# Approximating Kolmogorov complexity function

Ruslan Ishkuvatov
(joint work with Daniil Musatov, CiE 2019)

# Kolmogorov complexity

- $C(x)$ = the length of the shortest program $p$ producing $x$.

# Kolmogorov complexity

- $C(x) =$ the length of the shortest program $p$ producing $x$.
- Depends on programming language:

$C_U(x) = \min\{|p| : U(p) = x\}$ where $U$ is the interpreter (decompressor, description mode,...)

# Kolmogorov complexity

- $C(x)$ = the length of the shortest program $p$ producing $x$.
- Depends on programming language:

$C_U(x) = \min\{|p| : U(p) = x\}$ where $U$ is the interpreter (decompressor, description mode,...)

- (Solomonoff – Kolmogorov theorem) There is an *optimal* $U$ that makes $C_U$ minimal up to $O(1)$ additive term: for any other method $U'$ there exists $c$ such that

$$C_U(x) \leqslant C_{U'}(x) + c$$

for all $x$.

# Kolmogorov complexity

- $C(x)$ = the length of the shortest program $p$ producing $x$.
- Depends on programming language:

$C_U(x) = \min\{|p| \colon U(p) = x\}$ where $U$ is the interpreter (decompressor, description mode, …)

- (Solomonoff – Kolmogorov theorem) There is an *optimal* $U$ that makes $C_U$ minimal up to $O(1)$ additive term: for any other method $U'$ there exists $c$ such that

$$C_U(x) \leqslant C_{U'}(x) + c$$

for all $x$.

**Definition**

Fix some optimal $U$ and call $C_U(x)$ the *Kolmogorov complexity* of a string $x$. Notation: $C(x)$.

# Conditional complexity

- $C(x|y)$ = conditional complexity of $x$ given $y$

# Conditional complexity

- $C(x|y)$ = conditional complexity of $x$ given $y$
- the minimal length of a program that produces $x$ given $y$

# Conditional complexity

- $C(x|y)$ = conditional complexity of $x$ given $y$
- the minimal length of a program that produces $x$ given $y$
- again there exists an optimal programming language that makes $C(x|y)$ minimal up to an additive $O(1)$-term

# Conditional complexity

- $C(x|y)$ = conditional complexity of $x$ given $y$
- the minimal length of a program that produces $x$ given $y$
- again there exists an optimal programming language that makes $C(x|y)$ minimal up to an additive $O(1)$-term
- $C(x)$: amount of information in $x$

# Conditional complexity

- $C(x|y)$ = conditional complexity of $x$ given $y$
- the minimal length of a program that produces $x$ given $y$
- again there exists an optimal programming language that makes $C(x|y)$ minimal up to an additive $O(1)$-term
- $C(x)$: amount of information in $x$
- $C(x|y)$: amount of information in $x$ missing in $y$

# Basic properties of Kolmogorov complexity

# Basic properties of Kolmogorov complexity

- $C(x) \leqslant |x| + O(1)$: compare the optimal decompressor with a trivial one

# Basic properties of Kolmogorov complexity

- $C(x) \leqslant |x| + O(1)$: compare the optimal decompressor with a trivial one
- for every $k \in \mathbb{N}$ there are at most $2^k - 1$ strings of Kolmogorov complexity less than $k$ (not enough short programs).

# Basic properties of Kolmogorov complexity

- $C(x) \leqslant |x| + O(1)$: compare the optimal decompressor with a trivial one
- for every $k \in \mathbb{N}$ there are at most $2^k - 1$ strings of Kolmogorov complexity less than $k$ (not enough short programs).
- most strings of length $n$ have complexity close to $n$ ($2^{-d}$-fraction of strings can be compressed by $d$ bits)

# Basic properties of Kolmogorov complexity

- $C(x) \leqslant |x| + O(1)$: compare the optimal decompressor with a trivial one
- for every $k \in \mathbb{N}$ there are at most $2^k - 1$ strings of Kolmogorov complexity less than $k$ (not enough short programs).
- most strings of length $n$ have complexity close to $n$ ($2^{-d}$-fraction of strings can be compressed by $d$ bits)
- Algorithmic transformation does not increase complexity: $C(A(x)) \leqslant C(x) + O(1)$ for computable $A$

# Basic properties of Kolmogorov complexity

- $C(x) \leqslant |x| + O(1)$: compare the optimal decompressor with a trivial one
- for every $k \in \mathbb{N}$ there are at most $2^k - 1$ strings of Kolmogorov complexity less than $k$ (not enough short programs).
- most strings of length $n$ have complexity close to $n$ ($2^{-d}$-fraction of strings can be compressed by $d$ bits)
- Algorithmic transformation does not increase complexity: $C(A(x)) \leqslant C(x) + O(1)$ for computable $A$
- Kolmogorov complexity is not computable and does not have nontrivial computable lower bounds

# Basic properties of Kolmogorov complexity

- $C(x) \leqslant |x| + O(1)$: compare the optimal decompressor with a trivial one
- for every $k \in \mathbb{N}$ there are at most $2^k - 1$ strings of Kolmogorov complexity less than $k$ (not enough short programs).
- most strings of length $n$ have complexity close to $n$ ($2^{-d}$-fraction of strings can be compressed by $d$ bits)
- Algorithmic transformation does not increase complexity: $C(A(x)) \leqslant C(x) + O(1)$ for computable $A$
- Kolmogorov complexity is not computable and does not have nontrivial computable lower bounds
- "the minimal string of complexity greater than N" (Berry)

# Approximating complexity for most inputs

# Approximating complexity for most inputs

- $(d, e)$-approximation: $|\tilde{C}(x) - C(x)| < d$ for all $x$ of length $n$ except $\varepsilon = 2^{-e}$ fraction

# Approximating complexity for most inputs

- $(d, e)$-approximation: $|\tilde{C}(x) - C(x)| < d$ for all $x$ of length $n$ except $\varepsilon = 2^{-e}$ fraction
- $\tilde{C}(x) = |x|$ is a $(d, d)$-approximation

# Approximating complexity for most inputs

- $(d, e)$-approximation: $|\tilde{C}(x) - C(x)| < d$ for all $x$ of length $n$ except $\varepsilon = 2^{-e}$ fraction
- $\tilde{C}(x) = |x|$ is a $(d, d)$-approximation
- cheap trick: $\tilde{C}(x) = |x| - d$ is a $(d, 2d)$-approximation

# Approximating complexity for most inputs

- $(d, e)$-approximation: $|\tilde{C}(x) - C(x)| < d$ for all $x$ of length $n$ except $\varepsilon = 2^{-e}$ fraction
- $\tilde{C}(x) = |x|$ is a $(d, d)$-approximation
- cheap trick: $\tilde{C}(x) = |x| - d$ is a $(d, 2d)$-approximation
- disclaimer: $O(1)$ is omitted everywhere

**Theorem**

*This is essentially optimal: no significantly better computable approximation exists.*

# Formal statement (uniform setting)

# Formal statement (uniform setting)

- Let $d(n)$ and $e(n)$ be computable functions

# Formal statement (uniform setting)

- Let $d(n)$ and $e(n)$ be computable functions
- Let $\tilde{C}(x)$ be a computable function on strings of all lengths that for every $n$ is a $(d(n), e(n))$-approximation of $C(x)$ for $n$-bit strings.

**Theorem**

$e(n) \leqslant 2d(n) + O(1)$

# Mass problems: example

# Mass problems: example

**Theorem (A)**

*Kolmogorov complexity is not computable.*

# Mass problems: example

**Theorem (A)**

*Kolmogorov complexity is not computable.*

**Theorem (B)**

*Given an oracle ("external procedure") that computes Kolmogorov complexity function, one can solve the halting problem.*

# Mass problems: example

**Theorem (A)**

*Kolmogorov complexity is not computable.*

**Theorem (B)**

*Given an oracle ("external procedure") that computes Kolmogorov complexity function, one can solve the halting problem.*

obviously B implies A, but in general B is a stronger statement

# Mass problems: warning

# Mass problems: warning

**Theorem (A)**

*Nontrivial lower bounds for Kolmogorov complexity are not computable.*

# Mass problems: warning

**Theorem (A)**

*Nontrivial lower bounds for Kolmogorov complexity are not computable.*

**Theorem (B)**

*(FALSE!) Given an oracle that computes a nontrivial lower bound for Kolmogorov complexity, one can solve the halting problem.*

# Mass problems: warning

**Theorem (A)**

*Nontrivial lower bounds for Kolmogorov complexity are not computable.*

**Theorem (B)**

*(FALSE!) Given an oracle that computes a nontrivial lower bound for Kolmogorov complexity, one can solve the halting problem.*

still some results about Kolmogorov complexity are true in (B)-version

# Approximating $C$ is as difficult as computing it

# Approximating $C$ is as difficult as computing it

**Theorem**

*There is a machine that, given an oracle that computes $C$ up to factor $100$, computes $C$ exactly*

# Approximating $C$ is as difficult as computing it

**Theorem**

*There is a machine that, given an oracle that computes $C$ up to factor $100$, computes $C$ exactly*

Note: exact computation of $C$ is equivalent to solving halting problem

# Approximating $C$ is as difficult as computing it

**Theorem**

*There is a machine that, given an oracle that computes $C$ up to factor 100, computes $C$ exactly*

Note: exact computation of $C$ is equivalent to solving halting problem

**Theorem**

*Let $d(n)$ and $e(n)$ be two computable functions such that $e(n) - 2d(n) \to \infty$. Let $\tilde{C}(x)$ be a function on strings that $(d(n), e(n))$-approximates $C(x)$ for every $n$. Then there is a machine that computes $C$ using $\tilde{C}$ as an oracle.*

# Approximating $C$ is as difficult as computing it

**Theorem**

*There is a machine that, given an oracle that computes $C$ up to factor 100, computes $C$ exactly*

Note: exact computation of $C$ is equivalent to solving halting problem

**Theorem**

*Let $d(n)$ and $e(n)$ be two computable functions such that $e(n) - 2d(n) \to \infty$. Let $\tilde{C}(x)$ be a function on strings that $(d(n), e(n))$-approximates $C(x)$ for every $n$. Then there is a machine that computes $C$ using $\tilde{C}$ as an oracle.*

Note: the machine does not depend on $\tilde{C}$, only on $d$ and $e$.

# Finite case

**Theorem**

*If $e - 2d$ is large, then every (finite) function that $(d, e)$-approximates $C(x)$ for $n$-bit inputs has high complexity*

# Finite case

**Theorem**
*If $e - 2d$ is large, then every (finite) function that $(d, e)$-approximates $C(x)$ for $n$-bit inputs has high complexity*

Why is this enough? If $\tilde{C}$ is a computable approximation, then its restriction on $n$-bit strings cannot have high complexity.

# Finite case

**Theorem**

*If $e - 2d$ is large, then every (finite) function that $(d, e)$-approximates $C(x)$ for $n$-bit inputs has high complexity*

Why is this enough? If $\tilde{C}$ is a computable approximation, then its restriction on $n$-bit strings cannot have high complexity.

Proof idea: if $\tilde{C}$ approximates $C$, then one can construct a complex object knowing $\tilde{C}$ and few bits of advice

# More details

## More details

Let $x_1, \ldots, x_{2^n}$ be an enumeration of $n$-bit strings in $\tilde{C}$-descending order. Among the first $2^{n-e}$ of them there is a string $\hat{x}$ such that $|\tilde{C}(\hat{x}) - C(\hat{x})| < d$

# More details

Let $x_1, \ldots, x_{2^n}$ be an enumeration of $n$-bit strings in $\tilde{C}$-descending order. Among the first $2^{n-e}$ of them there is a string $\hat{x}$ such that $|\tilde{C}(\hat{x}) - C(\hat{x})| < d$

In particular, $C(\hat{x}) > \tilde{C}(\hat{x}) - d$

## More details

Let $x_1, \ldots, x_{2^n}$ be an enumeration of $n$-bit strings in $\tilde{C}$-descending order. Among the first $2^{n-e}$ of them there is a string $\hat{x}$ such that $|\tilde{C}(\hat{x}) - C(\hat{x})| < d$

In particular, $C(\hat{x}) > \tilde{C}(\hat{x}) - d$

Assuming $e \geqslant 2$, for at least $\frac{3}{4}$ of all $n$-bit strings their $\tilde{C}$-values are close to their complexity, and at least a half of the strings have complexity at least $n - 1$. Thus for at least a quarter of the strings their $\tilde{C}$-value is at least $n - d$.

## More details

Let $x_1, \ldots, x_{2^n}$ be an enumeration of $n$-bit strings in $\tilde{C}$-descending order. Among the first $2^{n-e}$ of them there is a string $\hat{x}$ such that $|\tilde{C}(\hat{x}) - C(\hat{x})| < d$

In particular, $C(\hat{x}) > \tilde{C}(\hat{x}) - d$

Assuming $e \geqslant 2$, for at least $\frac{3}{4}$ of all $n$-bit strings their $\tilde{C}$-values are close to their complexity, and at least a half of the strings have complexity at least $n-1$. Thus for at least a quarter of the strings their $\tilde{C}$-value is at least $n-d$.

Since $\hat{x}$ is taken from $2^{n-e}$ strings with the biggest $\tilde{C}$-value, $\tilde{C}(\hat{x}) \geqslant n-d$ and $C(\hat{x}) > n-2d$.

# More details

Let $x_1, \ldots, x_{2^n}$ be an enumeration of $n$-bit strings in $\tilde{C}$-descending order. Among the first $2^{n-e}$ of them there is a string $\hat{x}$ such that $|\tilde{C}(\hat{x}) - C(\hat{x})| < d$

In particular, $C(\hat{x}) > \tilde{C}(\hat{x}) - d$

Assuming $e \geqslant 2$, for at least $\frac{3}{4}$ of all $n$-bit strings their $\tilde{C}$-values are close to their complexity, and at least a half of the strings have complexity at least $n-1$. Thus for at least a quarter of the strings their $\tilde{C}$-value is at least $n-d$.

Since $\hat{x}$ is taken from $2^{n-e}$ strings with the biggest $\tilde{C}$-value, $\tilde{C}(\hat{x}) \geqslant n-d$ and $C(\hat{x}) > n-2d$.

Given $\tilde{C}$, the string $\hat{x}$ can be encoded by its index in enumeration: $C(\hat{x}|\tilde{C}) < n - e + O(1)$.

# More details

Let $x_1, \ldots, x_{2^n}$ be an enumeration of $n$-bit strings in $\tilde{C}$-descending order. Among the first $2^{n-e}$ of them there is a string $\hat{x}$ such that $|\tilde{C}(\hat{x}) - C(\hat{x})| < d$

In particular, $C(\hat{x}) > \tilde{C}(\hat{x}) - d$

Assuming $e \geqslant 2$, for at least $\frac{3}{4}$ of all $n$-bit strings their $\tilde{C}$-values are close to their complexity, and at least a half of the strings have complexity at least $n-1$. Thus for at least a quarter of the strings their $\tilde{C}$-value is at least $n-d$.

Since $\hat{x}$ is taken from $2^{n-e}$ strings with the biggest $\tilde{C}$-value, $\tilde{C}(\hat{x}) \geqslant n - d$ and $C(\hat{x}) > n - 2d$.

Given $\tilde{C}$, the string $\hat{x}$ can be encoded by its index in enumeration: $C(\hat{x}|\tilde{C}) < n - e + O(1)$.

these inequalities imply $C(\tilde{C}) > e - 2d + O(\log n)$.

# How to prove oracle results

# How to prove oracle results

- Main tool: busy beaver numbers

# How to prove oracle results

- Main tool: busy beaver numbers
- $BB(n)$ = the maximal time needed by programs of size at most $n$

# How to prove oracle results

- Main tool: busy beaver numbers
- $BB(n)$ = the maximal time needed by programs of size at most $n$
- $B(n)$ = the maximal number of complexity at most $n$

# How to prove oracle results

- Main tool: busy beaver numbers
- $BB(n) =$ the maximal time needed by programs of size at most $n$
- $B(n) =$ the maximal number of complexity at most $n$
- $B(n) \approx BB(n \pm O(1))$

# How to prove oracle results

- Main tool: busy beaver numbers
- $BB(n)$ = the maximal time needed by programs of size at most $n$
- $B(n)$ = the maximal number of complexity at most $n$
- $B(n) \approx BB(n \pm O(1))$
- if we know some approximation for $\tilde{C}$, we wait until $C^T$ (Kolmogorov complexity with time bound $T$) becomes compatible with this approximation

# How to prove oracle results

- Main tool: busy beaver numbers
- $BB(n)$ = the maximal time needed by programs of size at most $n$
- $B(n)$ = the maximal number of complexity at most $n$
- $B(n) \approx BB(n \pm O(1))$
- if we know some approximation for $\tilde{C}$, we wait until $C^T$ (Kolmogorov complexity with time bound $T$) becomes compatible with this approximation
- this $T$ is larger than some busy beaver number since all numbers greater than $T$ have large complexity (finite case), and $B \approx BB$

## How to prove oracle results

- Main tool: busy beaver numbers
- $BB(n)$ = the maximal time needed by programs of size at most $n$
- $B(n)$ = the maximal number of complexity at most $n$
- $B(n) \approx BB(n \pm O(1))$
- if we know some approximation for $\tilde{C}$, we wait until $C^T$ (Kolmogorov complexity with time bound $T$) becomes compatible with this approximation
- this $T$ is larger than some busy beaver number since all numbers greater than $T$ have large complexity (finite case), and $B \approx BB$
- knowing busy beaver number, we know which programs halt / do not halt

# How to prove oracle results

- Main tool: busy beaver numbers
- $BB(n)$ = the maximal time needed by programs of size at most $n$
- $B(n)$ = the maximal number of complexity at most $n$
- $B(n) \approx BB(n \pm O(1))$
- if we know some approximation for $\tilde{C}$, we wait until $C^T$ (Kolmogorov complexity with time bound $T$) becomes compatible with this approximation
- this $T$ is larger than some busy beaver number since all numbers greater than $T$ have large complexity (finite case), and $B \approx BB$
- knowing busy beaver number, we know which programs halt / do not halt
- ...and can compute $C$ exactly