Barcan Marcus et al., eds., Logic, Methodology and Philosophy of Science VII
© Elsevier Science Publishers B.V. (1986) 141–158

ARE RECURSION THEORETIC ARGUMENTS USEFUL IN COMPLEXITY THEORY?

WOLFGANG MAASS*

Dept. of Mathematics and Computer Science Division,
Univ. of California, Berkeley, CA 94720, U.S.A.

1. Introduction

Recursion theory is that area of mathematical logic where one studies the *qualitative* aspects of computability. Here one is only interested in the question whether a computation converges at all, i.e. yields a result after finitely many computation steps. In complexity theory, which is part of computer science, one studies in addition *quantitative* aspects of computations. For example one studies for computations on a mathematical computer model the computation time as a function of the size of the input.

Over the last few decades a number of quite powerful techniques have been developed in recursion theory — most of them so-called priority arguments — that finally allowed to solve a number of difficult open recursion theoretic problems (see SOARE [25]). In complexity theory, on the other hand, a variety of concepts and methods have been introduced but many basic and important problems remain open. We analyze and survey in this paper some of our recent research in the light of the question whether arguments from recursion theory are useful in complexity theory. We arrive at the conclusion that recursion theoretic techniques are in fact useful in complexity theory, although in general only in combination with arguments about algorithms for concrete problems or with arguments about concrete computer models.

Many problems in complexity theory deal with the question whether certain mathematical problems can be solved by computations whose computation time is polynomially related to the size of the input. It is

* During the preparation of this paper the author has been supported by the Heisenberg Programm of the Deutsche Forschungsgemeinschaft, Bonn.

Permanent address (after Fall 84): Dept. of Mathematics, Statistics and Computer Science, University of Illinois at Chicago.

tempting to view such quantitative questions as qualitative questions in a new generalized recursion theory where one interprets the basic concept of "finite" as "of polynomial size in the considered parameters" and "recursive function" as "in polynomial time computable function." It is well known that many arguments from recursion theory can be transferred to generalized recursion theories, where the basic notions of "finite" and "recursive function" are substituted by other notions (see e.g. FENSTAD [5]). We look in Section 2 of this paper at a number of open problems about the structure of NP where one can *prove* that even under the assumption $P \neq NP$ recursion theoretic arguments will not suffice. Ironically our proof uses a recursion theoretic argument.

In Sections 3 and 4, on the other hand, we exhibit examples from complexity theory where a strategy that is very reminiscent of a well-known strategy from priority arguments in recursion theory is used in combination with concrete arguments about algorithms (Section 3) resp. computer models (Section 4). In Section 3 we construct polynomial time approximation schemes for some strongly NP-complete problems that arise e.g. in robotics. In Section 4 we survey a proof of optimal lower bounds for two tapes versus one on deterministic and nondeterministic Turing machines. We further get results that show a substantial superiority of nondeterminism over determinism resp. co-nondeterminism over nondeterminism for one-tape Turing machines (which have an additional one-way input tape). We show that both in Section 3 and in Section 4 one can view the proof of the desired result as the construction of a winning strategy for a two-person game. Further the winning strategy that we give employs a tactic that is familiar from modern priority arguments. Our winning strategy consists of a system of different strategies which have the property that the failure of one strategy (which after all tells us a little bit about the opponent) increases the chances of the other strategies to beat the opponent. Such tactic is actually used quite often in complexity theory, although it usually remains hidden in the combinatorics. We believe that it is worthwhile to make this feature more explicit because its full power has not yet been exploited. It is quite plausible that the proofs of many theorems in complexity theory have not yet been found for the same reasons that delayed the solution of several problems in recursion theory. One tends to insist on winning strategies that try to reach their goal too uniformly, i.e. besides the *outcome* of the game they also want to prescribe *how* the game is won (which is unnecessary and often impossible). The previously sketched tactic leaves it open *which* strategy in our system will overcome the opponent. Thus it offers a way to exploit the power of inconstructive

mathematics. It further appears that similarly as in recursion theory the description of lower bound proofs as games makes it possible to keep track of increasingly complex situations (with nested strategies, etc.).

There are many interesting interactions between recursion theory and complexity theory that we do not even touch in this paper. We refer to SOARE [24] for a recent survey concerning the qualitative theory of complexity measures (it turns out that in this area one also finds applications of concepts from complexity theory to recursion theory, see also MAASS [17]). Additional results and references can be found in HARTMANIS and HOPCROFT [9] and JOSEPH [16].

The previously indicated possibility to view polynomial time computable functions as the "recursive" functions of a generalized recursion theory is made explicit in forthcoming work by Moschovakis.

We do not assume in this paper any knowledge from complexity theory. In particular we try to define and illustrate all concepts from complexity theory that we use.

2. On the limits of recursion theoretic arguments in complexity theory

We assume that the reader is familiar with the standard definition of a Turing machine (abbreviated: TM). A set of binary strings is in the class P if its characteristic function can be computed by a deterministic TM in time $p(n)$ for some polynomial p (n is the length of the input for the considered computation). The only new feature of a nondeterministic TM N is that its transition function is multiple-valued. Thus for every input w one has instead of one computation a tree of many different computations of TM N on this input. One says that N accepts input w if one of the branches in the tree ends with an accepting final state (assume that all final states of N have been partitioned into accepting and nonaccepting states). N accepts w in time t if there is at least one such branch of length $\leqslant t$ (or one can demand that every accepting branch has length $\leqslant t$ — it does not make a difference in the following). Finally one says that a set of binary strings is in the class NP if there is a nondeterministic TM that accepts exactly the strings in this set and further accepts each string of length n in time $p(n)$ for some polynomial p. Notice that for sets in NP there is an asymmetry between being *in* the set and being *out* of the set, similarly as for recursively enumerable sets.

It is tempting to view the classes P and NP as downward projections of the classes of recursive and recursively enumerable sets. Note that one may

view the elements of a recursively enumerable set $f[\mathbf{N}]$, where f is some total recursive function, as those elements w that are accepted by a nondeterministic TM that tries in each computation branch a different argument x and halts at the end of the branch in an accepting state iff $f(x) = w$.

Unfortunately so far one cannot answer even the most basic questions about this downward projected recursion theory (e.g. P = NP?). BAKER et al. [2] have shown that the situation is even worse. They consider relativizations P^O and NP^O of P and NP where the attached "oracle" O is some set of binary strings. One can use e.g. oracle-TM's like in recursion theory to define such relativized complexity classes. An oracle-TM may ask its attached oracle O at any time and as often as it likes during the computation whether the string u that it has currently written on its special oracle-tape is in the set O or not. The oracle O gives in one step the correct answer. General experience says that every recursion theoretic argument "relativizes", i.e. remains valid if one attaches the same oracle O everywhere in the argument (for an arbitrarily chosen set O). This relativized argument proves then an accordingly relativized theorem. BAKER et al. [2] show that it is impossible to prove P = NP or $\mathrm{P} \neq \mathrm{NP}$ by an argument that relativizes. They do this by constructing via simple diagonalization sets A and B s.t. $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$.

This result leaves the possibility open that one can get under the assumption $\mathrm{P} \neq \mathrm{NP}$ via recursion theoretic arguments a clear picture of the structure of the classes P and NP (following the standard tradition in logic to take as an axiom what one cannot prove). The following result shows that there are also limitations to this program.

THEOREM 2.1 (HOMER and MAASS [14]). *The following statements S are "independent" from the assumption $\mathrm{P} \neq \mathrm{NP}$ in the sense that there are recursive sets A and B s.t. $\mathrm{P}^A \neq \mathrm{NP}^A$ and S^A but $\mathrm{P}^B \neq \mathrm{NP}^B$ and $\neg S^B$:*

(1) *every infinite set in* NP *has an infinite subset that is in* P,

(2) *there are simple elements in the lattice of* NP*-sets (with set theoretic inclusion),*

(3) *there is a set U in* NP *that is universal for* P, *i.e.* $\mathrm{P} = \{\{v \mid \langle v, w\rangle \in U\} \mid w \text{ a binary string}\}$ *for some standard pairing operation* $\langle \cdot, \cdot \rangle$.

To prove Theorem 2.1 one splits for each statement S the desired properties of A resp. B into infinitely many requirements. One constructs A and B in stages s.t. gradually all requirements become satisfied. This

construction is somewhat delicate because there arise conflicts between requirements of different types. One possible way to solve such conflicts is to use a finite injury priority construction. Alternatively — since one has in these constructions a recursive a priori bound on the stages where an earlier attempt might be injured — one can eliminate with some additional work all injuries. On the other hand, one encounters usually still delays of the activities for a given requirement and in order to show that each requirement is only finitely long delayed one has to argue like in a finite injury priority argument. In general it may be appropriate to view a delay of a requirement in the restricted world of constructions of recursive sets (instead of recursively enumerable sets) as a form of injury.

Following Theorem 2.1 a large number of similar "independence" results has been found (see references in JOSEPH [16]).

What methods remain that might possibly answer the mentioned questions from complexity theory if recursion theoretic arguments (actually more generally: arguments from mathematical logic) do not suffice? We would like to mention two possible escapes. If one proves (by any argument) that a *concrete* NP-complete problem (say HAMILTONICITY) is not in P then this proof of $P \neq NP$ does not relativize. There is not even a natural definition of HAMILTONICITY^O for an oracle O. Second one might analyze more closely the *concrete* structure of computations on a specific computation model. In general such arguments do not remain valid if one adds an oracle tape to the computation model. Thus in any case it appears to be unavoidable that the recursion theorist gets "his fingers dirty".

3. Approximation algorithms

In this section we apply a strategy that is familiar from recursion theory in order to design approximation algorithms.

The following computational problem arises in the context of motion planning and positioning of robots:

Given: n points in Euclidean space (e.g. spots that have to be welded by a robot) and some type of industrial robot.

Wanted: a minimal number k of positions for the base of the robot s.t. each of the n points can be reached by the arm of the robot from one of these k positions.

We look first at the 1- and 2-dimensional versions of this problem. Assume that all given points lie in a fixed horizontal plane. Assume that

from any fixed base position the arm of the robot can reach any point that has a distance between r and $r+w$ from the (vertical axis through the) base of the robot, where r and w depend on the flexibility of the arm of the considered type of robot. Thus we arrive at the mathematical problem of covering n given points in the Euclidean plane by a minimal number of rings with inner radius r and outer radius $r+w$. Unfortunately the following result suggests that no computer is able to solve this problem (for nontrivial sizes of n).

THEOREM 3.1 (FOWLER, PATERSON and TAMIMOTO [6]). *The problem whether n given points in the Euclidean plane can be covered by k rings of inner radius r and outer radius $r+w$ is strongly* NP*-complete (even if we fix $r=0$, i.e. consider only discs).*

We would like to explain briefly to those readers that are not familiar with nondeterminiinistic computations what this means. It is easy to see that the considered problem (which we identify with the *set* $\{\langle\langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle, r, w, k\rangle \mid$ the n points with coordinates $\langle x_i, y_i\rangle$ can be covered by k rings of inner radius r and outer radius $r+w$; all numbers are rational$\}$) lies in the class NP. A nondeterministic Turing machine (see definition in Section 2) just guesses the positions of up to k rings and checks whether all points are covered by these rings. If a tuple $\langle\langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle, r, w, k\rangle$ is in the considered set then along some branch of the computation tree of the nondeterministic computation the TM guesses k ring positions that cover all points and therefore it halts at the end of this branch in an accepting state. Since it takes only polynomially many steps (in the length of the considered tuple) to write down k guessed ring positions and to verify that all n points are covered (assume that one can compute in one step the distance between two points), this accepting branch is of polynomial length in the size of the input. Thus the problem is in NP. To say that the problem is NP-complete means that it is in NP and that every other problem in NP can be reduced to it by a deterministic polynomial time computable function (like in many-one reducibility). This implies that the problem is not in P unless P = NP. *Strong* NP-completeness means that this holds even if we write down the data of the problem in unary code (which makes the size of the input much longer).

Notice that if we cannot compute in polynomial time the minimal number k of rings that are needed for a covering, we certainly cannot compute an optimal covering in polynomial time.

We refer to GAREY and JOHNSON [7] for further information about NP-completeness.

Usually one can escape NP-completeness in geometric location problems by looking only at special cases that are essentially 1-dimensional. In our case one might want to consider given points on a straight line (or on a fixed number of parallel lines). Notice that the intersection of a ring with a straight line is a pair of closed intervals. Unfortunately our problem is quite obnoxious.

THEOREM 3.2 (MAASS [18]). *The problem whether n given points on the line can be covered by k pairs of closed intervals of length w and distance 2r is strongly* NP*-complete.*

Does NP-completeness imply that it is hopeless to attack these problems on a computer? No, because even NP-complete problems may have good approximation algorithms (another possibility would be to look at randomized algorithms, a third possibility would be to show that P = NP). Instead of a minimal number of robot positions an approximation algorithm for the considered problem computes a nearly minimal number of robot positions from which all points can be reached. If for an instance I of our problem $\mathrm{OPT}(I)$ is an optimal solution and $\mathrm{A}(I)$ is an approximate solution that is produced by approximation algorithm A one calls

$$\frac{\big||\mathrm{A}(I)|-|\mathrm{OPT}(I)|\big|}{|\mathrm{OPT}(I)|}$$

the error of A on instance I ($|\mathrm{OPT}(I)|$, $|\mathrm{A}(I)|$ are the numbers of robot positions that are used in the respective solutions). One calls A a *polynomial time approximation scheme* for some combinatorial optimization problem Π if the scheme A provides for every given $\varepsilon > 0$ a polynomial time approximation algorithm A_ε that has error $\leq \varepsilon$ for all instances I of Π.

Not all NP-complete problems have good approximation algorithms. In particular polynomial time approximation schemes for strongly NP-complete problems are very rare (see [7]).

We sketch in the rest of this section the construction of polynomial time approximation schemes for the considered strongly NP-complete problems. We will also point out how one can view these constructions as the construction of winning strategies in certain 2-person games. Our winning strategy employs a system of complementary strategies with the properties that we described in Section 1.

THEOREM 3.3 (HOCHBAUM and MAASS [13]). *For every finite dimension d the problem of computing for n given points in d-dimensional Euclidean space positions for a minimal number of d-dimensional balls with radius w that cover all n points has a polynomial time approximation scheme (this problem is strongly* NP*-complete for* $d \geq 2$).

PROOF. It is sufficient to illustrate the idea for $d = 2$. For a given $\varepsilon > 0$ we describe a polynomial time approximation algorithm A_ε. Fix a natural number l s.t. $(1 + 1/l)^2 \leq 1 + \varepsilon$. Cut the 2-dimensional Euclidean plane into vertical strips of width $l \cdot 2w$. We use the divide-and-conquer method and solve the covering problem separately in each strip. We then take the union of all discs that we use for the coverings in the strips and get a covering of all n given points. The problem with this approach is that it may cause an error $\geq \varepsilon$. This occurs in particular if most of the given points happen to lie close to a cut line.

To view algorithm design as a 2-person game one imagines that player I ("we") wants to produce an algorithm with the desired properties and player II ("opponent") wants to construct an instance of the considered problem where player I's algorithm fails. In the preceding situation player II can win by placing most of the n given points in suitable positions close (i.e. in distance $\leq 2w$) to a cut line of player I's algorithm. Player I is now more clever and uses instead of one strategy S for cutting the plane into strips of width $l \cdot 2w$, l different strategies $S_1, \ldots, S_l$ where $S_1 \equiv S$ and S_{i+1} results from S_i by shifting all cut lines of strategy S_i over a distance $2w$ to the right. The rationale behind this approach is that if player II decides to place e.g. most of the n points close to the cut lines of strategy S_1, he must place accordingly fewer points close to the cut lines of the other strategies S_i. This implies that one of the strategies S_i causes a relatively small error. More precisely each disc of a fixed optimal global solution is cut by a cut line of at most one strategy S_i. Thus for some S_i the cut lines of S_i cut at most $1/l$ of these discs. Further the number of additional discs (compared with the fixed optimal solution) that the divide-and-conquer strategy S_i uses can be bounded by the number of discs in the fixed optimal solution that are cut by a cut line of this strategy S_i. Therefore some strategy S_i causes an error $\leq 1/l$.

So far we have assumed that each strategy S_i computes an optimal covering in each of its strips of width $l \cdot 2w$. Since we do not know how to do this in polynomial time, we use again for each strip an approximation algorithm. We cut now the considered strip by horizontal lines in distance $l \cdot 2w$ and apply again the divide-and-conquer method. In each resulting

$l \cdot 2w \times l \cdot 2w$ square we can afford to compute an optimal covering by exhaustive search because this takes only polynomially in m many steps (where m is the number of points in that square). We use here the fact that at most $2l^2$ discs are needed for an optimal covering of such square. Further every disc in an optimal covering that covers more than one of the given points has w.l.o.g. at least two of the given points on its periphery and these two points determine its position up to two possibilities.

Of course we may again produce an error $\geqslant \varepsilon$ by this divide-and-conquer method for the considered strip. Therefore we try l different substrategies $T_1, \ldots, T_l$ for cutting this strip into $l \cdot 2w \times l \cdot 2w$ squares, where T_{i+1} results from T_i by moving all cut lines of T_i upward over distance $2w$. Some T_i is guaranteed to cause in this strip an error $\leqslant 1/l$ (same argument as before).

Altogether approximation algorithm A_ε proceeds as follows. It tries successively l strategies $S_1, \ldots, S_l$ for cutting the plane into vertical strips of width $l \cdot 2w$. Separately for each strip that arises in some S_i it tries successively l substrategies $T_1, \ldots, T_l$ for cutting it into $l \cdot 2w \times l \cdot 2w$ squares. For each resulting square it computes an optimal covering by exhaustive search. It returns with the resulting covering from that substrategy T_j which uses the fewest discs. Finally A_ε outputs the covering of the n given points which arises from that strategy S_i which uses the fewest discs.

By the preceding A_ε uses at most $(1 + 1/l)^2 \cdot |\mathrm{OPT}(I)|$ discs. It is easy to verify that the running time of A_ε is polynomial in n and w.

Compared with a "supermind" which *knows* immediately the best cutting strategy the previous algorithm A_ε has to try various guesses at the opponent's strategy. A_ε has to pay for this lack of knowledge with a time penalty: a factor of l^2 in the time bound for A_ε. These delays correspond to the injuries in a finite injury priority construction.

Concerning the problem from Theorem 3.1 one gets in the same way a polynomial time approximation scheme for each fixed bound on the "nonconvexity" measure r/w of the covering rings. For the 1-dimensional problem there is a more subtle approach that allows to eliminate the term r/w from the exponent of the time bounds. This yields the following result.

THEOREM 3.4 (HOCHBAUM and MAASS [12]). *There is a polynomial time approximation scheme for the strongly* NP*-complete problem of Theorem* 3.2.

One can improve the time bounds of the previous approximation

algorithms considerably by using insight into the combinatorial resp. geometrical structure of an optimal local covering (see Sections 5 and 6 in [12]).

Of course one gets in the same way approximation schemes for covering with objects of various other shapes. A nice application is the problem of covering given points with a minimal number of squares, which comes up in image processing [26]. Also the same methods happen to provide polynomial time approximation schemes for NP-complete packing problems where one wants to pack without overlap a maximal number of objects of a given size and shape into a given area (JOHNSON [15] describes how such problems arise in the context of VLSI-design).

4. Lower bounds for Turing machines

The generic question of machine-based complexity theory is the following. Given are two classes T_1 and T_2 of mathematical models for computers, where models of type T_2 appear to be more powerful than those of type T_1. Find the slowest growing function S_{T_1,T_2} s.t. any model of type T_2 whose time bound is $t(n)$ (for some function $t(n)$) can be simulated by a model of type T_1 with time bound $O(S_{T_1,T_2}(t(n)))$ (simulation just means that the same output is produced on the same input). Nonlinear lower bounds for S_{T_1,T_2} tell us that models of type T_2 are in fact more powerful and the precise growth rate of S_{T_1,T_2} provides a quantitative measure for the superiority of models of type T_2 over models of type T_1.

Questions of this form arise quite frequently in computer science, e.g. if one wants to make an intelligent choice between several competing designs for hardware or software. Such questions also arise in more theoretical considerations where one wants to classify the inherent computational difficulty of mathematical problems (which often can be determined only for a special type of computer model, e.g. only for nondeterministic machines).

Unfortunately questions of the considered type have only been solved for very few classes T_1 and T_2. The most prominent open problem is the instance where T_1 is the class of deterministic Turing machines and T_2 is the class of nondeterministic Turing machines ($P \stackrel{?}{=} NP$ problem, S_{T_1,T_2} is nonlinear by [22]). Many other open problems of the considered type are not related to nondeterminism. This suggests that there is not just a single "trick" missing (the one that shows $P \neq NP$). Rather a new mathematical

area has to be developed that provides techniques for sharp lower bound results.

We want to report in this section about some new results in this area, that rely on the method of playing simultaneously several strategies against the opponent in a 2-person game. We will describe primarily those aspects that are relevant to this aspect and refer to MAASS [19] for all missing details.

The first problem that we consider is the instance where T_1 is the class of 1-tape deterministic Turing machines and T_2 is the class of 2-tape deterministic Turing machines. We assume that every Turing machine (TM) possesses besides its work tapes (whose number we indicate) an additional one-way input tape (one-way means that the associated head can move only in one direction). Further one head is associated which each tape. All heads may move simultaneously.

The problem of comparing these two classes T_1 and T_2 is actually quite old. Traditionally only 1-tape TM's have been considered. 2-Tape TM's emerged right at the beginning of machine-based complexity theory because one can write for these machines programs that run substantially faster than all known programs for 1-tape TM's. Unfortunately although similarly fast programs have not been found, one was neither able to prove that they do not exist. The obvious disadvantage of a 1-tape TM is the fact that it needs $\Omega(l \cdot d)$ steps to move on its work tape a string of l symbols over a distance of d cells, while a 2-tape TM can do this in time $O(l + d)$. This observation allows to prove easily quadratic lower bounds for a weak form of 1-tape TM's that do not have an extra input tape (they receive the input on the work tape), see HENNIE [11]. E.g. such machine cannot compute for any $\delta > 0$ in $O(n^{2-\delta})$ steps whether a string $x_1 \cdots x_n y_1 \cdots y_n$ is a "palindrome", i.e. for all $i : y_i = x_{n+1-i}$. The 1-tape TM with an extra one-way input tape — this is the model that is usually studied in the current lower bound literature — is quite a bit more powerful and can e.g. recognize palindromes in linear time. In addition for more difficult problems such 1-tape TM has the option to choose a clever "datastructure" for the representation of the input on its work tape which makes it unnecessary to perform during the computation a lot of time-consuming copying operations. In particular, the machine can use several "tracks" on its single work tape and it may also write immediately each input symbol that it reads from the input tape at a number of different locations on the work tape. In order to get strong lower bound results for the function S_{T_1,T_2} in question one has to show that all these tricks cannot help. On the other hand, there are related situations where the use of clever datastructures

helps very well. For example one can simulate a k-tape TM with time bound $t(n)$ by a 2-tape TM without a severe time loss in time $O(t(n) \cdot \log t(n))$, for any $k > 2$ (HENNIE and STEARNS [10]).

The best known upper bound for the function S_{T_1,T_2} in question is $S_{T_1,T_2}(m) = O(m^2)$ (HARTMANIS and STEARNS [8]). The best known lower bound result shows that not $S_{T_1,T_2}(m) = O(m \cdot \log \log m)$ (DURIS et al. [4]; one should also mention related earlier work by RABIN [23], AANDERAA [1] and PAUL [21]).

THEOREM 4.1. *For no* $\delta > 0$, $S_{T_1,T_2}(m) = O(m^{2-\delta})$.

Another open problem of the considered type (see DURIS et al. [4] for a recent list of open problems, we solve here 1. and 7.) deals with the classes T_1^N of nondeterministic 1-tape TM's and T_2^N of nondeterministic 2-tape TM's. The HARTMANIS and STEARNS simulation [8] provides again the best upper bound $S_{T_1^N,T_2^N}(m) = O(m^2)$ and the best lower bound result shows that not $S_{T_1^N,T_2^N}(m) = O(m \cdot \log \log m)$ (DURIS et al. [4]).

Strong lower bounds for *nondeterministic* 1-tape TM's are a bit more difficult. Notice that these machines accept e.g. some NP-complete problems like 3-COLORABILITY in linear time. Further in terms of the previously discussed possibilities a nondeterministic 1-tape TM has an important additional tool. In order to simulate a 2-tape TM without significant time loss it can choose for each input an "individualized" data-structure on its work tape, which facilitates the particular computation that is performed on this particular input. In addition BOOK et al. [3] have proved that for any $k > 2$ one can simulate a nondeterministic k-tape TM by a nondeterministic 2-tape TM without any increase in computation time. Furthermore for alternating TM's (which iterate nondeterminism) PAUL et al. [20] have shown that for any $k > 1$ one can simulate a k-tape alternating TM by a 1-tape alternating TM without any increase in computation time.

THEOREM 4.2. *For no* $\delta > 0$, $S_{T_1^N,T_2^N}(m) = O(m^{2-\delta})$.

So far we have compared classes that have the same control structure but different storage facilities. We consider now pairs of classes which have the same storage facilities (one work-tape besides the one-way input tape) but different control structures. We write $\mathrm{DTIME}_1(t(n))$ and $\mathrm{NTIME}_1(t(n))$ for the classes of sets that are accepted by deterministic resp. nondeterministic 1-tape TM's (always with an additional one-way input tape). We

write CO-NTIME$_1(t(n))$ for the class of sets whose complement is in NTIME$_1(t(n))$.

THEOREM 4.3. NTIME$_1(n) \not\subseteq \bigcup_{\delta>0}$ DTIME$_1(n^{2-\delta})$.

THEOREM 4.4. CO-NTIME$_1(n) \not\subseteq \bigcup_{\delta>0}$ NTIME$_1(n^{2-\delta})$.

Notice that Theorem 4.4 implies Theorem 4.3. In a somewhat related result PAUL et al. [22] have shown that

$$\text{NTIME}_2(n) \not\subseteq \bigcup_{k>1} \text{DTIME}_k(n \cdot (\log^* n)^{1/4}).$$

Concerning stronger separation results the authors of [22] point out that their method might yield at best an $n \cdot \log n$ lower bound. We use here a different type of argument (analysis of the structure of computations for concrete languages) which seems to have no a priori limitations. We construct a language L_1 that satisfies the following lemmata (which obviously imply Theorems 4.1–4.4).

LEMMA 4.5. *L_1 is accepted by a deterministic 2-tape* TM *in linear (even real) time.*

LEMMA 4.6. *The complement of L_1 is accepted by a nondeterministic 1-tape* TM *in linear (even real) time.*

LEMMA 4.7 (Main Lemma). *There is no $\delta > 0$ s.t. L_1 is accepted by a non-deterministic 1-tape* TM *in time* $O(n^{2-\delta})$.

The language L_1 consists of finite sequences of symbols 0, 1, 2, 3, 4. We interpret these symbols as commands that tell a deterministic 2-tape TM M' to perform certain operations and tests. We assume that initially M' is always in "writing mode". In this mode M' copies the initial segment of its input Y from left to right on both work tapes until it encounters in the input a first symbol $z \notin \{0, 1\}$. M' rejects the input unless $z = 4$. M' changes now into the "testing mode" (it never changes back to the writing mode). M' always interprets the symbol 4 as the command to change the direction of movement for both of its work heads. M' interprets 2(3) as the command to move work head 1(2) one cell in the currently required direction. M' in testing mode interprets a symbol $y \in \{0, 1\}$ as the command to test whether the work head that moved last reads currently the symbol y. We

put a string Y in L_1 iff all these test that M' performs for input Y have a positive outcome. With this definition of L_1 we have proved simultaneously Lemma 4.5. The proof of Lemma 4.6 is also quite obvious.

As an example for words in L_1 we note that a binary string $x_1 \cdots x_n y_1 \cdots y_n$ is a palindrome iff the string $x_1 \cdots x_n 42 y_1 2 y_2 \cdots 2 y_n$ is in L_1. For the lower bound argument we will consider words in L_1 of the following structure. Let $X = x_1 \cdots x_n$ be a binary string and let $L = l_1, l_2, \ldots$ and $R = r_1, r_2, \ldots$ be two sets of subsequences of consecutive bits ("blocks") from X. We assume that the blocks in L and R are listed in the order of their occurrence from right to left in X. Let $l_{i,1} \cdots l_{i,p}$ and $r_{i,1} \cdots r_{i,p}$ be the symbols of block l_i resp. r_i in the order of their occurrence in X from right to left. Let $d_l(i)$ ($d_r(i)$) be the number of bits between blocks l_i and l_{i+1} (r_i and r_{i+1}) in X. Further let $d_l(0)$ ($d_r(0)$) be the number of bits in X to the right of block l_1 (r_1). Then the following string is in L_1:

$$x_1 \cdots x_n \underbrace{42 \cdots 22}_{d_l(0) \text{ times}} l_{1,1} 2 l_{1,2} \cdots 2 l_{1,p} \underbrace{3 \cdots 33}_{d_r(0) \text{ times}} r_{1,1} 3 r_{1,2} \cdots 3 r_{1,p} \underbrace{2 \cdots 22}_{d_l(1) \text{ times}} l_{2,1} 2 l_{2,2} \cdots 2 l_{2,p}$$

$$\underbrace{3 \cdots 33}_{d_r(1) \text{ times}} r_{2,1} 3 r_{2,2} \cdots 3 r_{2,p} \cdots \text{(etc., alternating through all blocks of } L \text{ and } R\text{)}.$$

We view the proof of Lemma 4.7 as a 2-person game where player I ("we") wants to prove the claimed lower bound and player II ("opponent") claims to have a counterexample. The opponent starts the game by choosing a nondeterministic 1-tape TM M and constants $\delta, K > 0$. He claims that M accepts L_1 in time $K \cdot n^{2-\delta}$. Player I continues the game by choosing an input $X^\frown Z$ in L_1 on which he tests M. $X^\frown Z$ is chosen as follows.

We assume that some canonical way of coding TM's $\tilde{M}$ by binary strings has been fixed. We write $|\tilde{M}|$ for the length of the binary string that codes $\tilde{M}$. The first part $X = x_1 \cdots x_n$ of the input is a binary string s.t. $K(X) \geqslant n \gg |M|$. Here the Kolmogorov complexity $K(X)$ is defined as

$$K(X) := \min\{|\tilde{M}| \mid \tilde{M} \text{ is a TM which produces (for the empty input) output } X\}.$$

The notion of Kolmogorov complexity has been introduced into complexity theory by PAUL (see [2]). Its advantage is that if $K(X) \geqslant |X| \geqslant |M|$ we can be sure that TM M has nearly no special knowledge about X (X looks like a random string to M).

We define for the rest of this section $\tilde{n} := n^{1-\delta/3}$. Note that (for large n) $\tilde{n}^2$ is bigger than the time bound for M.

To motivate the choice of the second part Z of the input we first give a result that holds for any Z.

LEMMA 4.8 ("Desert Lemma"). *Assume that C is an accepting computation of* TM M *on input $X^\frown Z$ with no more than $K(10n \cdot \log n)^{2-\delta}$ steps (n is the length of string X). Then for large enough n there is an interval D ("desert") of $\bar{n}$ cells on the work tape of M and there are two sets $\bar{L}$ and $\bar{R}$ s.t. both $\bar{L}$ and $\bar{R}$ contain exactly $\bar{n}/2 - 2n^{1-\delta/2}$ blocks B from X with $|B| = n^{\delta/3}$ for each B and s.t. in computation C the work head of M is always left (right) of D while its input head reads from a block B in X that belongs to $\bar{L}$ ($\bar{R}$).*

The *proof* of Lemma 4.8 requires a lengthy combinatorial argument which we cannot give here. One uses in particular that among any $\bar{n}$ cells on the work tape of M there is one which is visited during at most $\bar{n}$ steps. This may be viewed as playing $\bar{n}$ substrategies against the opponent — one of which is guaranteed to win.

If we put ourselves for a moment in the easier situation of the proof of Theorem 4.1 where the opponent's 1-tape TM M is a deterministic machine, we are after Lemma 4.8 already quite close to the completion of the proof. In this case the first part of the computation C of M on input $X^\frown Z$ until the step t_0 where M's input head moves onto the first symbol of Z does not depend on Z. Therefore we need not specify Z before step t_0. Lemma 4.8 deals only with the part of C before step t_0. Thus we can use the sets $\bar{L}$ and $\bar{R}$ that are provided by Lemma 4.8 for the definition of Z. From $\bar{L}$ and $\bar{R}$ we define Z as in the example right after the definition of L_1, with $p := n^{\delta/3}$, $L := \bar{L}$, $R := \bar{R}$. Then we can complete the proof by using Lemma 4.10 below (call every subsequence of Z an $\bar{L} - \bar{R}$ pair that consists of the commands to check a block from $\bar{L}$ and to check in immediate succession a block from $\bar{R}$).

When we return now to the proof of Lemma 4.7 (the nondeterministic case) we see that our strategic situation is much weaker. In this case the first part of computation C until step t_0 depends already on the second part Z of the input (e.g. M may choose a representation of X on its work tape that facilitates the particular test sequence Z; technically M can guess Z while reading X and verify its guesses later while reading Z). But if we define already Z before the beginning of the computation, with some arbitrarily chosen sets L, R in the way of our previous example, we can hardly expect that the opponent is so kind to arrange C s.t. the sets $\bar{L}$, $\bar{R}$ that come out of Lemma 4.8 are the same — or even similar — to the sets L, R we started with. Therefore we use a system of several different

strategies against the opponent. We use in our first strategy a guess L_1, R_1 at the future $\bar{L}$, $\bar{R}$ that may be totally wrong. But if this is the case we learn at least something about the opponent and the second guess L_2, R_2 that we use in our second strategy is designed to approximate any $\bar{L}$, $\bar{R}$ that are totally different from L_1, R_1. Analogously L_3, R_3 is designed to approximate any $\bar{L}$, $\bar{R}$ that are totally different from L_1, R_1 and L_2, R_2. Altogether we design a system of $\log \tilde{n}$ "guesses" L_i, R_i and we use L_i, R_i to define the ith section Z_i of Z. Z_1 is defined from L_1, R_1 exactly as the string in our previous example had been defined from sets L, R. Z_2 is a similar command sequence that tells the 2-tape TM M' to check in alternation the blocks in L_2 and R_2, the first ones with head 1, the second ones with head 2. This is done on M' during one sweep from left to right of both heads. Z_3 uses like Z_1 a sweep from right to left to check in alternation the blocks in L_3, R_3.

We partition X into $\tilde{n}$ blocks of length $n^{\delta/3}$. We number these blocks in X from left to right by binary sequences of length $\log \tilde{n}$ (assume w.l.o.g. that $\log \tilde{n}$ is a natural number). We say that two blocks are i-connected if their associated binary sequences differ exactly at the i-last bit. If two blocks are i-connected we put the left one into L_i and the right one into R_i.

Finally we define $Z := Z_1 \cap \cdots \cap Z_{\log \tilde{n}}$. Notice that any two blocks from X that are i-connected for some i are tested in immediate succession somewhere in command sequence Z. It is obvious that $X \cap Z \in L_1$.

We have now specified the complete input $X \cap Z$ and Lemma 4.8 provides for this input a "desert" D and two sets $\bar{L}$, $\bar{R}$ of $\tilde{n}/2 - 2n^{1-\delta/2}$ blocks each. We call a subsequence of Z an $\bar{L}$–$\bar{R}$ pair if it consists of the commands to check in immediate succession two blocks b_1, b_2 from X s.t. one belongs to $\bar{L}$ and the other to $\bar{R}$.

LEMMA 4.9. *Assume that the $\tilde{n}$ blocks of X have been partitioned into any three sets $\bar{L}$, $\bar{R}$, G (G consists of those blocks that are neither in $\bar{L}$ nor in $\bar{R}$). Then there are at least* $\min\{|\bar{L}|, |\bar{R}|\} - |G| \log \tilde{n}$ *$\bar{L}$–$\bar{R}$ pairs in the previously defined sequence Z.*

PROOF OF LEMMA 4.9. We verify now that our previously described tactic — where we play a system of $\log \tilde{n}$ strategies against the opponent — is successful. Assume for simplicity that $G = \emptyset$ and $|\bar{L}| = |\bar{R}| = \tilde{n}/2$. We view the partition into $\bar{L}$, $\bar{R}$ as a coloring of the blocks in X. Consider the case where our first strategy fails completely and Z_1 contains no $\bar{L}$–$\bar{R}$ pair. This implies (by the definition of L_1, R_1 respectively the definition of "1-connected") that the first and second block in X have received the same

color, the third and fourth block in X have received the same color, etc. Assume in addition that the second strategy fails completely and the second section Z_2 of Z contains also no $\tilde{L}$–$\tilde{R}$ pair. Together with the previous information this implies that the first through fourth block in X have the same color, the fifth through eighth block in X have the same color, etc. Apparently, this cannot go on for all sections $Z_1, \ldots, Z_{\log \tilde{n}}$ of Z because otherwise all blocks in X would have received the same color, a contradiction to $|\tilde{L}| = |\tilde{R}| = \tilde{n}/2$.

It is not difficult to fill in the precise proof of Lemma 4.9, which proceeds by induction on $\log \tilde{n}$.

Lemma 4.9 implies that for the two sets $\tilde{L}$, $\tilde{R}$ that have been provided by Lemma 4.8 there are at least $\tilde{n}/2 - 6n^{1-\delta/2} \log \tilde{n}$ $\tilde{L}$–$\tilde{R}$ pairs in Z, which is more than $\tilde{n}/4$ for large n. The final knockout is delivered by the following lemma.

LEMMA 4.10. *For at least* 1/3 *of the* $\tilde{L}$–$\tilde{R}$ *pairs in* Z *the work head of* M *crosses the* $\tilde{n}/3$ *cells in the middle of desert* D *during those steps where its input head reads from that* $\tilde{L}$–$\tilde{R}$ *pair in* Z.

The *proof* of Lemma 4.10 requires a lengthy combinatorial argument. The intuition is that M cannot too often check blocks from X (as demanded by Z) without moving its work head close to the area where it had written notes about this block while reading the corresponding part of X. Of course one has to be aware that M may have written down each block at several locations and it may also have spread information about each block to other areas during its later head movements.

Lemma 4.10 implies that the work head of M crosses (for large n) at least $1/3 \cdot \tilde{n}/4$ often the $\tilde{n}/3$ cells in the middle of desert D. This takes at least $\tilde{n}^2/36$ steps, which exceeds for large n the time bound of $K(10n \cdot \log n)^{2-\delta}$ steps for machine M on input $X^\frown Z$. This finishes the proof of Lemma 4.7.

References

[1] AANDERAA, S.O., 1974, *On k-tape versus* $(k-1)$*-tape real time computations*, in: Complexity of Computation, R.M. Karp, ed., SIAM-AMS Proceedings, Vol. 7 (AMS, Providence), pp. 75–96.

[2] BAKER, T., GILL, J. and SOLOVAY, R., 1975, *Relativizations of the* P ≟ NP *question*, SIAM J. Comput 4 (4), pp. 431–442.

[3] BOOK, R.V., GREIBACH, S.A. and WEGBREIT, B., 1970, *Time and tape bounded Turing acceptors and AFL's*, J. Comput. Syst. Sci. 4, pp. 606–621.
[4] DURIS, P., GALIL, Z., PAUL, W. and REISCHUK, R., 1983, *Two nonlinear lower bounds*, Proceedings of the STOC Conference of the ACM, pp. 127–132.
[5] FENSTAD, J.E., 1980, *General Recursion Theory: An Axiomatic Approach* (Springer, Berlin).
[6] FOWLER, R.J., PATERSON, M.S. and TAMIMOTO, S.L., 1981, *Optimal packing and covering in the plane are NP-complete*, Inform. Process. Lett. 12, pp. 133–137.
[7] GAREY, M.R. and JOHNSON, D.S., 1979, *Computers and Intractability* (Freeman, San Francisco).
[8] HARTMANIS, J. and STEARNS, R.E., 1965, *On the computational complexity of algorithms*, Trans. AMS 117, pp. 285–306.
[9] HARTMANIS, J. and HOPCROFT, J., 1971, *An overview of the theory of computational complexity*, J. ACM 18, pp. 444–475.
[10] HENNIE, F.C. and STEARNS, R.E., 1966, *Two-tape simulation of multitape Turing machines*, J. ACM 13, pp. 533–546.
[11] HENNIE, F.C., 1965, *One-tape, off-line Turing machine computations*, Information and Control 8, pp. 553–578.
[12] HOCHBAUM, D.S. and MAASS, W., *Fast approximation algorithms for a nonconvex covering problem*, to appear.
[13] HOCHBAUM, D.S. and MAASS, W., 1985, *Approximation algorithms for covering and packing problems in image processing and VLSI*, J. ACM 32, pp. 130–136.
[14] HOMER, S. and MAASS, W., 1983, *Oracle dependent properties of the lattice of NP-sets*, Theoret. Comput. Sci. 24, pp. 279–289.
[15] JOHNSON, D.S., 1982, *The NP-completeness column: an ongoing guide*, J. Algorithms 3, pp. 182–195.
[16] JOSEPH, D., 1983, *Three proof techniques in complexity theory*, to appear in Proceedings of a Conference on Computational Complexity Theory in Santa Barbara (March 1983).
[17] MAASS, W., 1983, *Characterization of recursively enumerable sets with supersets effectively isomorphic to all recursively enumerable sets*, Trans. AMS 279, pp. 311–336.
[18] MAASS, W., *On the complexity of nonconvex covering*, SIAM J. Comput., to appear.
[19] MAASS, W., 1984, *Quadratic lower bounds for deterministic and nondeterministic one-tape Turing machines*, Proc. STOC Conf. ACM, pp. 401–408.
[20] PAUL, W.J., PRAUSS, E.J. and REISCHUK, R., 1980, *On alternation*, Acta Informatica 14, pp. 243–255.
[21] PAUL, W.J., 1982, *On-line simulation of $k+1$ tapes by k tapes requires nonlinear time*, Proceedings of the 23rd IEEE FOCS Conference, pp. 53–56.
[22] PAUL, W.J., PIPPENGER, N., SZEMEREDI, E. and TROTTER W.T., *On determinism versus nondeterminism and related problems*, Proceedings of the 24th IEEE FOCS Conference.
[23] RABIN, M.O., 1963, *Real time computation*, Israel J. Math. 1, pp. 203–211.
[24] SOARE, R.I., 1981, *Computational complexity and recursively enumerable sets*, to appear in Proceedings of the Workshop on Recursion Theoretic Approaches to Computer Science (Purdue, May).
[25] SOARE, R.I., 1984, *Recursively Enumerable Sets and Degrees: the Study of Computable Functions and Computably Generated Sets* (Springer, Berlin).

Added in proof. Some improvemehts and detailed proofs of the results in Section 4 can be found in: MAASS, W., Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines, Trans. AMS, to appear.