☺ → ☺☺
AG Automaten und Formale Sprachen

Preprint AFL-2011-02
Otto-von-Guericke-Universität Magdeburg, Germany

Deciding according to the shortest computations

Florin Manea[(A)]

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany
manea@iws.cs.uni-magdeburg.de

Abstract

In this paper we propose, and analyze from the computational complexity point of view, a new variant of nondeterministic Turing machines. Such a machine accepts a given input word if and only if one of its shortest possible computations on that word is accepting; on the other hand, the machine rejects the input word when all the shortest computations performed by the machine on that word are rejecting. Our main results are two new characterizations of $\mathbf{P}^{\mathbf{NP}[\log]}$ and $\mathbf{P}^{\mathbf{NP}}$ in terms of the time complexity classes defined for such machines.
Keywords: Computational Complexity, Turing Machine, Oracle Turing Machine, Shortest Computations

1. Introduction

The computation of a nondeterministic Turing machine (and, in fact, any computation of a nondeterministic machine, that consists in a sequence of moves) can be represented as a (potentially infinite) tree. Each node of this tree is an instantaneous description (ID for short, a string encoding the configuration of the machine at a given moment: the content of the tapes and the state), and its children are the IDs encoding the possible configurations in which the machine can be found after a (nondeterministic) move is performed. If the computation is finite then the tree is also finite and each leaf of the tree encodes a final ID: an ID in which the state is either accepting or rejecting. The machine accepts if and only if one of the leaves encodes the accepting state (also in the case of infinite trees), and rejects if the tree is finite and all the leaves encode the rejecting state.

Therefore, in the case of finite computations, one can check if a word is accepted/rejected by a machine by searching in the computation-tree for a leaf that encodes an accepting ID. Theoretically, this is done by a simultaneous traversal of all the possible paths in the tree (as we can deduce, for instance, from the definition of the time complexity of a nondeterministic computation). However, in practice, it is done by traversing each path at a time, until an accepting ID is found, or until the whole tree was traversed. Unfortunately, this may be a very time consuming

[(A)]Also at: Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, RO-010014 Bucharest, Romania (flmanea@fmi.unibuc.ro). The work of Florin Manea is supported by the *Alexander von Humboldt Foundation*.

task. Consequently, one may be interested in methods of using nondeterministic machines in a more efficient manner.

Our paper proposes such a method: the machine accepts (rejects) a word if and only if one of the shortest paths in the computation-tree ends (respectively, all the shortest paths end) with an accepting ID (with rejecting IDs). Intuitively, we traverse the computations-tree on levels and, as soon as we reach a level containing a leaf, we look if there is a leaf encoding an accepting ID on that level, and accept, or if all the leaves on that level are rejecting IDs, and, consequently, reject. We show that the class of languages that are decided according to this strategy by Turing machines, whose shortest computations have a polynomial number of steps, equals the class $\mathbf{P}^{\mathbf{NP}[\log]}$. As a consequence of this result we can also show that the class of languages that are decided by Turing machines, working in nondeterministic polynomial time on any input but deciding according to the computations that have a minimal number of nondeterministic moves, also equals the class $\mathbf{P}^{\mathbf{NP}[\log]}$. These results continue a series of characterizations of $\mathbf{P}^{\mathbf{NP}[\log]}$, started in [10]. Then, we propose another method: the machine accepts (rejects) a word if and only if the the first leaf that we meet in a breadth-first-traversal of the computations-tree encodes an accepting ID (respectively, encodes a rejecting ID); note that in this case, one must define first an order between the sons of a node in the computations-tree. We show that, in the case of ordering the tree lexicographically, the class of languages that are decided, according to this new strategy, by Turing machines whose shortest computations have a polynomial number of steps equals the class $\mathbf{P}^{\mathbf{NP}}$.

The research presented in this paper is related to a series of papers presenting variants of nondeterministic Turing machines, working in polynomial time, that accept (or reject) a word if and only if a specific property is (respectively, is not) verified by the possible computations of the machine on that word. We recall, for instance: polynomial machines that accept if and only if the number of accepting paths is even ($\oplus\mathbf{P}$ from [6]), polynomial machines which accept if at least $1/2$ of their computations are accepting, and reject if at least $1/2$ of their computations are rejecting (the class **PP**), or polynomial machines that accept if at least $2/3$ of the computation paths accept and reject if at most $1/3$ of the computation paths accept (the class of bounded-error probabilistic polynomial time $\mathbf{BPP}_{\mathbf{path}}$); several other examples can be found on the Complexity Zoo web page[1] or [7]. However, instead of looking at all the computations, we look just at the shortest ones, and instead of asking questions regarding the number of accepting/rejecting computations, we just ask existential questions about the shortest computations.

Our work finds motivations also in the area of nature-inspired supercomputing models. Some of these models (see [5, 8], for instance) were shown to be complete by simulating, in a massively parallel manner, all the possible computations of a nondeterministic Turing machine; characterizations of several complexity classes, like $\mathbf{NP}, \mathbf{P}$ and **PSPACE**, were obtained in this framework. However, these machines were, generally, used to accept languages, not to decide them; in the case when a deciding model was considered ([5]), the rejecting condition was just a mimic of the rejecting condition from classical computing models. Modifying such nature-inspired machines in order to decide as soon as a possible accepting/rejecting configuration is obtained, in one of the computations simulated in parallel, seems to be worth analyzing: such a halting condition looks closer to what really happens in nature, and it leads to a reduced consume of resources, comparing to the case when the machine kept on computing until all the

[1] www.complexityzoo.com, a web page constructed and managed by Scott Aaronson

possibilities were explored. Also, from a theoretical point of view, considering such halting conditions could lead to novel characterizations of a series of complexity classes (like the ones discussed in this paper) by means of nature-inspired computational models, as they seem quite close to the idea of deciding with respect to the shortest computations.

2. Basic Definitions

The reader is referred to [2, 4, 7] for the basic definitions regarding Turing machines, oracle Turing machines, complexity classes and complete problems. In the following we present just the intuition behind these concepts, as a more detailed presentation would exceed th purpose of this paper.

A k-tape Turing machine is a construct $M = (Q, V, U, q_o, acc, rej, B, \delta)$, where Q is a finite set of states, q_0 is the initial state, acc and rej are the accepting state, respectively the rejecting state, U is the working alphabet, B is the blank-symbol, V is the input alphabet and $\delta : (Q \setminus \{acc, rej\}) \times U^k \rightarrow 2^{(Q \times (U \setminus \{B\})^k \times \{L,R\}^k)}$ is the transition function (that defines the moves of the machine). An instantaneous description (ID for short) of a Turing machine is a word that encodes the state of the machine and the contents of the tapes (actually, the finite strings of non-blank symbols that exist on each tape), and the position of the tape heads, at a given moment of the computation. An ID is said to be final if the state encoded in it is the accepting or the rejecting state. A computation of a Turing machine on a given word can be described as a sequence of IDs: each ID is transformed into the next one by simulating a move of the machine. If the computation is finite then the associated sequence is also finite and it ends with a final ID; a computation is said to be an accepting (respectively, rejecting) one, if and only if the final ID encodes the accepting state (respectively, rejecting state). All the possible computations of a nondeterministic machine on a given word can be described as a (potentially infinite) tree of IDs: each ID is transformed into its sons by simulating the possible moves of the machine; this tree is called computations-tree.

A word is accepted by a Turing machine if there exists an accepting computation of the machine on that word; it is rejected if all the computations are rejecting. A language is accepted (decided) by a Turing machine if all its words are accepted by the Turing machine, and no other words are accepted by that machine (respectively, all the other words are rejected by that machine). The class of languages accepted by Turing machines is denoted by **RE** (and called the class of recursively enumerable languages), while the class of languages decided by Turing machines is denoted by **REC** (and called the class of recursive languages).

The time complexity (or length) of a finite computation on a given word is the minimum between the number of IDs that occur in an accepting computation of that word and the height of the computations-tree of the machine on the word. A language is said to be decided in polynomial time if there exists a Turing M machine and a polynomial f such that the time complexity of a computation of M on each word of length n is less than $f(n)$, and M accepts exactly the given language. The class of languages decided by deterministic Turing machines in polynomial time is denoted **P** and the class of languages decided by nondeterministic Turing machines in polynomial time is denoted **NP**. If a machine decides a language in polynomial time we usually say that this machine works in polynomial time.

A Turing machine with oracle A, where A is a language over the working alphabet of the

machine, is a regular Turing machine that has a special tape (the oracle tape) and a special state (the query state). The oracle tape is just as any other tape of the machine, but, every time the machine enters the query state, a move of the machine consists in checking if the word found on the oracle tape is in A or not, and returning the answer.

We denote by $\mathbf{P}^{\mathbf{NP}}$ the class of languages decided by deterministic Turing machines, that work in polynomial time, with oracles from **NP**. We denote by $\mathbf{P}^{\mathbf{NP}[\log]}$ the class of languages decided by deterministic Turing machines, that work in polynomial time, with oracles from **NP**, and which can enter the query state at most $\mathcal{O}(\log n)$ times in a computation on a input word of length n.

The following problem is complete for $\mathbf{P}^{\mathbf{NP}}$, with respect to polynomial time reductions (see [9] for a proof):

Problem 1 *(Odd - Traveling Salesman Problem, TSP_{odd}) Let n be a natural number, and d be a function $d : \{1,\ldots,n\} \times \{1,\ldots,n\} \to I\!N$. Decide if the minimum value of the set $I = \{\sum_{i=1}^{n} d(\pi(i),\pi(i+1)) \mid \pi$ is a permutation of $\{1,\ldots,n\}$, and $\pi(n+1) = \pi(1)\}$ is odd.*

We assume that the input of this problem is given as the natural number n, and n^2 numbers representing the values $d(i,j)$, for all i and j. The size of the input is the number of bits needed to represent the values of d times n^2.

Next we describe a $\mathbf{P}^{\mathbf{NP}[\log]}$-complete problem; however, we need a few preliminary notions (see [3] for a detailed presentation). Let n be a natural number and let $C = \{c_1,\ldots,c_n\}$ be a set of n candidates. A preference order on C is an ordered list $\langle c_{\pi(1)} < c_{\pi(2)} < \ldots < c_{\pi(n)}\rangle$, where π is a permutation of $\{1,\ldots,n\}$; if c_i appears before c_j in the list we say that the candidate c_i is preferred to the candidate c_j in this order. Given a multiset V of preference orders on a set of n candidates C (usually V is given as a list of preference orders) we say that the candidate c_i is a Condorcet winner, with respect to the preferences orders of V, if c_i is preferred to each other candidate in strictly more than half of the preference orders. We define the Dodgson score of a candidate c, with respect to V, as the smallest number of exchanges of two adjacent elements in the preference orders from V (switches, for short) needed to make c a Condorcet winner; we denote this score with $Score(C,c,V)$. In [3] it was shown that the following problem is $\mathbf{P}^{\mathbf{NP}[\log]}$-complete, with respect to polynomial time reductions:

Problem 2 *(Dodgson Ranking, $DodRank$) Let n be a natural number, let C be a set of n candidates, and c and d two candidates from C. Let V be a multiset of preference orders on C. Decide if $Score(C,c,V) \leq Score(C,d,V)$.*

We assume that the input of this problem is given as the natural number n, two numbers c and d less or equal to n, and a list of preference orders V, encoded as permutations of the set $\{1,\ldots,n\}$. If we denote by $\#(V)$ the number of preference orders in V, then the size of the input is $\mathcal{O}(\#(V)n\log n)$.

The connection between decision problems and languages is discussed in [4]. When we say that a decision problem is solved by a Turing machine, of certain type, we actually mean that the language corresponding to that decision problem is decided by that machine.

3. Shortest computations

In this section we propose a modification of the way Turing machines decide an input word. Then we propose a series of results on the computational power of these machines and the computational complexity classes defined by them.

Definition 3.1 *Let M be a Turing machine and w be a word over the input alphabet of M. We say that w* is accepted by M with respect to shortest computations *if there exists at least one finite possible computation of M on w, and one of the shortest computations of M on w is accepting; w* is rejected by M w.r.t. shortest computations *if there exists at least one finite computation of M on w, and all the shortest computations of M on w are rejecting. We denote by $L_{sc}(M)$* the language accepted by M w.r.t. shortest computations*, i.e., the set of all words accepted by M, w.r.t. shortest computations. We say that the language $L_{sc}(M)$ is decided by M w.r.t. shortest computations if all the words not accepted by M, w.r.t. shortest computations, are rejected w.r.t. shortest computations.*

The following remark shows that the computational power of the newly defined machines coincides with that of classic Turing machines.

Remark 1 *The class of languages accepted by Turing machines w.r.t. shortest computations equals* **RE***, while the class of languages decided by Turing machines w.r.t. shortest computations equals* **REC***.*

Proof. Since any language from **REC** (respectively, **RE**) is decided (accepted) by a deterministic Turing machine, it is clear that it is also decided (accepted) w.r.t. shortest computations by the same machine. On the other hand, if a language is decided (respectively, accepted) by a Turing machine M w.r.t. shortest computations then that language is decided (accepted) by a classic deterministic Turing machine M' as follows: M' simply generates the computations-tree of M on an input word level by level, and as soon as it generates a level that contains a final ID it accepts the input word if the level contains an accepting ID, and rejects otherwise. □

Next we define a computational complexity measure for the Turing machines that decide w.r.t. shortest computations.

Definition 3.2 *Let M be a Turing machine, and w be a word over the input alphabet of M.* The time complexity of the computation of M on w, measured w.r.t. shortest computations, *is the length of the shortest possible computation of M on w. A language L is said to be decided in polynomial time w.r.t. shortest computations if there exists a Turing M machine and a polynomial f such that the time complexity of a computation of M on each word of length n, measured w.r.t. shortest computations, is less than $f(n)$, and $L_{sc}(M) = L$. We denote by $PTime_{sc}$* the class of languages decided by Turing machines in polynomial time w.r.t. shortest computations.

The main result of this section is the following:

Theorem 3.3 $PTime_{sc} = \mathbf{P}^{\mathbf{NP}[\log]}$.

Proof. The proof will be structured in two parts. First we show the *upper bound* $PTime_{sc} \subseteq \mathbf{P}^{\mathbf{NP}[\log]}$, and, then we show the *lower bound* $PTime_{sc} \supseteq \mathbf{P}^{\mathbf{NP}[\log]}$.

For the first part of the proof let $L \subseteq V^*$ be a language in $PTime_{sc}$ and let M be a Turing machine that decides L in polynomial time w.r.t. shortest computations. Also, let f be a polynomial such that the time complexity of the computation of M on each word of length n, measured w.r.t. shortest computations, is less than $f(n)$. Finally, let # be a symbol not contained in V.

We define the language $L' = \{x\#w\#1^k \mid w \in V^*, x \in \{0,1\}$, and, if $x = 1$ (respectively, $x = 0$) there exists an accepting (respectively, rejecting) computation of M, of length less than k, on the input word $w\}$. It is not hard to see that L' is in **NP**. A nondeterministic machine deciding L' works as follows: it simulates, nondeterministically, a computation of at most k steps of M, and accepts if and only if $x = 1$, or, respectively, $x = 0$, and the simulated computation is accepting, or, respectively, rejecting; otherwise (i.e., if in the k steps simulated by the machine a final configuration was not obtained) it rejects. Clearly, this machine works in polynomial time.

A deterministic Turing machine M', with oracle L', accepting L implements the following strategy, on an input word w:

1. M' searches (by binary search) the minimum length of an accepting computation of M on w, with length less or equal to $f(|w|)$. In this search, the machine queries the oracle L' for $\mathcal{O}(\log_2(f(|w|)))$ times, asking, in each of these queries, if a string of the form $1\#w\#1^k$, with $k \leq f(|w|)$, is in L'.

2. Let n_0 be the minimum length of an accepting computation, with length less or equal to $f(|w|)$, computed in the previous step (we assume that n_0 is set to a special value, $f(n)+1$ for instance, if the search is unsuccessful). The machine verifies now, by another oracle query, if $0\#w\#1^{n_0-1} \in L'$ (i.e., if there exists a shorter rejecting computation of M). If the answer of the last query is positive, M' rejects the input word, otherwise, it accepts.

Since the machine M has at least one possible computation on w of length less than $f(|w|)$, and that $w \in L$ if and only if the shortest computation of M accepts, it is clear that the machine M' decides the language L. Also, M' works in polynomial time and makes at most $\mathcal{O}(\log n)$ queries to the oracle L'; therefore, $L \in \mathbf{P}^{\mathbf{NP}[\log]}$. This completes the proof of the upper bound.

For the second inclusion, note that the class $PTime_{sc}$ is closed to polynomial-time reductions. That is, if $L \in PTime_{sc}$ and L' is polynomial-time reducible to L, then $L' \in PTime_{sc}$. Indeed, assume that g is a function, that can be computed in polynomial time by a deterministic Turing machine such that, $w \in L'$ if and only if $g(w) \in L$. A machine that decides w.r.t. shortest computations the language L' works as follows: first, for the input w, it computes deterministically the function $g(w)$, and, then, runs the machine accepting L on the input $g(w)$; it is clear that this machine implements the desired behavior, and that it works in polynomial time, measured w.r.t. shortest computations. Therefore, it is sufficient to show that the $\mathbf{P}^{\mathbf{NP}[\log]}$-complete problem $DodRan$ can be solved in polynomial time by a Turing machine M that decides w.r.t. shortest computations.

Let us first make several denotations. The input of M consists in the number n, the set C of n candidates, c and d two candidates from C, and V the multiset of preference orders on C (encoded as explained in the previous section). It is not hard to see that one can verify if a candidate is a Condorcet winner for the multiset V of preference orders on C in polynomial time; let f be a polynomial that upper bounds the time needed to do this checking, for every n

and $\#(V)$. Note that one needs at most $(n-1)\left(\left\lfloor\frac{\#(V)}{2}\right\rfloor+1\right)$ switches to make a candidate a Condorcet winner, since, in the worst case, we must bring this candidate from the last position to the first position in $\left\lfloor\frac{\#(V)}{2}\right\rfloor+1$ of the orders. Also, making $(n-1)\left(\left\lfloor\frac{\#(V)}{2}\right\rfloor+1\right)$ switches in the orders of V requires polynomial time. Let g be a polynomial that upper bounds the time needed to make $(n-1)\left(\left\lfloor\frac{\#(V)}{2}\right\rfloor+1\right)$ switches, for every n and $\#(V)$.

This machine implements the following algorithm:

1. M writes, nondeterministically, two numbers k_1 and k_2 (as the strings 1^{k_1} and 1^{k_2}), with $k_i \leq (n-1)\left(\left\lfloor\frac{\#(V)}{2}\right\rfloor+1\right)$ for $i \in \{1,2\}$. Then, M chooses nondeterministically k_1 switches to be made in V, and saves them as the set T_1, and k_2 switches to be made in V, and saves them as the set T_2.

2. M makes (deterministically) the switches from T_1, and saves the newly obtained preference orders as a multiset V_1. M makes (deterministically) the switches from T_2, and saves the newly obtained preference orders as a multiset V_2.

3. M checks (deterministically) if c is a Condorcet winner in V_1. If the answer is positive it goes to step 4, otherwise it makes $2f(n,\#(V))+2g(n,\#(V))$ dummy steps and rejects the input word.

4. M checks (deterministically) if d is a Condorcet winner in V_2. If the answer is positive it goes to step 7, otherwise it makes $2f(n,\#(V))+2g(n,\#(V))$ dummy steps and rejects the input word.

5. If $k_1 \leq k_2$ the machine accepts the input, otherwise it rejects.

First let us see that M works correctly. In step 1 it chooses nondeterministically some switches in V, that are supposed to make c and d Condorcet winners, respectively. Notice that the length of a possible computation performed in this step depends on the choice of the numbers k_1 and k_2; if these numbers are smaller, then the computation is shorter. Then in step 2 the machine actually makes (deterministically) the switches chosen in the previous step. The length of a possible computation, until this moment, is still determined by the choice of k_1 and k_2. In steps 3 and 4 the machine verifies if those switches were indeed good to make c and d winners, according to the orders modified by the previously chosen moves. If they were both transformed in winners by the chosen switches, the computation continues with to step 5; otherwise, the machine makes a sequence of dummy steps, long enough to make that computation irrelevant for the final answer of the machine on the given input. Note that at least one choice of the switches, in step 1, makes both c and d winners. Now, the shortest computations are those ones in which both c and d were transformed into winners and the chosen numbers k_1 and k_2 are minimal. But this is exactly the case when $k_1 = Score(C,c,V)$ and $k_2 = Score(C,d,V)$. In the step 5, all the computations in which c and d were transformed into winners are completed by a deterministic comparison between k_1 and k_2. Thus, after the execution of this step the shortest computations remain the ones where $k_1 = Score(C,c,V)$ and $k_2 = Score(C,d,V)$; the decision of this computation is to accept, if $k_1 \leq k_2$, or to reject, otherwise. Consequently, M accepts if and only if $Score(C,c,V) \leq Score(C,d,V)$, and reject otherwise. Also, it is rather easy to see that M works in polynomial time, since each of the 5 steps described above can be completed in polynomial time.

In conclusion we showed that $DodRan$ can be solved in polynomial time by a Turing machine that decides w.r.t. shortest computations. It follows that $PTime_{sc} \supseteq \mathbf{P}^{\mathbf{NP}[\log]}$, and this ends our proof. □

The technique used in the previous proof to show that $\mathbf{P}^{\mathbf{NP}[\log]}$-complete problems can be solved by in polynomial time by Turing machines that decide w.r.t. shortest computations suggests another characterization of $\mathbf{P}^{\mathbf{NP}[\log]}$. In this respect, consider nondeterministic Turing machines, working in polynomial time, that decide an input according to the decisions of the computations in which the least number of nondeterministic moves is made. Such a machine can be formally defined as follows:

Definition 3.4 *Let M be a Turing machine working in polynomial time and w be a word over the input alphabet of M. We say that w is* accepted by M with respect to the computations with minimum number of nondeterministic moves *if one of the possible computations of M on w, in which M makes the minimum number of nondeterministic moves, is accepting; w is* rejected by M w.r.t. the computations with minimum number of nondeterministic moves *if all the possible computations of M on w, in which M makes the minimum number of nondeterministic moves, are rejecting.We denote by $L_{nm}(M)$* the language decided by M w.r.t. the computations with minimum number of nondeterministic moves *and by $PTime_{nm}$* the class of all the languages decided *in this manner.*

It is not hard to see that, given a Turing machine working in polynomial time and an input word for that machine, the machine will always decide the input word w.r.t. the computations with minimum number of nondeterministic moves, since all of its computations are finite. One can show the following result.

Theorem 3.5 $PTime_{nm} = \mathbf{P}^{\mathbf{NP}[\log]}$.

Proof. We can use a proof similar to the one of Theorem 3.3.

For the inclusion $PTime_{nm} \subseteq \mathbf{P}^{\mathbf{NP}[\log]}$ we can assume, without loss of generality, that the machine accepting a language from $PTime_{nm}$ has all the possible computations on an input of length n of the same length $f(n)$, for some polynomial f (we can complete some of the computations with dummy deterministic steps, in order to make this happen). Then we just have to search (using binary search) for the computation with the minimum number of nondeterministic moves, and check if it is an accepting or rejecting one.

For the inclusion $PTime_{nm} \supseteq \mathbf{P}^{\mathbf{NP}[\log]}$, we use the machine constructed in the proof of $PTime_{sc} \supseteq \mathbf{P}^{\mathbf{NP}[\log]}$, and note that the shortest computations performed by this machine on a certain input are also the computations where the minimum number of nondeterministic moves are made. This concludes our proof. □

4. The first shortest computation

In the previous section we have proposed an decision mechanism of Turing machines that basically consisted in identifying the shortest computations of a machine on an input word, and checking if one of these computations is an accepting one, or not. Now we analyze how the

properties of the model are changed if we order the computations of a machine and the decision is made according to the first shortest computation, in the defined order.

Let $M = (Q, V, U, q_0, acc, rej, B, \delta)$ be a t-tape Turing machine, and assume that $\delta(q, a_1, \ldots, a_t)$ is a totally ordered set, for all $a_i \in U$, $i \in \{1, \ldots, t\}$, and $q \in Q$; we call such a machine an *ordered Turing machine*. Let w be a word over the input alphabet of M. Assume s_1 and s_2 are two (potentially infinite) sequences describing two possible computations of M on w. We say that s_1 is lexicographically smaller than s_2 if s_1 has fewer moves than s_2, or they have the same number of steps (potentially infinite), the first k IDs of the two computations coincide and the transition that transforms the kth ID of s_1 into the $k+1$th ID of s_1 is smaller than the transition that transforms the kth ID of s_2 into the $k+1$th ID of s_2, with respect to the predefined order of the transitions. It is not hard to see that this is a total order on the computations of M on w. Therefore, given a finite set of computations of M on w one can define the lexicographically first computation of the set as that one which is lexicographically smaller than all the others.

Definition 4.1 *Let M be an ordered Turing machine, and w be a word over the input alphabet of M. We say that w* is accepted by M with respect to the lexicographically first computation *if there exists at least one finite possible computation of M on w, and the lexicographically first computation of M on w is accepting; w* is rejected by M w.r.t. the lexicographically first computation *if the lexicographically first computation of M on w is rejecting. We denote by $L_{lex}(M)$* the language accepted by M w.r.t. the lexicographically first computation. *We say that the language $L_{lex}(M)$ is decided by M w.r.t. the lexicographically first computation if all the words not contained in $L_{lex}(M)$ are rejected by M.*

As in the case of Turing machines that decide w.r.t. shortest computations, the class of languages accepted by Turing machines w.r.t. the lexicographically first computation equals **RE**, while the class of languages decided by Turing machines w.r.t. the lexicographically first computation equals **REC**. The time complexity of the computations of Turing machines that decide w.r.t. the lexicographically first computation is defined exactly as in the case of machines that decide w.r.t. shortest computations. We denote by $PTime_{lex}$ *the class of languages decided by Turing machines in polynomial time w.r.t. the lexicographically first computation.* In this context, we are able to show the following theorem.

Theorem 4.2 $PTime_{lex} = \mathbf{P}^{\mathbf{NP}}$.

Proof. In the first part of the proof we show that $PTime_{lex} \subseteq \mathbf{P}^{\mathbf{NP}}$. Let L be a language in $PTime_{lex}$ and let M be a Turing machine that decides L in polynomial time w.r.t. the lexicographically first computation. Also, let f be a polynomial such that the time complexity of the computation of M on each word of length n, measured w.r.t. the lexicographically first computation, is less than $f(n)$.

We define the language $L' = \{x\#w\#w'\#1^k \mid w \in V^*, w'$ is a sequence of consecutive IDs of M, $x \in \{0,1\}$, and, if $x = 1$ (respectively, $x = 0$) there exists an accepting (respectively, rejecting) computation of M on the input word w of length less than k, starting with the sequence of IDs $w'\}$. It is not hard to see that L' is in **NP**. A nondeterministic machine deciding it works as follows: it simulates, nondeterministically, a computation of at most k steps of M, starting with the IDs in the sequence w', and accepts if and only if this $x = 1$,or, respectively, $x = 0$, and the simulated computation is accepting, or, respectively, rejecting; otherwise (i.e., if in the simu-

lated computation steps a final configuration was not obtained) it rejects. Clearly, this machine works in polynomial time.

A deterministic Turing machine M', with oracle L', accepting L implements the following strategy, on an input word w:

1. M' searches (by binary search) the minimum length of a computation of M on w, with length less or equal to $f(|w|)$. In this search, the machine queries the oracle L' for $\mathcal{O}(\log_2(f(|w|)))$ times, asking, in each of these queries, if a string of the form $1\#w\#\epsilon\#1^k$ and $0\#w\#\epsilon\#1^k$, with $k \leq f(|w|)$, is in L'. Let n_0 be the minimum length of a computation, with length less or equal to $f(|w|)$.

2. Next M' tries to construct, ID by ID, the first (shortest) computation of length n_0, using the oracle L'. Assume that w' is a sequence of IDs identified until a given moment as a prefix of the sequence encoding the first computation of length n_0, and we try to prolongate this sequence. Assume that $w_1, w_2, \ldots, w_k$ are the IDs that can be obtained from the last ID of w', ordered according to the transitions that were used to obtain them. We search the minimum i, with $1 \leq 1 \leq k$, such that $0\#w\#w'w_i\#1^{n_0}$ or $1\#w\#w'w_i\#1^{n_0}$ is in L'. Once we have identified this minimum value, denoted i_0, we add the ID w_{i_0} to the sequence w', and repeat the process described above, until w' contains n_0 IDs.

3. The machine finally checks if the string $1\#w\#w'\#1^{n_0}$ is in L', and if it is so accepts, or, if the string $0\#w\#w'\#1^{n_0}$ is in L', and, in this case, rejects.

It is not hard to see that M' correctly computes the length n_0 of the shortest computation of M on an input word w. Also, once this length computed, the first shortest computation is identified, and the machine checks if this computation is an accepting or a rejecting one. Thus, M' implements the desired behavior. Finally, note that M' works in polynomial time: in step 2 it makes $\mathcal{O}(n_0)$ queries, asking if strings of polynomial length are in L', while the rest of the computation is clearly carried out in polynomial time. This completes the proof of the upper bound on $PTime_{lex}$.

To show the second inclusion, note that, similar to the case of machines deciding w.r.t. shortest computations, the class $PTime_{lex}$ is closed to polynomial-time reductions. Thus, it is sufficient to show that the $\mathbf{P}^{\mathbf{NP}}$-complete problem TSP_{odd} can be solved in polynomial time by a Turing machine M that decides w.r.t. the lexicographically first computation.

Therefore we construct a Turing machine M that solves TSP_{odd} w.r.t. the lexicographically first computation. The input of this machine consists in a natural number n, and n^2 natural numbers, encoding the values of the function $d : \{1, \ldots, n\} \times \{1, \ldots, n\} \to I\!N$. We can assume, without losing generality, that all the input numbers are given as decimal numbers; also, we assume that all the n^2 numbers, that encode the values of the function d, have the same number of decimal digits, denoted by m (we may add some leading zeros at the beginning of these numbers in order to make this assumption hold). Therefore the size of the input is $\mathcal{O}(n^2 m)$. Also, let us make the assumption that every time we sum up n numbers of m digits we make exactly $f(m, n)$ steps, where f is a polynomial, and the sum is always represented using the same number of digits (clearly bounded by the input size).

This machine implements the following algorithm:

1. M writes, nondeterministically, a permutation π of $\{1, \ldots, n\}$ and computes, deterministically, the sum $S = \sum_{i=1}^{n} d(\pi(i), \pi(i+1))$. Let k be the number of digits of S.

2. M writes, nondeterministically, a number S_0 of k digits; this number may have some leading zeros. We assume that this step is performed in k computational steps, each consisting in choosing one of the moves $\{m_0, m_1, \ldots, m_9\}$ in which one of the digits $0, \ldots, 9$, respectively, is written. These moves are ordered $m_0 < m_1 < \ldots < m_8 < m_9$.

3. M writes, nondeterministically, a permutation π' of $\{1, \ldots, n\}$ and computes, deterministically, the sum $S' = \sum_{i=1}^{n} d(\pi'(i), \pi'(i+1))$.

4. M checks, deterministically, if $S' = S_0$. If yes it goes to step 5, otherwise it makes $2n^2m$ dummy step and rejects.

5. M checks, deterministically, if S' is odd. If yes it accepts, otherwise it rejects.

It is important to state that the order of the nondeterministic moves that are executed in steps 1 and 3 has no impact on the computation. For uniformity we consider that they are ordered, but we don't make any assumption on what order is actually used.

Before showing that the machine works correctly, we notice that it works in polynomial time. Indeed, it is not hard to see that every possible computation of M consists in a sequence of steps of polynomial length, and always ends with a decision.

To show the soundness of our construction, let us observe that all the possible computations implemented by the first 3 steps of the above algorithm have the same length. In the first of these steps we choose a possible permutation π of $\{1, \ldots, n\}$ and compute the sum $S = \sum_{i=1}^{n} d(\pi(i), \pi(i+1))$; in this way we have computed a possible solution of the Traveling Salesman Problem, defined by the function d, and the real solution of the problem should be at most S. Then we try to find another permutation π' that leads to a smaller sum. For this we choose first a number S_0 that has as many digits as S (of course, it may have several leading zeros); however, the computations are ordered in such a manner that a computation in which smaller numbers are constructed comes before a computation in which a greater number is constructed. Then, in steps 3 and 4, M verifies if S' can be equal to the sum $\sum_{i=1}^{n} d(\pi'(i), \pi'(i+1))$, for a permutation π' nondeterministically chosen. If the answer is yes then it means that S_0 is also a possible solution of the problem; otherwise we conclude that the nondeterministic choices made so far were not really the good ones, so we reject after we make a long-enough sequence of dummy steps, in order not to influence the decision of the machine. Finally, we verify if S_0 is odd, and accept if and only if this condition holds. By the considerations made above, it is clear that in all the shortest computations we identified some numbers that can represent solutions of the Traveling Salesman Problem; moreover, in the first of the shortest computations we have identified the smallest such number, i.e., the real solution of the problem. Consequently, the decision of the machine is to accept or to reject the input according to the parity of the solution identified in the first shortest computation, which is correct.

Summarizing, we showed that TSP_{odd} can be solved in polynomial time by a Turing machine that decides w.r.t. the lexicographically first computation. It follows that $PTime_{lex} \supseteq \mathbf{P}^{\mathbf{NP}}$, and this concludes our proof. □

Remark 2 *Note that the proof of Theorem 3.3 shows that* $\mathbf{P}^{\mathbf{NP}[\log]}$ *can be also characterized as the class of languages that can be decided in polynomial time w.r.t. shortest computations by nondeterministic Turing machines whose shortest computations are either all accepting or all rejecting. On the other hand, in the proof of Theorem 4.2, the machine that we construct to solve w.r.t. the lexicographically first computation the* TSP_{odd} *problem may have both accepting and*

rejecting shortest computations on the same input. This shows that $\mathbf{P}^{\mathbf{NP}[\log]} = \mathbf{P}^{\mathbf{NP}}$ *if and only if all the languages in* $\mathbf{P}^{\mathbf{NP}}$ *can be decided w.r.t. shortest computations by nondeterministic Turing machines whose shortest computations on a given input are either all accepting or all rejecting.*

There is a point where the definition of the ordered Turing machine doesn't seem satisfactory: each time a machine has to execute a nondeterministic move, for a certain state and a tuple of scanned symbols, the order of the possible moves is the same, regardless of the input word and the computation performed until that moment. Therefore, we consider another variant of ordered Turing machines, in which such informations are considered:
Let M be a Turing machine. We denote by $\langle M\rangle$ a binary encoding of this machine (see, for instance, [4]). It is clear that the length of the string $\langle M\rangle$ is a polynomial with respect to the number of states and the working alphabet of the machine M. Let $g : \{0,1,\#\}^* \to \{0,1,\#\}^*$ be a function such that $g(\langle M\rangle\#w_1\#w_2\#\ldots\#w_k) = w'_1\#w'_2\#\ldots\#w'_p$, given that $w_1,\ldots,w_k$ are binary encodings of the IDs that appear in a computation of length k of M (we assume that they appear in this order, and that w_1 is an initial configuration), and $w'_1,\ldots,w'_p$ are the IDs that can be obtained in one move from w_k. Clearly, this function induces canonically an ordering on the computations of a Turing machine. Assume s_1 and s_2 are two (potentially infinite) sequences describing two possible computations of M on w. We say that s_1 is g-smaller than s_2 if the first k IDs of the two computations, which can be encoded by the strings $w_1,\ldots,w_k$, coincide, and $g(\langle M\rangle\#w_1\#w_2\#\ldots\#w_k) = w'_1\#w'_2\#\ldots\#w'_p$, the $k+1$th ID of s_1 is encoded by w'_i, the $k+1$th ID of s_2 is encoded by w'_j, and $i < j$. It is not hard to see that g induces a total order on the computations of M on w; thus we will call such a function *an ordering function*. Therefore, given a finite set of computations of M on w we can define the g-first computation of the set as the one that is g-smaller than all the others.

Definition 4.3 *Let M be a Turing machine, and $g : \{0,1,\#\}^* \to \{0,1,\#\}^*$ be an ordering function. We say that w* is accepted by M with respect to the g-first shortest computation *if there exists at least one finite possible computation of M on w, and the g-first of the shortest computations of M on w is an accepting one; w* is rejected by M w.r.t. the lexicographically first computation *if the g-first shortest computation of M on w is a rejecting computation. We denote by $L^g_{fsc}(M)$* the language accepted by M w.r.t. the g-first shortest computation*, i.e., the set of all words accepted by M, w.r.t. the g-first shortest computation. As in the case of regular Turing machines, we say that the language $L^g_{fsc}(M)$ is decided by M w.r.t. the g-first shortest computation if all the words not contained in $L^g_{fsc}(M)$ are rejected by that machine, w.r.t. the g-first shortest computation.*

It is not surprising that, if g is Turing computable, the class of languages accepted by Turing machines w.r.t. the g-first shortest computation equals **RE**, while the class of languages decided by Turing machines w.r.t. the lexicographically first computation equals **REC**. The time complexity of the computations of Turing machines that decide w.r.t. the g-first shortest computation is defined exactly as in the case of machines that decide w.r.t. shortest computations. We denote by $PTime^g_{fsc}$ *the class of languages decided by Turing machines in polynomial time w.r.t. the g-first shortest computation.* Also, we denote by $PTime_{ofsc}$ the union of all the classes $PTime^g_{fsc}$, where the ordering function g can be computed in polynomial deterministic time. We are now able to show the following theorem.

Theorem 4.4 $PTime_{ofsc} = \mathbf{P}^{\mathbf{NP}}$.

Proof. In fact, we will show that $PTime_{ofsc} = PTime_{lex}$. First, let us observe that the inclusion $PTime_{ofsc} \supseteq PTime_{lex}$ holds canonically. Indeed, the lexicographical order of the computations defined in the previous section is just a particular case of an order defined by an ordering function computable in deterministic polynomial time.

Further, we show that $PTime_{ofsc} \subseteq PTime_{lex}$. Given g an ordering function that can be computed in deterministic polynomial time, let L be a language and M be a Turing machine that decides in polynomial time L w.r.t. the g-first shortest computation. Let us assume, without loss of generality, that the time needed to compute the value of g for a string of k configurations of M, all having the same initial configuration, regardless of the configurations. We define an ordered machine M' and show that it decides L w.r.t. the lexicographically first computation, also in polynomial time.

We will not give the details of the construction of M', as they can be quite tedious, but we will give the main idea implemented by this machine. The machine M' basically simulates the computation of the machine M and keeps on a track (called "memory track") the encoding of M and the encodings of IDs of M that were obtained during the simulated computation. Assume that M' should simulate a move of M, provided that the current state of M is q and the scanned symbols are $(a_1, \ldots, a_k)$. First, M' enters in a state q_g in which it computes the value of the function g having as argument the string saved on the memory track. Suppose that the computed value is the string $w'_1\#w'_2\#\ldots\#w'_p$, and the machine M must make the transition m_i to obtain the ID w'_i from the current ID, for $i \in \{1, \ldots, p\}$. Accordingly, the machine M' enters in a state $q_{m_1,\ldots,m_p}$, and from this state it must make a nondeterministic move that simulates the move of M. But we define M' such that its possibilities, in this case, are ordered: the first comes the move m_1, then the move m_2, and so on, finally coming m_p ($m_1 < m_2 < \ldots < m_p$, in the formalism of ordered machines). Once the move is simulated, the machine M' saves the encoding of the current ID of the simulated machine (again, we may assume that this operation can be done in the same time for any ID, since their length is bounded by a polynomial), and goes on to simulate the next move of M.

It is not hard to see that M' simulates soundly the behavior of M. Basically, M' keeps a history of the computation performed by M and uses a subroutine, computing the function g, to ensure that the lexicographical order of the simulated computations coincides with the order defined by the function g for the machine M and its real computations. Also, the part of the algorithm implemented by M' that is not involved in the actual simulation (that is keeping the history of the simulated computation and computing the values of g) depends only on the number of steps of M simulated until that point and on the input word, so it is quite easy to see that the the shortest computations of M and are simulated by the shortest computations of M'; moreover, the g-first shortest computation of M is simulated by the lexicographically first shortest computation of M'.

It follows that the language L is decided by M' in polynomial time w.r.t. the lexicographically first computation.

To conclude, we showed that $PTime_{ofsc} \subseteq PTime_{lex}$.

It follows that $PTime_{ofsc} = PTime_{lex}$ and, according to Theorem 4.2, we obtain the identity $PTime_{ofsc} = \mathbf{P}^{\mathbf{NP}}$. □

Notice that $\mathbf{P}^{\mathbf{NP}[\log]} \subseteq PTime^{g}_{fsc} \subseteq \mathbf{P}^{\mathbf{NP}}$, for all the ordering functions g which can be com-

puted in polynomial deterministic time. The second inclusion is immediate from the previous Theorem, while the first one follows from the fact that any language in $\mathbf{P}^{\mathbf{NP}[\log]}$ is accepted w.r.t. shortest computations, in polynomial time, by a nondeterministic Turing machine whose shortest computations are either all accepting or all rejecting; clearly, the same machine can be used to show that the given language is in $PTime^{g}_{fsc}$.

It is interesting to see that for some particular ordering functions, as for instance the one that defines the lexicographical order discussed previously, a stronger result holds: $PTime^{g}_{fsc} = \mathbf{P}^{\mathbf{NP}}$ (where g is the ordering function). We leave as an open problem to see if this relation holds for all the ordering functions, or, if not, to see when it hold.

5. Conclusions and Further Work

In this paper we have shown that considering a variant of Turing machine, that decides an input word according to the decisions of the shortest computations of the machine on that word, leads to new characterizations of two well studied complexity classes $\mathbf{P}^{\mathbf{NP}[\log]}$ and $\mathbf{P}^{\mathbf{NP}}$. These results seem interesting since they provide alternative definitions of these two classes, that do not make use of any other notion than the Turing machine (like oracles, reductions, etc.). From a theoretical point of view, an attractive continuation of the present work would be to analyze if the equality results in Theorems 3.3and 4.2 relativize. It is not hard to see that the upper bounds shown in these proofs are true even if we allow all the machines to have access to an arbitrary oracle. It remains to be settled if a similar result holds in the case of the lower bounds.

Nevertheless, other accepting/rejecting conditions related to the shortest computations could be investigated. As we mentioned in the Introduction, several variants of Turing machines that decide a word according to the number of accepting, or rejecting, computations were already studied. We intend to analyze what happens if we use similar conditions for the shortest computations of a Turing machine. In this respect, using the ideas of the proof of Theorem 4.2, one can show that:

Theorem 5.1 *Given a nondeterministic polynomial Turing machine M_1, one can construct a nondeterministic polynomial Turing machine, with access to* **NP***-oracle, M_2, whose computations on an input word correspond bijectively to the short computations of M_1 on the same word, such that two corresponding computations are both either accepting, or rejecting.*

Proof. Let M be a nondeterministic Turing machine working in polynomial time. Also, let f be a polynomial such that the time complexity of the computation of M on each word of length n is less than $f(n)$.

Recall the language $L' = \{x\#w\#w'\#1^k \mid w \in V^*, w'$ is a sequence of consecutive IDs of M, $x \in \{0,1\}$, and, if $x = 1$ (respectively, $x = 0$) there exists an accepting (respectively, rejecting) computation of M on the input word w of length less than k, starting with the sequence of IDs $w'\}$, from the proof of Theorem 4.2. Also recall that L' is in **NP**.

We construct now a nondeterministic Turing machine M', with oracle L', that acts as follows:

1. M' searches (by binary search) the minimum length of a computation of M on w, with length less or equal to $f(|w|)$. In this search, the machine queries the oracle L' for $\mathcal{O}(\log_2(f(|w|)))$

times, asking, in each of these queries, if a string of the form $1\#w\#\epsilon\#1^k$ and $0\#w\#\epsilon\#1^k$, with $k \leq f(|w|)$, is in L'. Let n_0 be the minimum length of a computation, with length less or equal to $f(|w|)$. This step is executed deterministically.

2. Next M' tries to construct nondeterministically, ID by ID, one of the shortest computations of M on w (the length of this computation is n_0), using the oracle L'. Assume that w' is a sequence of IDs identified until a given moment as a prefix of the sequence encoding such a computation, and we try to prolongate this sequence. Assume that $w_1, w_2, \ldots, w_k$ are the IDs that can be obtained from the last ID of w'. We search all the possible i, with $1 \leq 1 \leq k$, such that $0\#w\#w'w_i\#1^{n_0}$ or $1\#w\#w'w_i\#1^{n_0}$ is in L'. Once we have identified these values, denoted by $i_1, \ldots, i_p$, we add, nondeterministically, one of the IDs w_{i_j}, with $j \in \{1, \ldots, p\}$ to the sequence w', and repeat the process described above, until w' contains n_0 IDs.

3. The machine finally checks if the string $1\#w\#w'\#1^{n_0}$, for a w' obtained in one of the possible computations, is in L', and if it is so the computation is accepting, or, if the string $0\#w\#w'\#1^{n_0}$, for a w' obtained in one of the possible computations, is in L', and, in this case, the computation is rejecting.

It is not hard to see that M' correctly computes the length n_0 of the shortest computation of M on an input word w. Also, once this length computed, the shortest computations of M' are identified, and the machine simulates these computations nondeterministically. Thus, the computations of M can be put in a bijectively correspondence with the shortest computations of M': one of the shortest computations of M corresponds to the computation of M' that simulates this shortest computation. Finally, note that M' works in nondeterministic polynomial time.

This concludes the proof of Theorem 5.1. □

This Theorem is useful to show upper bounds on the complexity classes defined by counting the accepting/rejecting shortest computations. Some examples in this direction are: $\mathbf{PP_{sc}} \subseteq \mathbf{PP^{NP}}$ (where $\mathbf{PP_{sc}}$ is the class of decision problems solvable by a nondeterministic polynomial Turing machine which accepts if and only if at least $1/2$ of the shortest computations are accepting, and rejects otherwise) or $\mathbf{BPP_{sc}} \subseteq \mathbf{BPP^{NP}_{path}}$ (where $\mathbf{BPP_{sc}}$ is the class of decision problems solvable by an nondeterministic polynomial Turing machine which accepts if at least $2/3$ of the shortest computations are accepting, and rejects if at least $2/3$ of the shortest computations are rejecting).

Remark 3 *However, in some cases one can show stronger upper bounds; for instance,* $\mathbf{PP_{sc}} \subseteq \mathbf{PP^{NP[log]}_{ctree}}$ *(where* $\mathbf{PP^{NP[log]}_{ctree}}$ *is the class of decision problems solvable by a* **PP***-machine which can make a total number of* $\mathcal{O}(\log n)$ *queries to an* **NP***-language in its entire computation tree, on an input of length* n*). It seems an interesting problem to find lower bounds for such classes, as well.*

Proof. Let M be a nondeterministic Turing machine working in polynomial time. Also, let f be a polynomial such that the time complexity of the computation of M on each word of length n is less than $f(n)$.

Recall the language $L' = \{x\#w\#w'\#1^k \mid w \in V^*, w'$ is a sequence of consecutive IDs of M, $x \in \{0,1\}$, and, if $x = 1$ (respectively, $x = 0$) there exists an accepting (respectively, rejecting) computation of M on the input word w of length less than k, starting with the sequence of IDs $w'\}$, from the proof of Theorem 4.2. Also recall that L' is in **NP**.

We construct now a nondeterministic Turing machine M', with oracle L', that acts as follows:

1. M' searches (by binary search) the minimum length of a computation of M on w, with length less or equal to $f(|w|)$. In this search, the machine queries the oracle L' for $\mathcal{O}(\log_2(f(|w|)))$ times, asking, in each of these queries, if a string of the form $1\#w\#\epsilon\#1^k$ and $0\#w\#\epsilon\#1^k$, with $k \leq f(|w|)$, is in L'. Let n_0 be the minimum length of a computation, with length less or equal to $f(|w|)$. This step is executed deterministically.
2. Next M' simulates the computations of M, counting how many steps it has already simulated. As soon as a computation has more than n_0 steps, it makes a nondeterministic move, with two possible continuations: one possibility is to accept the input, while the other one is to reject it. The computations with n_0 steps are fully simulated (and the decision of M' in those cases coincide with the decision of M).

It is not hard to see that M' correctly computes the length n_0 of the shortest computation of M on an input word w. Also, it is clear that the difference between the number of accepting paths and the number of rejecting paths of M' equals the difference between the number of accepting shortest computations and rejecting shortest computations of M. Finally, note that M' works in nondeterministic polynomial time, and it makes $\mathcal{O}(\log n)$ queries to a **NP** language, summed up over all the possible computations. So, if we see M' as a **PP**-machine, it makes exactly the same decision as M, seen as a $\mathbf{PP_{sc}}$-machine.

Clearly, this implies that $\mathbf{PP_{sc}} \subseteq \mathbf{PP}_{\mathbf{ctree}}^{\mathbf{NP}[\log]}$, and our proof is concluded.

Alternatively, one can see that all the languages from $\mathbf{PP_{sc}}$ can be accepted by deterministic Turing machines working in polynomial time, that are allowed to make $\mathcal{O}(\log n)$ queries to **NP** and exactly one query to **PP**, which gives the decision of the machine, on an input of length n. The only difference from the above idea is that step 2 of the algorithm is replaced by a **PP**-language query.

Another remark is that the idea presented above holds in the case of other classes, like $\oplus\mathbf{P}$ (where $\oplus\mathbf{P}$ is the class of decision problems solvable by a nondeterministic polynomial Turing machine which accepts if and only if the number of accepting paths is even), which was introduced in [6].

One can show, similarly to the above, that $\oplus\mathbf{P}_{sc} \subseteq \oplus\mathbf{P}_{\mathbf{ctree}}^{\mathbf{NP}[\log]}$ (where $\oplus\mathbf{P}_{\mathbf{ctree}}^{\mathbf{NP}[\log]}$ is the class of decision problems solvable by a $\oplus\mathbf{P}$-machine which can make a total number of $\mathcal{O}(\log n)$ queries to an **NP**-language in its entire computation tree, on an input of length n). The only difference from the above proof is that in step 2 of the algorithm, as soon as a computation has more than n_0 steps, the machine M makes a nondeterministic move with three possible continuations: two possibilities are to accept the input, and the other is to reject it.

The same idea works for the class **RP**, of decision problems solvable by a nondeterministic polynomial Turing machine which accepts if and only if at least 1/2 of computation paths accept and rejects if and only if all computation paths reject, introduced in [1]. In this case we get $\mathbf{RP}_{sc} \subseteq \mathbf{RP}_{\mathbf{ctree}}^{\mathbf{NP}[\log]}$ (where $\mathbf{RP}_{\mathbf{ctree}}^{\mathbf{NP}[\log]}$ is the class of decision problems solvable by a **RP**-machine which can make a total number of $\mathcal{O}(\log n)$ queries to an **NP**-language in its entire computation tree, on an input of length n).

According to Remark 2, one can see that the lower bounds $\mathbf{P}^{\mathbf{NP}[\log]} \subseteq \mathbf{PP_{sc}}$, $\mathbf{P}^{\mathbf{NP}[\log]} \subseteq \oplus\mathbf{P_{sc}}$ and $\mathbf{P}^{\mathbf{NP}[\log]} \subseteq \mathbf{RP_{sc}}$ hold. □

References

[1] J. GILL, Computational Complexity of Probabilistic Turing Machines. *SIAM J. Comput.* **6** (1977) 4, 675–695.

[2] J. HARTMANIS, R. E. STEARNS, On the Computational Complexity of Algorithms. *Trans. Amer. Math. Soc.* **117** (1965), 533–546.

[3] E. HEMASPAANDRA, L. A. HEMASPAANDRA, J. ROTHE, Exact Analysis of Dodgson Elections: Lewis Carroll's 1876 Voting System is Complete for Parallel Access to NP. In: *ICALP*. LNCS 1256, Springer, 1997, 214–224.

[4] J. E. HOPCROFT, J. D. ULLMAN, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[5] F. MANEA, M. MARGENSTERN, V. MITRANA, M. J. PÉREZ-JIMÉNEZ, A New Characterization of NP, P, and PSPACE with Accepting Hybrid Networks of Evolutionary Processors. *Theory Comput. Syst.* **46** (2010) 2, 174–192.

[6] C. H. PAPADIMITRIOU, S. ZACHOS, Two remarks on the power of counting. In: *Theoretical Computer Science*. LNCS 145, Springer, 1983, 269–276.

[7] C. M. PAPADIMITRIOU, *Computational complexity*. Addison-Wesley, 1994.

[8] M. J. PÉREZ-JIMÉNEZ, A Computational Complexity Theory in Membrane Computing. In: *Workshop on Membrane Computing*. LNCS 5957, Springer, 2009, 125–148.

[9] K. W. WAGNER, More Complicated Questions About Maxima and Minima, and Some Closures of NP. *Theor. Comput. Sci.* **51** (1987), 53–80.

[10] K. W. WAGNER, Bounded Query Classes. *SIAM J. Comput.* **19** (1990) 5, 833–846.