Deciding According to the Shortest Computations

Florin Manea

Faculty of Computer Science, Otto-von-Guericke-University of Magdeburg,
Faculty of Mathematics and Computer Science, University of Bucharest.

CiE 2011 - Sofia

Outline

1. Introduction
2. Definitions
3. Our results

1 Introduction

2 Definitions

3 Our results

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID),

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), and its children are the IDs encoding the possible configurations in which the machine can be found after a (nondeterministic) move is performed.

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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;

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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; each leaf of the tree encodes a final ID.

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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; each leaf of the tree encodes a final ID.
  – 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.

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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; each leaf of the tree encodes a final ID.
  – 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.
- For finite computations, one can check whether a word is accepted/rejected by searching in the computation-tree for an accepting ID-leaf.

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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; each leaf of the tree encodes a final ID.
  – 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.
- For finite computations, one can check whether a word is accepted/rejected by searching in the computation-tree for an accepting ID-leaf. Theoretically: simultaneous traversal of all the possible paths in the tree.

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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; each leaf of the tree encodes a final ID.
  – 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.
- For finite computations, one can check whether a word is accepted/rejected by searching in the computation-tree for an accepting ID-leaf. Theoretically: simultaneous traversal of all the possible paths in the tree. In practice: traversing each path at a time, until an accepting ID is found, or until the whole tree was traversed.

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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; each leaf of the tree encodes a final ID.
  – 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.
- For finite computations, one can check whether a word is accepted/rejected by searching in the computation-tree for an accepting ID-leaf. Theoretically: simultaneous traversal of all the possible paths in the tree. In practice: traversing each path at a time, until an accepting ID is found, or until the whole tree was traversed. Thus, a very time consuming task.

Computations in Turing machines

- The computation of a nondeterministic machine: a (potentially infinite) tree.
  – Each node of this tree is an instantaneous description (ID), 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; each leaf of the tree encodes a final ID.
  – 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.
- For finite computations, one can check whether a word is accepted/rejected by searching in the computation-tree for an accepting ID-leaf. Theoretically: simultaneous traversal of all the possible paths in the tree. In practice: traversing each path at a time, until an accepting ID is found, or until the whole tree was traversed. Thus, a very time consuming task. Alternative ways of using nondeterministic machines?

Deciding by the shortest computations

- 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).

Deciding by the shortest computations

- 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.

Deciding by the shortest computations

- 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.

OR

- 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 children of a node in the computations-tree).

1 Introduction

2 Definitions

3 Our results

Basic definitions: Turing machines

A k-tape Turing machine: $M = (Q, V, U, q_o, acc, rej, B, \delta)$, Q is a finite set of states, q_0 is the initial state, acc and rej are the accepting / rejecting state, U is the working alphabet, B is the blank-symbol, V is the input alphabet, $\delta : (Q \setminus \{acc, rej\}) \times U^k \to 2^{(Q \times (U \setminus \{B\})^k \times \{L,R\}^k)}$ is the transition function.

Basic definitions: Turing machines

A k-tape Turing machine: $M = (Q, V, U, q_o, acc, rej, B, \delta)$, Q is a finite set of states, q_0 is the initial state, acc and rej are the accepting / rejecting state, U is the working alphabet, B is the blank-symbol, V is the input alphabet, $\delta : (Q \setminus \{acc, rej\}) \times U^k \to 2^{(Q \times (U \setminus \{B\})^k \times \{L,R\}^k)}$ is the transition function.

An ID: a word that encodes the state of the machine and the contents of the tapes, the position of the tape heads, at a given moment of the computation. An ID is final if the state encoded in it is acc or rej.

Basic definitions: Turing machines

A k-tape Turing machine: $M = (Q, V, U, q_o, acc, rej, B, \delta)$, Q is a finite set of states, q_0 is the initial state, *acc* and *rej* are the accepting / rejecting state, U is the working alphabet, B is the blank-symbol, V is the input alphabet, $\delta : (Q \setminus \{acc, rej\}) \times U^k \to 2^{(Q \times (U \setminus \{B\})^k \times \{L,R\}^k)}$ is the transition function.

An ID: a word that encodes the state of the machine and the contents of the tapes, the position of the tape heads, at a given moment of the computation. An ID is final if the state encoded in it is *acc* or *rej*.

A computation of a TM on a word is a sequence of IDs: each ID is transformed into the next one by a move of the machine. If the computation is finite then the sequence is also finite and ends with a final ID; a computation is accepting (respectively, rejecting), if the final ID encodes *acc* (respectively, *rej*). All the possible computations of a nondeterministic TM on a given word can be described as a (potentially infinite) tree of IDs.

Basic definitions: Turing machines

A word is accepted by a TM 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 TM if all its words are accepted, and no other words are accepted (respectively, all the other words are rejected).
The class of languages accepted by TM: **RE** (recursively enumerable languages). The class of languages decided by Turing machines: **REC** (recursive languages).

Basic definitions: Turing machines

A word is accepted by a TM 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 TM if all its words are accepted, and no other words are accepted (respectively, all the other words are rejected).
The class of languages accepted by TM: **RE** (recursively enumerable languages). The class of languages decided by Turing machines: **REC** (recursive languages).

A non-deterministic machine is of time complexity $f(n)$ if no sequence of choices of move causes the machine to make more than $f(n)$ moves.
A language L is decided in polynomial time if there exists M and a polynomial f such that M is of time complexity $f(n)$ and M accepts L.

The class of languages decided by deterministic TMs in polynomial time is **P** and the class of languages decided by nondeterministic TMs in polynomial time is **NP**.

Basic definitions: Oracles

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.

Basic definitions: Complete problems

Complete problems for a class, with respect to polynomial time reductions: any problem of that class can be reduced in polynomial time to the complete problem.

Basic definitions: Complete problems

Complete problems for a class, with respect to polynomial time reductions: any problem of that class can be reduced in polynomial time to the complete problem.

The following problem is complete for $\mathbf{P}^{\mathbf{NP}}$, with respect to polynomial time reductions (Wagner, 1987):

Problem

(Odd - Traveling Salesman Problem, TSP_{odd}) Let n be a natural number, and d be a function $d : \{1, \ldots, n\} \times \{1, \ldots, n\} \to \mathbf{N}$. 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.

Input of this problem: the number n, and n^2 numbers representing $d(i, j)$, for all i, j. Size of the input: the number of bits needed to represent these values.

Basic definitions: Complete problems

The following problem is $\mathbf{P}^{\mathbf{NP}[\log]}$-complete, with respect to polynomial time reductions (Hemaspaandra et al., 1997):

Problem

(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 (permutations) on C. Decide if $Score(C, c, V) \leq Score(C, d, V)$.

The $Score(C, c, V)$: minimum number of exchanges of two adjacent elements from the permutations of V, needed to make c preferred to each other candidate in strictly more than half of the preference orders (Condorcet winner).

Input of this problem: the 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 $\{1, \ldots, n\}$. Size of the input is $\mathcal{O}(\#(V) n \log n)$.

1 Introduction

2 Definitions

3 Our results

Definitions 1

Definition

Let M be a Turing machine and w be a word over the input alphabet of M.

Definitions 1

Definition

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;

Definitions 1

Definition

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.

Definitions 1

Definition

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.

Definitions 1

Definition

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.

Definitions 1: Complexity

Definition

Let M be a Turing machine, and w be a word over the input alphabet of M.

Definitions 1: Complexity

Definition

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.

Definitions 1: Complexity

Definition

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 $L_{sc}(M) = L$.

Definitions 1: Complexity

Definition

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.*

Results 1

Remark

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**.

Theorem

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

Proof:
"$\subseteq$": binary search for the length of the shortest computations + simulation.
"$\supseteq$": we construct a machine M deciding *DodRank* w.r.t. shortest computations.

Results 1

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.

Definitions 2

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.

Definitions 2

Definition

Let M be an ordered Turing machine, and w be a word over the input alphabet of M.

Definitions 2

Definition

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.

Definitions 2: Complexity

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.*

Results 2

Theorem

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

Proof:
"$\subseteq$": perform binary search for the length of the shortest computation and then identify the lexicographically-first such shortest computation by asking queries to an NP-oracle.
"$\supseteq$": we construct a machine M deciding $OddTSP$ w.r.t. shortest computations.

Results 2

1. M, deterministically, writes the identical permutation π of $\{1, \ldots, n\}$ and computes 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.

Results 1 and 2

Remark

Note 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, 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.*

Other Results and Further work

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

Other Results and Further work

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

Other Results and Further work

- $PTime_{nm} = \mathbf{P}^{\mathbf{NP}[\log]}$.
- 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.
  – $\mathbf{BPP}_{\mathbf{sc}} \subseteq \mathbf{BPP}^{\mathbf{NP}}_{\mathbf{path}}$ (where $\mathbf{BPP}_{\mathbf{sc}}$ 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).
  – $\mathbf{PP}_{\mathbf{sc}} \subseteq \mathbf{PP}^{\mathbf{NP}}$ (where $\mathbf{PP}_{\mathbf{sc}}$ 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)

Other Results and Further work

- $PTime_{nm} = \mathbf{P}^{\mathbf{NP}[\log]}$.
- 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.
  – $\mathbf{BPP}_{\mathbf{sc}} \subseteq \mathbf{BPP}^{\mathbf{NP}}_{\mathbf{path}}$ (where $\mathbf{BPP}_{\mathbf{sc}}$ 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).
  – $\mathbf{PP}_{\mathbf{sc}} \subseteq \mathbf{PP}^{\mathbf{NP}}$ (where $\mathbf{PP}_{\mathbf{sc}}$ 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)
  – $\mathbf{PP}_{\mathbf{sc}} \subseteq \mathbf{PP}^{\mathbf{NP}[\log]}_{\mathbf{ctree}}$ (where $\mathbf{PP}^{\mathbf{NP}[\log]}_{\mathbf{ctree}}$ 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).

Other Results and Further work

- Can we characterize precisely other classes by means of shortest computations? ($\mathbf{PP}, \mathbf{BPP}, \ldots$)

Other Results and Further work

- Can we characterize precisely other classes by means of shortest computations? ($\mathbf{PP}, \mathbf{BPP}, \ldots$)
- What if the order in which we traverse the computations-tree is not predefined, but depends on the history of each computation?

Other Results and Further work

- Can we characterize precisely other classes by means of shortest computations? ($\mathbf{PP}, \mathbf{BPP}, \ldots$)
- What if the order in which we traverse the computations-tree is not predefined, but depends on the history of each computation?
- Applications...

Other Results and Further work

- Can we characterize precisely other classes by means of shortest computations? ($\mathbf{PP}, \mathbf{BPP}, \ldots$)
- What if the order in which we traverse the computations-tree is not predefined, but depends on the history of each computation?
- Applications... Complexity measures for masively parallel computing models (Networks of Evolutionary Processors - DLT 2011).

THANK YOU!

The speaker's acknowledges the support of the
Alexander von Humboldt Foundation.