Stopping time complexity and monotone-conditional complexity

alexander.shen@lirmm.fr, www.lirmm.fr/~ashen

LIRMM CNRS & University of Montpellier

Dagstuhl, February 2017

Vovk & Pavlovich idea

Vovk & Pavlovich idea

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

Vovk & Pavlovich idea

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

Vovk & Pavlovich idea

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

Vovk & Pavlovich idea

- How to tell which exit on a long road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits of information
- 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 long road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits of information
- 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 long road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits of information
- 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 complexity of a TM that stops after reading input x without trying to read the next bit

Vovk & Pavlovich idea

- How to tell which exit on a long road one should take?
- "Nth exit": $\log N$ bits of information
- "First exit after the bridge": $O(1)$ bits of information
- 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 complexity of a TM that stops after reading input x without trying to read the next bit
- = the minimal complexity of an algorithm that enumerates a prefix-free set of strings containing x

The classification of complexities

decompressor: descriptions $\rightarrow$ objects

The classification of complexities

decompressor: descriptions $\rightarrow$ objects

different "topologies" on descriptions and objects

The classification of complexities

decompressor: descriptions $\to$ 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 $x = C(x|x*)$

The classification of complexities

decompressor: descriptions $\to$ 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 $\to$ objects

8 versions of conditional complexities

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

objects: isolated;

descriptions: isolated;

conditions: prefixes

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)
- $C_D(y|x*) = \min\{|p|\colon D(p,x) = y\}$ (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)
- $C_D(y|x*) = \min\{|p| : D(p,x) = y\}$ (x is a condition)
- but D is required to be monotone (‘prefix-stable’) *with respect to condition*:
- if $D(p,x) = y$, then $D(p,x') = y$ for every extension x' of x

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 $D(p,x') = y$ for every extension x' of x
- $C(y|x*)$ = 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 $D(p,x') = y$ for every extension x' of x
- $C(y|x*)$ = 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 $D(p,x') = y$ for every extension x' of x
- $C(y|x*)$ = the minimal plain complexity of a prefix-stable program that maps x to y
- $C(x|x*)$ is not $O(1)$ anymore
- an equivalent definition of (plain) stopping time complexity

A quantitative characterization

A quantitative characterization

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

A quantitative characterization

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

A quantitative characterization

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

A quantitative characterization

- 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

A quantitative characterization

- 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
- Stopping time complexity: also upper semicomputable

A quantitative characterization

- 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
- Stopping time complexity: also upper semicomputable
- for every path α in the binary tree and for every n there are less than 2^n strings on this path with $C(x|x*) < n$.

A quantitative characterization

- 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
- Stopping time complexity: also upper semicomputable
- 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.

A quantitative characterization

- 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
- Stopping time complexity: also upper semicomputable
- 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 (Gleb Posobin)

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 $[C(x|y0*) \leq C(x|y*),\ C(x|y1*) \leq C(x|y*)]$ such that for every y and n there are at most 2^n objects x such that $C(x|y*) < n$. (Mikhail Andreev)

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]

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]
- $=$ minus logarithm of the maximal lower semicomputable function $m(x)$ whose sum along every path does not exceed 1 [Andreev]

Monotone-conditional prefix complexity

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 $\sum_x m(x|y*) \le 1$ 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?

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?
Formal version: are the monotone-conditional complexities obtained using prefix-free and prefix-stable (w.r.t. first argument) decompressors the same or not?