algorithm for the n-Dimensional Knapsack Problem, J. ACM 31(3), 668-676, 1984.

[8] *M. R. Garey, D. S. Johnson :* Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[9] *C. H. Papadimitriou :* On the Complexity of Unique Solutions, J. ACM 31(2), 392-400, 1984.

[10] *F. Meyer auf der Heide :* Lower Time Bounds for Testing the Solvability of Diophantine Equations on Several Parallel Computational Models, Interner Bericht 7.84, Fb Informatik, Universitaet Frankfurt, 1984.

[11] *M. Ben Or :* Lower Bounds for Algebraic Computation Trees, 15th ACM Symposium on Theory of Computing, Boston, 80-86, 1983.