Stopping time complexity and monotone-conditional complexity

Mikhail Andreev, Gleb Posobin, Alexander Shen

IPONWEB (Berlin), HSE (Moscow), LIRMM (Montpellier)

MFCS 2018

Kolmogorov complexity

Kolmogorov complexity

- "amount of information" in a binary string x

Kolmogorov complexity

- "amount of information" in a binary string x
- (Shannon information theory: amount of information in a random variable)

Kolmogorov complexity

- "amount of information" in a binary string x
- (Shannon information theory: amount of information in a random variable)
- complexity $C(x)$: "the minimal length of a description of x", "compressed size"

Kolmogorov complexity

- "amount of information" in a binary string x
- (Shannon information theory: amount of information in a random variable)
- complexity $C(x)$: "the minimal length of a description of x", "compressed size"
- $=$ the minimal length of a program that produces x

Kolmogorov complexity

- "amount of information" in a binary string x
- (Shannon information theory: amount of information in a random variable)
- complexity $C(x)$: "the minimal length of a description of x", "compressed size"
- $=$ the minimal length of a program that produces x
- depends on the description mode (programming language)

Kolmogorov complexity

- "amount of information" in a binary string x
- (Shannon information theory: amount of information in a random variable)
- complexity $C(x)$: "the minimal length of a description of x", "compressed size"
- $=$ the minimal length of a program that produces x
- depends on the description mode (programming language)
- choose and fix an optimal one

Vovk & Pavlovich idea

Vovk & Pavlovich idea

- How to tell which exit on a road one should take?

Vovk & Pavlovich idea

- How to tell which exit on a road one should take?
- "Nth exit": $\log N$ bits of information

Vovk & Pavlovich idea

- How to tell which exit on a road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits

Vovk & Pavlovich idea

- How to tell which exit on a road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits
- "Last exit before the brigde": not allowed

Vovk & Pavlovich idea

- How to tell which exit on a road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits
- "Last exit before the brigde": not allowed
- you get a sequence of bits (one at a time) and decide when to stop

Vovk & Pavlovich idea

- How to tell which exit on a road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits
- "Last exit before the brigde": not allowed
- you get a sequence of bits (one at a time) and decide when to stop
- TM: input one-directional read-only tape

Vovk & Pavlovich idea

- How to tell which exit on a road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits
- "Last exit before the brigde": not allowed
- you get a sequence of bits (one at a time) and decide when to stop
- TM: input one-directional read-only tape
- stopping time complexity of x = the minimal plain complexity of a TM that stops after reading input x (not seeing the next bit)

Prefix-free characterization

Prefix-free characterization

- stopping time complexity = the minimal complexity of a program that enumerates a prefix-free set of strings containing x

Prefix-free characterization

- stopping time complexity = the minimal complexity of a program that enumerates a prefix-free set of strings containing x
- M: machine with one-directional input tape

Prefix-free characterization

- stopping time complexity = the minimal complexity of a program that enumerates a prefix-free set of strings containing x
- M: machine with one-directional input tape
- domain of M: an enumerable prefix-free set (obvious)

Prefix-free characterization

- stopping time complexity = the minimal complexity of a program that enumerates a prefix-free set of strings containing x
- M: machine with one-directional input tape
- domain of M: an enumerable prefix-free set (obvious)
- Every enumerable prefix-free set M is a domain of a machine with one-directional input tape (less obvious)

Prefix-free characterization

- stopping time complexity = the minimal complexity of a program that enumerates a prefix-free set of strings containing x
- M: machine with one-directional input tape
- domain of M: an enumerable prefix-free set (obvious)
- Every enumerable prefix-free set M is a domain of a machine with one-directional input tape (less obvious)
- sketch: read the next bit only when you know that some proper extension is in M

Question

Is there a machine-free characterization for sets of pairs that are domains of machines with two one-directional input tapes ("twice prefix free machines")?

decompressor: descriptions $\rightarrow$ objects

The classification of complexities

decompressor: descriptions $\rightarrow$ objects

different "topologies" on descriptions and objects

The classification of complexities

decompressor: descriptions $\rightarrow$ objects

different "topologies" on descriptions and objects

	isolated descriptions	descriptions as prefixes
isolated objects	plain complexity	prefix complexity
objects as prefixes	decision complexity	monotone complexity

The classification of complexities

decompressor: descriptions $\rightarrow$ objects

different "topologies" on descriptions and objects

	isolated descriptions	descriptions as prefixes
isolated objects	plain complexity	prefix complexity
objects as prefixes	decision complexity	monotone complexity

decompressor: descriptions $\times$ conditions $\rightarrow$ objects

8 versions of conditional complexities

The classification of complexities

decompressor: descriptions $\rightarrow$ objects

different "topologies" on descriptions and objects

	isolated descriptions	descriptions as prefixes
isolated objects	plain complexity	prefix complexity
objects as prefixes	decision complexity	monotone complexity

decompressor: descriptions $\times$ conditions $\rightarrow$ objects

8 versions of conditional complexities

stopping time complexity of $x = C(x|x*)$

decompressor: descriptions $\rightarrow$ objects

different "topologies" on descriptions and objects

	isolated descriptions	descriptions as prefixes
isolated objects	plain complexity	prefix complexity
objects as prefixes	decision complexity	monotone complexity

decompressor: descriptions $\times$ conditions $\rightarrow$ objects

8 versions of conditional complexities

stopping time complexity of $x = C(x|x*)$

objects: isolated; descriptions: isolated;

conditions: prefixes (condition $x*$)

The "monotone-conditional" complexity $C(y|x*)$

The "monotone-conditional" complexity $C(y|x*)$

- $D(p, x)$: partial computable function (conditional decompressor)

The “monotone-conditional” complexity $C(y|x*)$

- $D(p,x)$: partial computable function (conditional decompressor)
- $C_D(y|x*) = \min\{|p|: D(p,x) = y\}$ (x is a condition)

The "monotone-conditional" complexity $C(y|x*)$

- $D(p, x)$: partial computable function (conditional decompressor)
- $C_D(y|x*) = \min\{|p| : D(p, x) = y\}$ (x is a condition)
- but D is required to be monotone ('prefix-stable') *with respect to condition*:

The “monotone-conditional” complexity $C(y|x*)$

- $D(p, x)$: partial computable function (conditional decompressor)
- $C_D(y|x*) = \min\{|p| \colon D(p, x) = y\}$ (x is a condition)
- but D is required to be monotone (‘prefix-stable’) *with respect to condition*:
- if $D(p, x) = y$, then $D(p, xz) = y$ for every z

The "monotone-conditional" complexity $C(y|x*)$

- $D(p, x)$: partial computable function (conditional decompressor)
- $C_D(y|x*) = \min\{|p| : D(p, x) = y\}$ (x is a condition)
- but D is required to be monotone ('prefix-stable') *with respect to condition*:
- if $D(p, x) = y$, then $D(p, xz) = y$ for every z
- $C(y|x*)$ = the minimal plain complexity of a prefix-stable program that maps x to y

The "monotone-conditional" complexity $C(y|x*)$

- $D(p, x)$: partial computable function (conditional decompressor)
- $C_D(y|x*) = \min\{|p| : D(p, x) = y\}$ (x is a condition)
- but D is required to be monotone ('prefix-stable') *with respect to condition*:
- if $D(p, x) = y$, then $D(p, xz) = y$ for every z
- $C(y|x*) =$ the minimal plain complexity of a prefix-stable program that maps x to y
- $C(x|x*)$ is not $O(1)$ anymore

The "monotone-conditional" complexity $C(y|x*)$

- $D(p,x)$: partial computable function (conditional decompressor)
- $C_D(y|x*) = \min\{|p| : D(p,x) = y\}$ (x is a condition)
- but D is required to be monotone ('prefix-stable') *with respect to condition*:
- if $D(p,x) = y$, then $D(p,xz) = y$ for every z
- $C(y|x*)$ = the minimal plain complexity of a prefix-stable program that maps x to y
- $C(x|x*)$ is not $O(1)$ anymore
- $C(x|x*)$ = (plain) stopping time complexity

A quantitative characterization of complexity

A quantitative characterization of complexity

- How to define $C(x)$ not mentioning descriptions/programs?

- How to define $C(x)$ not mentioning descriptions/programs?
- $C(x)$ is upper semicomputable;

- How to define $C(x)$ not mentioning descriptions/programs?
- $C(x)$ is upper semicomputable;
- $\#\{x\colon C(x) < n\} < 2^n$ for all n;

- How to define $C(x)$ not mentioning descriptions/programs?
- $C(x)$ is upper semicomputable;
- $\#\{x\colon C(x) < n\} < 2^n$ for all n;
- $C(\cdot)$ is the minimal function with these properties

Similar characterization of stopping time complexity

Similar characterization of stopping time complexity

- Stopping time complexity: also upper semicomputable

Similar characterization of stopping time complexity

- Stopping time complexity: also upper semicomputable
- how many simple strings? now all 0^n1 have complexity $O(1)$

Similar characterization of stopping time complexity

- Stopping time complexity: also upper semicomputable
- how many simple strings? now all 0^n1 have complexity $O(1)$
- but for every path α in the binary tree and for every n there are less than 2^n strings *on this path* with $C(x|x*) < n$.

Similar characterization of stopping time complexity

- Stopping time complexity: also upper semicomputable
- how many simple strings? now all 0^n1 have complexity $O(1)$
- but for every path α in the binary tree and for every n there are less than 2^n strings *on this path* with $C(x|x*) < n$.
- Stopping time complexity is the minimal function in this class.

Similar characterization of stopping time complexity

- Stopping time complexity: also upper semicomputable
- how many simple strings? now all 0^n1 have complexity $O(1)$
- but for every path α in the binary tree and for every n there are less than 2^n strings *on this path* with $C(x|x*) < n$.
- Stopping time complexity is the minimal function in this class.
- less obvious (Vovk, Pavlovich)

An oracle characterization

An oracle characterization

- an oracle (bit sequence X) can be added to everything in recursion theory (general computability theory)

An oracle characterization

- an oracle (bit sequence X) can be added to everything in recursion theory (general computability theory)
- $C^X(x)$: complexity of x if decompressor has free access to X

An oracle characterization

- an oracle (bit sequence X) can be added to everything in recursion theory (general computability theory)
- $\mathrm{C}^X(x)$: complexity of x if decompressor has free access to X
- $\mathrm{C}^X(x) \le \mathrm{C}(x|x*) + O(1)$ for every extension X of x

An oracle characterization

- an oracle (bit sequence X) can be added to everything in recursion theory (general computability theory)
- $C^X(x)$: complexity of x if decompressor has free access to X
- $C^X(x) \le C(x|x*) + O(1)$ for every extension X of x
- $C(x|x*) = \max\{C^X(x)\colon X \text{ is an extension of } x\}$

Five equivalent definitions of stopping time complexity

Five equivalent definitions of stopping time complexity

- minimal complexity of a Turing machine with one-directional input tape that stops at x

Five equivalent definitions of stopping time complexity

- minimal complexity of a Turing machine with one-directional input tape that stops at x
- minimal complexity of enumeration a prefix-free set containing x

Five equivalent definitions of stopping time complexity

- minimal complexity of a Turing machine with one-directional input tape that stops at x
- minimal complexity of enumeration a prefix-free set containing x
- $C(x|x*)$

Five equivalent definitions of stopping time complexity

- minimal complexity of a Turing machine with one-directional input tape that stops at x
- minimal complexity of enumeration a prefix-free set containing x
- $C(x|x*)$
- minimal function in the class of lower semicomputable functions K such that no more than 2^n prefixes of any sequence have complexity at most n

Five equivalent definitions of stopping time complexity

- minimal complexity of a Turing machine with one-directional input tape that stops at x
- minimal complexity of enumeration a prefix-free set containing x
- $C(x|x*)$
- minimal function in the class of lower semicomputable functions K such that no more than 2^n prefixes of any sequence have complexity at most n
- maximal$C^X(x)$ for all extensions X of x

What is not true

What is not true

- $C(x|y*)$ is *not* the minimal complexity of a prefix-free function that maps some prefix of y to x

What is not true

- $C(x|y*)$ is *not* the minimal complexity of a prefix-free function that maps some prefix of y to x
- $C(x|y*)$ does *not* have the natural quantitative characterization as a monotone over y function $[C(x|y0*) \le C(x|y*), C(x|y1*) \le C(x|y*)]$ such that for every y and n there are at most 2^n objects x such that $C(x|y*) < n$; the difference may be by a factor of 2 (but not more)

Thanks!

Prefix stopping time complexity

- $K(x|y*)$: the decompressor is monotone (prefix-stable) w.r.t. both arguments.

Prefix stopping time complexity

- $K(x|y*)$: the decompressor is monotone (prefix-stable) w.r.t. both arguments.
- conditions and programs are prefixes, objects are isolated

Prefix stopping time complexity

- $K(x|y*)$: the decompressor is monotone (prefix-stable) w.r.t. both arguments.
- conditions and programs are prefixes, objects are isolated
- Why should we bother?

Prefix stopping time complexity

- $K(x|y*)$: the decompressor is monotone (prefix-stable) w.r.t. both arguments.
- conditions and programs are prefixes, objects are isolated
- Why should we bother?
- Vovk and Pavlovich tried to define this version

Prefix stopping time complexity

- $K(x|y*)$: the decompressor is monotone (prefix-stable) w.r.t. both arguments.
- conditions and programs are prefixes, objects are isolated
- Why should we bother?
- Vovk and Pavlovich tried to define this version
- separates many things that coincide for prefix complexity

Prefix (stopping time) complexity: different definitions

Prefix (stopping time) complexity: different definitions

- the length of the shortest prefix-stable program

Prefix (stopping time) complexity: different definitions

- the length of the shortest prefix-stable program
- minus logarithm of the a priori probability

Prefix (stopping time) complexity: different definitions

- the length of the shortest prefix-stable program
- minus logarithm of the a priori probability
- minus logarithm of the maximal lower semicomputable semimeasure

Prefix (stopping time) complexity: different definitions

- the length of the shortest prefix-stable program
- minus logarithm of the a priori probability
- minus logarithm of the maximal lower semicomputable semimeasure

Now:

Prefix (stopping time) complexity: different definitions

- the length of the shortest prefix-stable program
- minus logarithm of the a priori probability
- minus logarithm of the maximal lower semicomputable semimeasure

Now:

- minimal prefix complexity of a prefix-free set containing x [Vovk-Pavlovic]

Prefix (stopping time) complexity: different definitions

- the length of the shortest prefix-stable program
- minus logarithm of the a priori probability
- minus logarithm of the maximal lower semicomputable semimeasure

Now:

- minimal prefix complexity of a prefix-free set containing x [Vovk-Pavlovic]
- $> K(x|x*)$

Prefix (stopping time) complexity: different definitions

- the length of the shortest prefix-stable program
- minus logarithm of the a priori probability
- minus logarithm of the maximal lower semicomputable semimeasure

Now:

- minimal prefix complexity of a prefix-free set containing x [Vovk-Pavlovic]
- $> K(x|x*)$
- $>$ minus logarithm of the a priori probability (probability for the universal probabilistic machine to stop at x) [Andreev]

- the length of the shortest prefix-stable program
- minus logarithm of the a priori probability
- minus logarithm of the maximal lower semicomputable semimeasure

Now:

- minimal prefix complexity of a prefix-free set containing x [Vovk-Pavlovic]
- $> K(x|x*)$
- $>$ minus logarithm of the a priori probability (probability for the universal probabilistic machine to stop at x) [Andreev]
- $=$ minus logarithm of the maximal lower semicomputable function $m(x)$ whose sum along

Monotone-conditional prefix complexity

- Even more splitting...

Monotone-conditional prefix complexity

- Even more splitting...
- A priori probability: random program (for the universal decompressor) maps y to x

Monotone-conditional prefix complexity

- Even more splitting…
- A priori probability: random program (for the universal decompressor) maps y to x
- maximal lower semicomputable function $m(x|y*)$ that is monotone w.r.t. y and $\sum_x m(x|y*) \leq 1$ for every y

Monotone-conditional prefix complexity

- Even more splitting. . .
- A priori probability: random program (for the universal decompressor) maps y to x
- maximal lower semicomputable function $m(x|y*)$ that is monotone w.r.t. y and $\sum_x m(x|y*) \leq 1$ for every y
- Now they differ [Andreev]

Open question: can one prove the equivalence of prefix complexity definitions using prefix-free and prefix-stable decompressors, not using a priori probability as an intermediate step?

Monotone-conditional prefix complexity

- Even more splitting...
- A priori probability: random program (for the universal decompressor) maps y to x
- maximal lower semicomputable function $m(x|y*)$ that is monotone w.r.t. y and $\sum_x m(x|y*) \le 1$ for every y
- Now they differ [Andreev]

Open question: can one prove the equivalence of prefix complexity definitions using prefix-free and prefix-stable decompressors, not using a priori probability as an intermediate step?

Formal version: are the monotone-conditional complexities obtained using prefix-free and