Wizards vs. Time Machines

Jalex Stark

Department of Mathematics
California Institute of Technology

Caltech Undergraduate Math Seminar, 6 January 2017

Outline

1. What is Complexity Theory?
    - Models of computation
    - Complexity classes
    - Interactive Proofs
    - Closed Timelike Curves

Decision problems

Informally, a computational *decision problem* is a yes/no question asked about an input which can be encoded as a string.
Some examples:

Decision problems

Informally, a computational *decision problem* is a yes/no question asked about an input which can be encoded as a string.
Some examples:

- Given three matrices A, B, C, does $AB = C$?

Decision problems

Informally, a computational *decision problem* is a yes/no question asked about an input which can be encoded as a string.
Some examples:

- Given three matrices A, B, C, does $AB = C$?
- Given two graphs, are they isomorphic?

Decision problems

Informally, a computational *decision problem* is a yes/no question asked about an input which can be encoded as a string.
Some examples:

- Given three matrices A, B, C, does $AB = C$?
- Given two graphs, are they isomorphic?
- Given a graph, is it possible to assign one of three colors to each vertex such that no adjacent vertices have the same color?

Decision problems

Informally, a computational *decision problem* is a yes/no question asked about an input which can be encoded as a string.
Some examples:

- Given three matrices A, B, C, does $AB = C$?
- Given two graphs, are they isomorphic?
- Given a graph, is it possible to assign one of three colors to each vertex such that no adjacent vertices have the same color?
- Given a state in the board game Hex, does the first player win under optimal play?

Decision problems

Informally, a computational *decision problem* is a yes/no question asked about an input which can be encoded as a string.
Some examples:

- Given three matrices A, B, C, does $AB = C$?
- Given two graphs, are they isomorphic?
- Given a graph, is it possible to assign one of three colors to each vertex such that no adjacent vertices have the same color?
- Given a state in the board game Hex, does the first player win under optimal play?

Decision problems, II

One of the main goals of complexity theory is to classify how *hard* problems are to solve. In order to give a notion of *solving a problem*, let's be more precise about what a problem is.

Decision problems, II

One of the main goals of complexity theory is to classify how *hard* problems are to solve. In order to give a notion of *solving a problem*, let's be more precise about what a problem is.

Definition (Decision problem)

Fix some finite alphabet Σ. Let Σ^* be the set of finite strings with characters from Σ. A *decision problem* or *language* L is a subset of Σ^*.

Decision problems, II

One of the main goals of complexity theory is to classify how *hard* problems are to solve. In order to give a notion of *solving a problem*, let's be more precise about what a problem is.

Definition (Decision problem)

Fix some finite alphabet Σ. Let Σ^* be the set of finite strings with characters from Σ. A *decision problem* or *language* L is a subset of Σ^*.

For example, let $\Sigma = \{(,), 0, 1\}$ and let MATRIX MULTIPLICATION be the set of strings (A, B, C) for which each of A, B, C is an n^2-length list of binary strings, and $AB = C$ when these are interpreted as $n \times n$ matrices of binary integers.

Outline

1. What is Complexity Theory?
   - Models of computation
   - Complexity classes
   - Interactive Proofs
   - Closed Timelike Curves

What should a model of computation be?

- Fix a process that takes in an input string $x \in \Sigma^*$ and has some output behavior.

What should a model of computation be?

- Fix a process that takes in an input string $x \in \Sigma^*$ and has some output behavior.
- Informally, we say that a process *decides* L if it has some behavior for $x \in L$ and a different behavior for $x \notin L$.

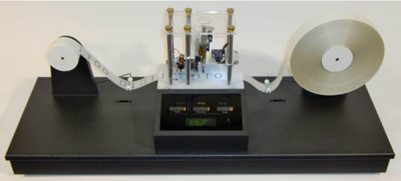
What should a model of computation be?

- Fix a process that takes in an input string $x \in \Sigma^*$ and has some output behavior.
- Informally, we say that a process *decides* L if it has some behavior for $x \in L$ and a different behavior for $x \notin L$.
- A *model of computation* is a way to specify what kind of thing the computational process can be.

A precise model of computation

Definition (Turing Machine)

A *Turing machine* T consists of...



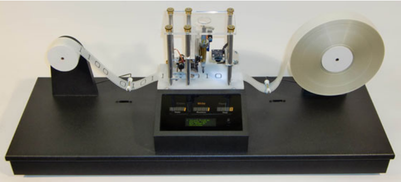
(Photo from `http://www.aturingmachine.com/`)

A precise model of computation

Definition (Turing Machine)

A *Turing machine* T consists of...

- A finite set of states Γ



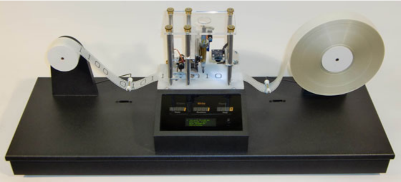
(Photo from http://www.aturingmachine.com/)

A precise model of computation

Definition (Turing Machine)

A *Turing machine T* consists of...

- A finite set of states Γ
- An infinite tape for symbols to sit on



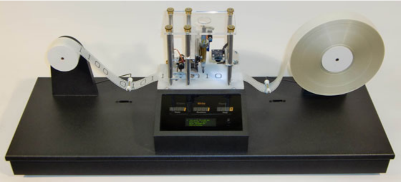
(Photo from `http://www.aturingmachine.com/`)

A precise model of computation

Definition (Turing Machine)

A *Turing machine T* consists of...

- A finite set of states Γ
- An infinite tape for symbols to sit on
- A "head" which points to some square on the tape.



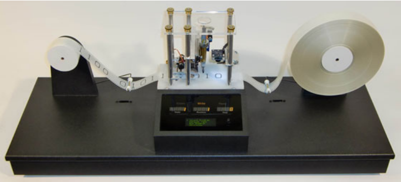
(Photo from `http://www.aturingmachine.com/`)

A precise model of computation

Definition (Turing Machine)

A *Turing machine* T consists of...

- A finite set of states Γ
- An infinite tape for symbols to sit on
- A "head" which points to some square on the tape.
- A finite list of instructions, one for each element of $\Gamma \times \Sigma$



(Photo from `http://www.aturingmachine.com/`)

A precise model of computation, II

Definition (Turing Machine)

At time-step, the Turing machine reads its internal state and the symbol at its current head, and does some of the following:

- Write a symbol to the current head spot

A precise model of computation, II

Definition (Turing Machine)

At time-step, the Turing machine reads its internal state and the symbol at its current head, and does some of the following:

- Change the state
- Write a symbol to the current head spot
- Move the head to a spot on the tape

A precise model of computation, II

Definition (Turing Machine)

At time-step, the Turing machine reads its internal state and the symbol at its current head, and does some of the following:

- Change the state
- Write a symbol to the current head spot
- Move the head to a spot on the tape

Computation *halts* when the machine enters the *accept state* or the *reject state*.

Outline

1. What is Complexity Theory?
   - Models of computation
   - Complexity classes
   - Interactive Proofs
   - Closed Timelike Curves

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

- when T is run on $x \in L$, T halts and accepts, and

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

- when T is run on $x \in L$, T halts and accepts, and
- when T is run on $x \notin L$, T halts and rejects.

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

- when T is run on $x \in L$, T halts and accepts, and
- when T is run on $x \notin L$, T halts and rejects.

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

- when T is run on $x \in L$, T halts and accepts, and
- when T is run on $x \notin L$, T halts and rejects.

Let $f : \mathbb{N} \to \mathbb{N}$. We say that T decides L *in time* f if

- T decides L.

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

- when T is run on $x \in L$, T halts and accepts, and
- when T is run on $x \notin L$, T halts and rejects.

Let $f : \mathbb{N} \to \mathbb{N}$. We say that T decides L *in time* f if

- T decides L.
- When T is run on an input of length at most n, T halts within $f(n)$ steps.

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

- when T is run on $x \in L$, T halts and accepts, and
- when T is run on $x \notin L$, T halts and rejects.

Let $f : \mathbb{N} \to \mathbb{N}$. We say that T decides L *in time* f if

- T decides L.
- When T is run on an input of length at most n, T halts within $f(n)$ steps.

Polynomial time

Definition

Decidability We say that a language L is *decidable* by Turing machine T if

- when T is run on $x \in L$, T halts and accepts, and
- when T is run on $x \notin L$, T halts and rejects.

Let $f : \mathbb{N} \to \mathbb{N}$. We say that T decides L *in time* f if

- T decides L.
- When T is run on an input of length at most n, T halts within $f(n)$ steps.

Definition

P We say that $L \in \mathrm{P}$ or L is *decidable in polynomial time* if there is some polynomial p and Turing machine T such that T decides L in time p.

P is robust

Our definition of Turing machine is arbitrary.

P is robust

Our definition of Turing machine is arbitrary.
The class of polynomials is closed under multiplication and composition.

P is robust

Our definition of Turing machine is arbitrary.
The class of polynomials is closed under multiplication and composition.
So P is closed under subroutines and poly-length for-loops.

P is robust

Our definition of Turing machine is arbitrary.
The class of polynomials is closed under multiplication and composition.
So P is closed under subroutines and poly-length for-loops.
In particular, any two sufficiently powerful models of a computer can simulate each other in polynomial time.

P is robust

Our definition of Turing machine is arbitrary.
The class of polynomials is closed under multiplication and composition.
So P is closed under subroutines and poly-length for-loops.
In particular, any two sufficiently powerful models of a computer can simulate each other in polynomial time.
P would be the same if we replace our Turing machine with a multi-tape Turing machine, a Python program, DNA-based computation, etc.

A problem in P

Example

MATRIXMULTIPLICATION is in P.

A problem in P

Example

MATRIXMULTIPLICATION is in P.

Proof.

The standard matrix multiplication algorithm for two $n \times n$ matrices takes about n^3 arithmetic operations.

A problem in P

Example

MATRIXMULTIPLICATION is in P.

Proof.

The standard matrix multiplication algorithm for two $n \times n$ matrices takes about n^3 arithmetic operations. Implement this algorithm in your favorite programming language. $\square$

Beyond P

P captures the notion of "solvable in a reasonable amount of time on a normal computer". For our purposes, we will consider polytime computations as a "baseline" upon which everything else rests.

Beyond P

P captures the notion of "solvable in a reasonable amount of time on a normal computer". For our purposes, we will consider polytime computations as a "baseline" upon which everything else rests. In the rest of the talk, we'll discuss different ways to augment the power of polytime Turing machines by providing additional resources.

Randomness as a resource

Randomness is a useful resource!

Randomness as a resource

Randomness is a useful resource!

Definition

We say $L \in \mathrm{BPP}$ if there is a deterministic polynomial time algorithm M such that when r is chosen uniformly at random,

- If $x \in L$, then $M(x, r)$ accepts with probability at least $\frac{2}{3}$.

It is believed that $\mathrm{P} = \mathrm{BPP}$, however, there are problems known to be in BPP not currently known to be in P.

Randomness as a resource

Randomness is a useful resource!

Definition

We say $L \in \mathrm{BPP}$ if there is a deterministic polynomial time algorithm M such that when r is chosen uniformly at random,

- If $x \in L$, then $M(x, r)$ accepts with probability at least $\frac{2}{3}$.
- If $x \notin L$, then $M(x, r)$ accepts with probability at most $\frac{1}{3}$.

It is believed that $\mathrm{P} = \mathrm{BPP}$, however, there are problems known to be in BPP not currently known to be in P.

Randomness as a resource

Randomness is a useful resource!

Definition

We say $L \in \mathrm{BPP}$ if there is a deterministic polynomial time algorithm M such that when r is chosen uniformly at random,

- If $x \in L$, then $M(x, r)$ accepts with probability at least $\frac{2}{3}$.
- If $x \notin L$, then $M(x, r)$ accepts with probability at most $\frac{1}{3}$.

It is believed that $\mathrm{P} = \mathrm{BPP}$, however, there are problems known to be in BPP not currently known to be in P.

Randomness as a resource

Randomness is a useful resource!

Definition

We say $L \in \mathrm{BPP}$ if there is a deterministic polynomial time algorithm M such that when r is chosen uniformly at random,

- If $x \in L$, then $M(x, r)$ accepts with probability at least $\frac{2}{3}$.
- If $x \notin L$, then $M(x, r)$ accepts with probability at most $\frac{1}{3}$.

It is believed that $\mathrm{P} = \mathrm{BPP}$, however, there are problems known to be in BPP not currently known to be in P. Before 2002, primality testing was such a problem.

A problem for which randomness helps.

Definition

Polynomial Identity Testing PIT is the decision problem: given a parenthesized expression describing a multivariate polynomial p over a finite field F, is p identically zero?

A problem for which randomness helps.

Definition

Polynomial Identity Testing PIT is the decision problem: given a parenthesized expression describing a multivariate polynomial p over a finite field F, is p identically zero?

By *identically zero*, we mean that all of the coefficients of the monomials are 0.

A problem for which randomness helps.

Definition

Polynomial Identity Testing PIT is the decision problem: given a parenthesized expression describing a multivariate polynomial p over a finite field F, is p identically zero?

By *identically zero*, we mean that all of the coefficients of the monomials are 0. For example, let F be the field with two elements. Then

A problem for which randomness helps.

Definition

Polynomial Identity Testing PIT is the decision problem: given a parenthesized expression describing a multivariate polynomial p over a finite field F, is p identically zero?

By *identically zero*, we mean that all of the coefficients of the monomials are 0. For example, let F be the field with two elements. Then

- $x^4 + y^4 + (x + y)^4$ is identically 0,

A problem for which randomness helps.

Definition

Polynomial Identity Testing PIT is the decision problem: given a parenthesized expression describing a multivariate polynomial p over a finite field F, is p identically zero?

By *identically zero*, we mean that all of the coefficients of the monomials are 0. For example, let F be the field with two elements. Then

- $x^4 + y^4 + (x + y)^4$ is identically 0,
- while $x^3 + y^3 + (x + y)^3 = x^2y + xy^2$ is not identically 0

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F.

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F. Let S be a finite subset of F (e.g. if F is finite, we can set $S = F$).

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F. Let S be a finite subset of F (e.g. if F is finite, we can set $S = F$). Choose $r_1, \ldots, r_n$ independently and uniformly from S. Then

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F. Let S be a finite subset of F (e.g. if F is finite, we can set $S = F$). Choose $r_1, \ldots, r_n$ independently and uniformly from S. Then

$$\Pr_{r_1,\ldots,r_n}[p(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}$$

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F. Let S be a finite subset of F (e.g. if F is finite, we can set $S = F$). Choose $r_1, \ldots, r_n$ independently and uniformly from S. Then

$$\Pr_{r_1,\ldots,r_n}[p(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}$$

If a nonzero polynomial has degree which is small compared to the size of the field, then a random point is not a zero with high probability.

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F. Let S be a finite subset of F (e.g. if F is finite, we can set $S = F$). Choose $r_1, \ldots, r_n$ independently and uniformly from S. Then

$$\Pr_{r_1,\ldots,r_n}[p(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}$$

If a nonzero polynomial has degree which is small compared to the size of the field, then a random point is not a zero with high probability.
This suggests a BPP algorithm for PIT: pick a random point and evaluate the polynomial. If it's a zero, declare that the polynomial is zero. If not, declare that it's not.

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F. Let S be a finite subset of F (e.g. if F is finite, we can set $S = F$). Choose $r_1, \ldots, r_n$ independently and uniformly from S. Then

$$\Pr_{r_1,\ldots,r_n}[p(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}$$

If a nonzero polynomial has degree which is small compared to the size of the field, then a random point is not a zero with high probability.

This suggests a BPP algorithm for PIT: pick a random point and evaluate the polynomial. If it's a zero, declare that the polynomial is zero. If not, declare that it's not. (If the degree is not small

A randomized algorithm

Lemma (Schwartz-Zippel)

Let $p = p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree d over a field F. Let S be a finite subset of F (e.g. if F is finite, we can set $S = F$). Choose $r_1, \ldots, r_n$ independently and uniformly from S. Then

$$\Pr_{r_1,\ldots,r_n}[p(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}$$

If a nonzero polynomial has degree which is small compared to the size of the field, then a random point is not a zero with high probability.

This suggests a BPP algorithm for PIT: pick a random point and evaluate the polynomial. If it's a zero, declare that the polynomial is zero. If not, declare that it's not. (If the degree is not small

Outline

1. What is Complexity Theory?
   - Models of computation
   - Complexity classes
   - Interactive Proofs
   - Closed Timelike Curves

One-way, one-round proof system

You want to solve a decision problem. You show the problem to Merlin and he gives you a piece of advice.

One-way, one-round proof system

You want to solve a decision problem. You show the problem to Merlin and he gives you a piece of advice. You don't trust him, so you have to check it yourself.

Definition (NP)

A language L is in NP if there is a poly time algorithm V (the *verifier*) such that

- If $x \in L$, then there is some witness w such that $M(x, w)$ accepts.

One-way, one-round proof system

You want to solve a decision problem. You show the problem to Merlin and he gives you a piece of advice. You don't trust him, so you have to check it yourself.

Definition (NP)

A language L is in NP if there is a poly time algorithm V (the *verifier*) such that

- If $x \in L$, then there is some witness w such that $M(x, w)$ accepts.
- If $x \notin L$, then for any candidate witness w, $M(x, w)$ rejects.

One-way, one-round proof system

You want to solve a decision problem. You show the problem to Merlin and he gives you a piece of advice. You don't trust him, so you have to check it yourself.

Definition (NP)

A language L is in NP if there is a poly time algorithm V (the *verifier*) such that

- If $x \in L$, then there is some witness w such that $M(x, w)$ accepts.
- If $x \notin L$, then for any candidate witness w, $M(x, w)$ rejects.

One-way, one-round proof system

You want to solve a decision problem. You show the problem to Merlin and he gives you a piece of advice. You don't trust him, so you have to check it yourself.

Definition (NP)

A language L is in NP if there is a poly time algorithm V (the *verifier*) such that

- If $x \in L$, then there is some witness w such that $M(x, w)$ accepts.
- If $x \notin L$, then for any candidate witness w, $M(x, w)$ rejects.

Additionally, we require that the length of w is bounded by a polynomial in the length of x.

One-way, one-round proof system

You want to solve a decision problem. You show the problem to Merlin and he gives you a piece of advice. You don't trust him, so you have to check it yourself.

Definition (NP)

A language L is in NP if there is a poly time algorithm V (the *verifier*) such that

- If $x \in L$, then there is some witness w such that $M(x, w)$ accepts.
- If $x \notin L$, then for any candidate witness w, $M(x, w)$ rejects.

Additionally, we require that the length of w is bounded by a polynomial in the length of x.

We'll refer to w variously as a witness, proof, or certificate.

Example

Two-way, one-round proof system

You want to solve a decision problem. You start by generating a question which you ask to Merlin. Merlin gives you an answer, and then you use his answer to come to a decision.

Example (Graph non-isomorphism)

You have two graphs, G and H, which you suspect are isomorphic. You want to prove this with Merlin's help. You undertake the following protocol:

Two-way, one-round proof system

You want to solve a decision problem. You start by generating a question which you ask to Merlin. Merlin gives you an answer, and then you use his answer to come to a decision.

Example (Graph non-isomorphism)

You have two graphs, G and H, which you suspect are isomorphic. You want to prove this with Merlin's help. You undertake the following protocol:

- Flip a coin to pick one of the graphs.

Two-way, one-round proof system

You want to solve a decision problem. You start by generating a question which you ask to Merlin. Merlin gives you an answer, and then you use his answer to come to a decision.

Example (Graph non-isomorphism)

You have two graphs, G and H, which you suspect are isomorphic. You want to prove this with Merlin's help. You undertake the following protocol:

- Flip a coin to pick one of the graphs.
- Randomly permute the graph you picked; hand it to Merlin.

Two-way, one-round proof system

You want to solve a decision problem. You start by generating a question which you ask to Merlin. Merlin gives you an answer, and then you use his answer to come to a decision.

Example (Graph non-isomorphism)

You have two graphs, G and H, which you suspect are isomorphic. You want to prove this with Merlin's help. You undertake the following protocol:

- Flip a coin to pick one of the graphs.
- Randomly permute the graph you picked; hand it to Merlin.
- Ask Merlin to tell you which graph you handed him.

Two-way, one-round proof system

You want to solve a decision problem. You start by generating a question which you ask to Merlin. Merlin gives you an answer, and then you use his answer to come to a decision.

Example (Graph non-isomorphism)

You have two graphs, G and H, which you suspect are isomorphic. You want to prove this with Merlin's help. You undertake the following protocol:

- Flip a coin to pick one of the graphs.
- Randomly permute the graph you picked; hand it to Merlin.
- Ask Merlin to tell you which graph you handed him.

Two-way, one-round proof system

You want to solve a decision problem. You start by generating a question which you ask to Merlin. Merlin gives you an answer, and then you use his answer to come to a decision.

Example (Graph non-isomorphism)

You have two graphs, G and H, which you suspect are isomorphic. You want to prove this with Merlin's help. You undertake the following protocol:

- Flip a coin to pick one of the graphs.
- Randomly permute the graph you picked; hand it to Merlin.
- Ask Merlin to tell you which graph you handed him.

If $G \ncong H$, then Merlin can always distinguish.

Two-way, one-round proof system

You want to solve a decision problem. You start by generating a question which you ask to Merlin. Merlin gives you an answer, and then you use his answer to come to a decision.

Example (Graph non-isomorphism)

You have two graphs, G and H, which you suspect are isomorphic. You want to prove this with Merlin's help. You undertake the following protocol:

- Flip a coin to pick one of the graphs.
- Randomly permute the graph you picked; hand it to Merlin.
- Ask Merlin to tell you which graph you handed him.

If $G \ncong H$, then Merlin can always distinguish.
If $G \cong H$, then the situation is identical from Merlin's point of view, regardless of which graph you picked. He will be right with probability exactly $\frac{1}{2}$

AM

Definition

We say a language is in AM if there is a randomized poly-time algorithm A, a function M, and a verifier V such that when x is an input of length at most n,

- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at least $1 - \varepsilon$

We require that the gap between ε and η is at least $1/p(n)$ for some polynomial p.

AM

Definition

We say a language is in AM if there is a randomized poly-time algorithm A, a function M, and a verifier V such that when x is an input of length at most n,

- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at least $1 - \varepsilon$
- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at most $1 - \eta$

We require that the gap between ε and η is at least $1/p(n)$ for some polynomial p.

AM

Definition

We say a language is in AM if there is a randomized poly-time algorithm A, a function M, and a verifier V such that when x is an input of length at most n,

- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at least $1 - \varepsilon$
- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at most $1 - \eta$

We require that the gap between ε and η is at least $1/p(n)$ for some polynomial p.

AM

Definition

We say a language is in AM if there is a randomized poly-time algorithm A, a function M, and a verifier V such that when x is an input of length at most n,

- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at least $1 - \varepsilon$
- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at most $1 - \eta$

We require that the gap between ε and η is at least $1/p(n)$ for some polynomial p.

Fact

NP $\subseteq$ AM.

AM

Definition

We say a language is in AM if there is a randomized poly-time algorithm A, a function M, and a verifier V such that when x is an input of length at most n,

- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at least $1 - \varepsilon$
- If $x \in L$, then $V(x, A(x), M(x, A(x)))$ accepts with probability at most $1 - \eta$

We require that the gap between ε and η is at least $1/p(n)$ for some polynomial p.

Fact

NP $\subseteq$ AM.

PSPACE

Space is more valuable than time!

PSPACE

Space is more valuable than time!

Definition

We say that a Turing machine T decides L in space s if T decides L and whenever T is run on an input of length at most n, it touches only $s(n)$ squares on its tape.

PSPACE

Space is more valuable than time!

Definition

We say that a Turing machine T decides L in space s if T decides L and whenever T is run on an input of length at most n, it touches only $s(n)$ squares on its tape. We say that a language L is in PSPACE if there is a polynomial p and a Turing machine deciding L in space p.

PSPACE

Space is more valuable than time!

Definition

We say that a Turing machine T decides L in space s if T decides L and whenever T is run on an input of length at most n, it touches only $s(n)$ squares on its tape. We say that a language L is in PSPACE if there is a polynomial p and a Turing machine deciding L in space p.

Theorem (PSPACE is big)

1. $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$

PSPACE

Space is more valuable than time!

Definition

We say that a Turing machine T decides L in space s if T decides L and whenever T is run on an input of length at most n, it touches only $s(n)$ squares on its tape. We say that a language L is in PSPACE if there is a polynomial p and a Turing machine deciding L in space p.

Theorem (PSPACE is big)

1. $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$
2. $\mathrm{P} \subseteq \mathrm{coNP} \subseteq \mathrm{PSPACE}$

PSPACE

Space is more valuable than time!

Definition

We say that a Turing machine T decides L in space s if T decides L and whenever T is run on an input of length at most n, it touches only $s(n)$ squares on its tape. We say that a language L is in PSPACE if there is a polynomial p and a Turing machine deciding L in space p.

Theorem (PSPACE is big)

1. $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$
2. $\mathrm{P} \subseteq \mathrm{coNP} \subseteq \mathrm{PSPACE}$
3. $\mathrm{P} \subseteq \mathrm{BPP} \subseteq \mathrm{PSPACE}$

Proof that PSPACE is big, I

Idea: Polynomial space is big enough to do *brute-force search*.

Proof of NP $\subseteq$ PSPACE.

Suppose that $L \in$ NP. Let V be a verifier and let q be a polynomial such that for $x \in L$ of length at most n, there is a witness w of length at most $q(n)$.

Proof that PSPACE is big, I

Idea: Polynomial space is big enough to do *brute-force search*.

Proof of NP $\subseteq$ PSPACE.

Suppose that $L \in \mathrm{NP}$. Let V be a verifier and let q be a polynomial such that for $x \in L$ of length at most n, there is a witness w of length at most $q(n)$.

We describe a PSPACE-algorithm deciding L.

1. Initialize $w \leftarrow 0^{q(n)}$.

Proof that PSPACE is big, I

Idea: Polynomial space is big enough to do *brute-force search*.

Proof of NP $\subseteq$ PSPACE.

Suppose that $L \in \mathrm{NP}$. Let V be a verifier and let q be a polynomial such that for $x \in L$ of length at most n, there is a witness w of length at most $q(n)$.

We describe a PSPACE-algorithm deciding L.

1. Initialize $w \leftarrow 0^{q(n)}$.
2. Try $V(x, w)$. Note whether it accepts or rejects, and then erase the memory used in the computation.

Proof that PSPACE is big, I

Idea: Polynomial space is big enough to do *brute-force search*.

Proof of NP $\subseteq$ PSPACE.

Suppose that $L \in$ NP. Let V be a verifier and let q be a polynomial such that for $x \in L$ of length at most n, there is a witness w of length at most $q(n)$.

We describe a PSPACE-algorithm deciding L.

1. Initialize $w \leftarrow 0^{q(n)}$.
2. Try $V(x, w)$. Note whether it accepts or rejects, and then erase the memory used in the computation.
3. If V accepted, halt and accept.

Proof that PSPACE is big, I

Idea: Polynomial space is big enough to do *brute-force search*.

Proof of NP $\subseteq$ PSPACE.

Suppose that $L \in$ NP. Let V be a verifier and let q be a polynomial such that for $x \in L$ of length at most n, there is a witness w of length at most $q(n)$.
We describe a PSPACE-algorithm deciding L.

1. Initialize $w \leftarrow 0^{q(n)}$.
2. Try $V(x, w)$. Note whether it accepts or rejects, and then erase the memory used in the computation.
3. If V accepted, halt and accept.
4. If V rejected and w is at the largest possible value, halt and reject.

Proof that PSPACE is big, I

Idea: Polynomial space is big enough to do *brute-force search*.

Proof of NP $\subseteq$ PSPACE.

Suppose that $L \in \mathrm{NP}$. Let V be a verifier and let q be a polynomial such that for $x \in L$ of length at most n, there is a witness w of length at most $q(n)$.
We describe a PSPACE-algorithm deciding L.

1. Initialize $w \leftarrow 0^{q(n)}$.
2. Try $V(x, w)$. Note whether it accepts or rejects, and then erase the memory used in the computation.
3. If V accepted, halt and accept.
4. If V rejected and w is at the largest possible value, halt and reject.
5. Otherwise, increment w and return to step 2.

Proof that PSPACE is big, II

Lemma

PSPACE *is closed under complement.*

Proof that PSPACE is big, II

Lemma

PSPACE *is closed under complement.*

This establishes that NP $\subseteq$ PSPACE iff coNP $\subseteq$ PSPACE.

Proof that PSPACE is big, II

Lemma

PSPACE *is closed under complement.*

This establishes that NP $\subseteq$ PSPACE iff coNP $\subseteq$ PSPACE.

Proof.

Given a PSPACE-algorithm for problem L, switch the accept and reject states. This is a PSPACE-algorithm for $\overline{L}$. $\square$

Proof that PSPACE is big, III

In our previous brute force search, we only cared about finding a single point in the search space with a specified property. PSPACE-computations can do much more than that, however.

Proof.

Proof that BPP $\subseteq$ PSPACE Let $L \in$ BPP with algorithm M such that $\Pr_r[M(x, r) accepts] \geq \frac{2}{3}$ for $x \in L$ and $\Pr_r[M(x, r) accepts] \leq \frac{1}{3}$ for $x \notin L$. We give a PSPACE-algorithm deciding M:

1. Initialize r to the all-zeroes string. Initialize counters "accept" and "reject".

Proof that PSPACE is big, III

In our previous brute force search, we only cared about finding a single point in the search space with a specified property. PSPACE-computations can do much more than that, however.

Proof.

Proof that BPP $\subseteq$ PSPACE Let $L \in$ BPP with algorithm M such that $\Pr_r[M(x,r) accepts] \geq \frac{2}{3}$ for $x \in L$ and $\Pr_r[M(x,r) accepts] \leq \frac{1}{3}$ for $x \notin L$. We give a PSPACE-algorithm deciding M:

1. Initialize r to the all-zeroes string. Initialize counters "accept" and "reject".
2. Run $M(x,r)$. If it accepts, increment the

Proof that PSPACE is big, III

In our previous brute force search, we only cared about finding a single point in the search space with a specified property. PSPACE-computations can do much more than that, however.

Proof.

Proof that BPP $\subseteq$ PSPACE Let $L \in$ BPP with algorithm M such that $\Pr_r[M(x,r) accepts] \geq \frac{2}{3}$ for $x \in L$ and $\Pr_r[M(x,r) accepts] \leq \frac{1}{3}$ for $x \notin L$. We give a PSPACE-algorithm deciding M:

1. Initialize r to the all-zeroes string. Initialize counters "accept" and "reject".
2. Run $M(x,r)$. If it accepts, increment the
3. If r is not the maximum value, increment r and return to step 2.

Proof that PSPACE is big, III

In our previous brute force search, we only cared about finding a single point in the search space with a specified property. PSPACE-computations can do much more than that, however.

Proof.

Proof that BPP $\subseteq$ PSPACE Let $L \in$ BPP with algorithm M such that $\Pr_r[M(x,r) accepts] \geq \frac{2}{3}$ for $x \in L$ and $\Pr_r[M(x,r) accepts] \leq \frac{1}{3}$ for $x \notin L$. We give a PSPACE-algorithm deciding M:

1. Initialize r to the all-zeroes string. Initialize counters "accept" and "reject".
2. Run $M(x,r)$. If it accepts, increment the
3. If r is not the maximum value, increment r and return to step 2.
4. If the accept counter is larger, halt and accept. Otherwise

Outline

1. What is Complexity Theory?
   - Models of computation
   - Complexity classes
   - Interactive Proofs
   - Closed Timelike Curves

CTCs

In 1949, Kurt Gödel proved that the equations of general relativity allow for the existence of *closed timelike curves*.

CTCs

In 1949, Kurt Gödel proved that the equations of general relativity allow for the existence of *closed timelike curves*. These are regions of spacetime where you can travel in a spatial loop and end up back at the beginning *before* you started.

CTCs

In 1949, Kurt Gödel proved that the equations of general relativity allow for the existence of *closed timelike curves*. These are regions of spacetime where you can travel in a spatial loop and end up back at the beginning *before* you started.

How can we use these to do computation?

The grandfather paradox

Here is a “proof” that interaction with CTCs is impossible.

- Travel along the CTC until you come out 50 years before you enter.

The grandfather paradox

Here is a “proof” that interaction with CTCs is impossible.

- Travel along the CTC until you come out 50 years before you enter.
- Shoot your grandparent in the head, killing them.

The grandfather paradox

Here is a “proof” that interaction with CTCs is impossible.

- Travel along the CTC until you come out 50 years before you enter.
- Shoot your grandparent in the head, killing them.
- Fail to be born.

The grandfather paradox

Here is a “proof” that interaction with CTCs is impossible.

- Travel along the CTC until you come out 50 years before you enter.
- Shoot your grandparent in the head, killing them.
- Fail to be born.
- Fail to step into the CTC.

The grandfather paradox

Here is a "proof" that interaction with CTCs is impossible.

- Travel along the CTC until you come out 50 years before you enter.
- Shoot your grandparent in the head, killing them.
- Fail to be born.
- Fail to step into the CTC.
- Contradiction!

The Markov chain model

Consider two states of the universe, corresponding to whether or not you are alive.

The Markov chain model

Consider two states of the universe, corresponding to whether or not you are alive. Let each of these states be a basis element in a two-dimensional vector space:
$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is "you are alive" and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is "you are dead".

The Markov chain model

Consider two states of the universe, corresponding to whether or not you are alive. Let each of these states be a basis element in a two-dimensional vector space:
$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is "you are alive" and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is "you are dead". Consider the "perform the shoot-my-grandparent experiment" operator which interchanges these:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The Markov chain model

Consider two states of the universe, corresponding to whether or not you are alive. Let each of these states be a basis element in a two-dimensional vector space:

$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is "you are alive" and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is "you are dead". Consider the "perform the shoot-my-grandparent experiment" operator which interchanges these:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The apparent contradiction arises because we want this experiment to *not* change the state of the world.

The Markov chain model

Consider two states of the universe, corresponding to whether or not you are alive. Let each of these states be a basis element in a two-dimensional vector space:
$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is "you are alive" and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is "you are dead". Consider the "perform the shoot-my-grandparent experiment" operator which interchanges these:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The apparent contradiction arises because we want this experiment to *not* change the state of the world.
This contradiction is easily resolved: set the state of the world as $\frac{1}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$!

Stable distributions of Markov chains

Say that a finite-dimensional matrix A is a Markov chain if:

- It takes probability distributions to probability distributions.

Stable distributions of Markov chains

Say that a finite-dimensional matrix A is a Markov chain if:

- It takes probability distributions to probability distributions.
- It is irreducible, i.e. it cannot be written in block diagonal form with more than 1 block.

Stable distributions of Markov chains

Say that a finite-dimensional matrix A is a Markov chain if:

- It takes probability distributions to probability distributions.
- It is irreducible, i.e. it cannot be written in block diagonal form with more than 1 block.

Stable distributions of Markov chains

Say that a finite-dimensional matrix A is a Markov chain if:

- It takes probability distributions to probability distributions.
- It is irreducible, i.e. it cannot be written in block diagonal form with more than 1 block.

Theorem

If A is a Markov chain, then A has a unique $+1$-eigenvalue eigenvector, called its stable distribution.

Stable distributions of Markov chains

Say that a finite-dimensional matrix A is a Markov chain if:

- It takes probability distributions to probability distributions.
- It is irreducible, i.e. it cannot be written in block diagonal form with more than 1 block.

Theorem

If A is a Markov chain, then A has a unique $+1$-eigenvalue eigenvector, called its stable distribution. Furthermore, it has the following explicit form:

$$v = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} A^i v_0, \tag{1}$$

for any starting distribution v_0.

A $\mathrm{BPP}_{\mathrm{CTC}}$ algorithm for NP

Definition

A language L is in $\mathrm{BPP}_{\mathrm{CTC}}$ if it can be decided in polynomial time by a randomized algorithm which can find stable distributions of implicitly-defined Markov chains as a unit time operation.

A $\mathrm{BPP}_{\mathrm{CTC}}$ algorithm for NP

Definition

A language L is in $\mathrm{BPP}_{\mathrm{CTC}}$ if it can be decided in polynomial time by a randomized algorithm which can find stable distributions of implicitly-defined Markov chains as a unit time operation.

Markov chains can encode brute force searches in much the same way as PSPACE algorithms.

A $\mathrm{BPP}_{\mathrm{CTC}}$ algorithm for NP

Definition

A language L is in $\mathrm{BPP}_{\mathrm{CTC}}$ if it can be decided in polynomial time by a randomized algorithm which can find stable distributions of implicitly-defined Markov chains as a unit time operation.

Markov chains can encode brute force searches in much the same way as PSPACE algorithms. Suppose we have $L \in \mathrm{NP}$ with verifier V with an input of length n. Let there be one basis state $|w\rangle$ for each possible witness w, along with one extra "accept basis state" $|\mathrm{accept}\rangle$.

A $\mathrm{BPP}_{\mathrm{CTC}}$ algorithm for NP

Definition

A language L is in $\mathrm{BPP}_{\mathrm{CTC}}$ if it can be decided in polynomial time by a randomized algorithm which can find stable distributions of implicitly-defined Markov chains as a unit time operation.

Markov chains can encode brute force searches in much the same way as PSPACE algorithms. Suppose we have $L \in \mathrm{NP}$ with verifier V with an input of length n. Let there be one basis state $|w\rangle$ for each possible witness w, along with one extra "accept basis state" $|\text{accept}\rangle$.

Let $A|w\rangle = |w+1\rangle$ if $V(x,w)$ rejects and $A|w\rangle = |\text{accept}\rangle$ if $V(x,w)$ accepts. Let $A|\text{accept}\rangle = |\text{accept}\rangle$.

Wizards = Time Machines

$$QIP \overset{\text{[Jai+10]}}{=} IP$$

Wizards = Time Machines

$$QIP \overset{\text{[Jai+10]}}{=} IP \overset{\text{[Sha92]}}{=} PSPACE$$

Wizards = Time Machines

$$QIP \overset{\text{[Jai+10]}}{=} IP \overset{\text{[Sha92]}}{=} PSPACE \overset{\text{[AW09]}}{=} BPP_{CTC} = BQP_{CTC} \tag{2}$$

Bibliography I

Scott Aaronson and John Watrous. "Closed timelike curves make quantum and classical computing equivalent". In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*. Vol. 465. 2102. The Royal Society. 2009, pp. 631–647.

Rahul Jain et al. "Qip= pspace". In: *Communications of the ACM* 53.12 (2010), pp. 102–109.

Adi Shamir. "Ip= pspace". In: *Journal of the ACM (JACM)* 39.4 (1992), pp. 869–877.