Algorithmic information theory: a gentle introduction

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

LIRMM CNRS & University of Montpellier

February 2016, IHP

Standard example

Standard example

- Four-letter alphabet $\{a, b, c, d\}$. Two bits per letter.

Standard example

- Four-letter alphabet $\{a, b, c, d\}$. Two bits per letter.
- Different frequencies: $1/8, 1/8, 1/4, 1/2$.

Standard example

- Four-letter alphabet $\{a, b, c, d\}$. Two bits per letter.
- Different frequencies: $1/8, 1/8, 1/4, 1/2$.
- Better encoding: $000, 001, 01, 1$

Standard example

- Four-letter alphabet $\{a, b, c, d\}$. Two bits per letter.
- Different frequencies: $1/8, 1/8, 1/4, 1/2$.
- Better encoding: $000, 001, 01, 1$
- In general: $\log(1/p_i)$ bits for a letter with frequency p_i, average $H = \sum p_i \log(1/p_i)$.

Standard example

- Four-letter alphabet $\{a, b, c, d\}$. Two bits per letter.
- Different frequencies: $1/8, 1/8, 1/4, 1/2$.
- Better encoding: $000, 001, 01, 1$
- In general: $\log(1/p_i)$ bits for a letter with frequency p_i, average $H = \sum p_i \log(1/p_i)$.
- "Statistical regularities can be used for compression"

Standard example

- Four-letter alphabet $\{a, b, c, d\}$. Two bits per letter.
- Different frequencies: $1/8, 1/8, 1/4, 1/2$.
- Better encoding: $000, 001, 01, 1$
- In general: $\log(1/p_i)$ bits for a letter with frequency p_i, average $H = \sum p_i \log(1/p_i)$.
- "Statistical regularities can be used for compression"
- Other types of regularities: block frequencies 50% compession if aa, bb, cc, dd only

Standard example

- Four-letter alphabet $\{a, b, c, d\}$. Two bits per letter.
- Different frequencies: $1/8, 1/8, 1/4, 1/2$.
- Better encoding: $000, 001, 01, 1$
- In general: $\log(1/p_i)$ bits for a letter with frequency p_i, average $H = \sum p_i \log(1/p_i)$.
- "Statistical regularities can be used for compression"
- Other types of regularities: block frequencies 50% compession if aa, bb, cc, dd only
- Non-statistical regularities: binary expansion of π is highly compressible.

Algorithmic complexity

Algorithmic complexity

- Goal: to define the amount of information in an individual object (genome, picture, . . .)

Algorithmic complexity

- Goal: to define the amount of information in an individual object (genome, picture,...)
- Idea: "amount of information = number of bits needed to define (describe, specify,...) a given object"

Algorithmic complexity

- Goal: to define the amount of information in an individual object (genome, picture,. . .)
- Idea: "amount of information = number of bits needed to define (describe, specify,. . .) a given object"
- "Define" is vague:

THE MINIMAL POSITIVE INTEGER THAT CANNOT BE DEFINED BY LESS THAN THOUSAND ENGLISH WORDS

Algorithmic complexity

- Goal: to define the amount of information in an individual object (genome, picture,...)
- Idea: "amount of information = number of bits needed to define (describe, specify,...) a given object"
- "Define" is vague:

THE MINIMAL POSITIVE INTEGER THAT CANNOT BE DEFINED BY LESS THAN THOUSAND ENGLISH WORDS

- More precise version: *algorithmic complexity of x is the minimal length of a program that produces x.*

Algorithmic complexity

- Goal: to define the amount of information in an individual object (genome, picture, . . .)
- Idea: "amount of information = number of bits needed to define (describe, specify, . . .) a given object"
- "Define" is vague:

THE MINIMAL POSITIVE INTEGER THAT CANNOT BE DEFINED BY LESS THAN THOUSAND ENGLISH WORDS

- More precise version: *algorithmic complexity of x is the minimal length of a program that produces x.*
- "compressed size"

Algorithmic complexity

- Goal: to define the amount of information in an individual object (genome, picture,...)
- Idea: "amount of information = number of bits needed to define (describe, specify,...) a given object"
- "Define" is vague:

  THE MINIMAL POSITIVE INTEGER THAT CANNOT BE DEFINED BY LESS THAN THOUSAND ENGLISH WORDS

- More precise version: *algorithmic complexity of x is the minimal length of a program that produces x*.
- "compressed size"
- but we do not care about compression, only decompression matters

Kolmogorov complexity and decompressors

Kolmogorov complexity and decompressors

- *decompressor* = any partial computable function V from $\{0,1\}^*$ to $\{0,1\}^*$ (we define complexity of strings)

Kolmogorov complexity and decompressors

- *decompressor* = any partial computable function V from $\{0,1\}^*$ to $\{0,1\}^*$ (we define complexity of strings)
- decompressor = programming language (without input): if $V(x) = z$ we say that "x is a program for z"

Kolmogorov complexity and decompressors

- *decompressor* = any partial computable function V from $\{0,1\}^*$ to $\{0,1\}^*$ (we define complexity of strings)
- decompressor = programming language (without input): if $V(x) = z$ we say that "x is a program for z"

  ("description" of z, "compressed version" of z, etc.)

Kolmogorov complexity and decompressors

- *decompressor* = any partial computable function V from $\{0,1\}^*$ to $\{0,1\}^*$ (we define complexity of strings)
- decompressor = programming language (without input): if $V(x) = z$ we say that "x is a program for z"

  ("description" of z, "compressed version" of z, etc.)
- Given decompressor V, we define the *complexity* of a string z w.r.t. this decompressor

Kolmogorov complexity and decompressors

- *decompressor* = any partial computable function V from $\{0,1\}^*$ to $\{0,1\}^*$ (we define complexity of strings)
- decompressor = programming language (without input): if $V(x) = z$ we say that "x is a program for z"

  ("description" of z, "compressed version" of z, etc.)
- Given decompressor V, we define the *complexity* of a string z w.r.t. this decompressor

$$C_V(z) = min\{|x| : V(x) = z\}$$

Kolmogorov complexity and decompressors

- *decompressor* = any partial computable function V from $\{0,1\}^*$ to $\{0,1\}^*$ (we define complexity of strings)
- decompressor = programming language (without input): if $V(x) = z$ we say that "x is a program for z"

  ("description" of z, "compressed version" of z, etc.)
- Given decompressor V, we define the *complexity* of a string z w.r.t. this decompressor

$$C_V(z) = min\{|x| : V(x) = z\}$$

- $\min \varnothing = +\infty$

Kolmogorov complexity and decompressors

- *decompressor* = any partial computable function V from $\{0,1\}^*$ to $\{0,1\}^*$ (we define complexity of strings)
- decompressor = programming language (without input): if $V(x) = z$ we say that "x is a program for z"

  ("description" of z, "compressed version" of z, etc.)
- Given decompressor V, we define the *complexity* of a string z w.r.t. this decompressor

$$C_V(z) = min\{|x| : V(x) = z\}$$

- $\min \varnothing = +\infty$
- Can one achieve something by this trivial definition?!

Function C_V

Function C_V

- Without computability: let V be an arbitrary partial function from strings to strings

Function C_V

- Without computability: let V be an arbitrary partial function from strings to strings
- which functions C_V do we obtain?

Function C_V

- Without computability: let V be an arbitrary partial function from strings to strings
- which functions C_V do we obtain?
- $\#\{z\colon C_V(z) = k\} \le 2^k$

Function C_V

- Without computability: let V be an arbitrary partial function from strings to strings
- which functions C_V do we obtain?
- $\#\{z\colon C_V(z) = k\} \le 2^k$
- necessary and sufficient condition

Function C_V

- Without computability: let V be an arbitrary partial function from strings to strings
- which functions C_V do we obtain?
- $\#\{z\colon C_V(z) = k\} \le 2^k$
- necessary and sufficient condition

  use k-bit strings as “descriptions” (“compressed versions”) of strings z with $C(z) = k$.

Function C_V

- Without computability: let V be an arbitrary partial function from strings to strings
- which functions C_V do we obtain?
- $\#\{z\colon C_V(z)=k\} \le 2^k$
- necessary and sufficient condition

  use k-bit strings as "descriptions" ("compressed versions") of strings z with $C(z)=k$.
- for every z one can trivially find V that makes $C_V(z)=0$ (map empty string Λ to z)

Function C_V

- Without computability: let V be an arbitrary partial function from strings to strings
- which functions C_V do we obtain?
- $\#\{z\colon C_V(z)=k\}\le 2^k$
- necessary and sufficient condition

  use k-bit strings as “descriptions” (“compressed versions”) of strings z with $C(z)=k$.
- for every z one can trivially find V that makes $C_V(z)=0$ (map empty string Λ to z)
- So what? Even if we restrict V to computable partial functions, can we get anything non-trivial?

An easy exercise

An easy exercise

- For every two decompressors V_0 and V_1 there exist some V such that

$$C_V(z) \le \min(C_{V_0}(z), C_{V_1}(z)) + O(1) \quad \text{for all } z$$

An easy exercise

- For every two decompressors V_0 and V_1 there exist some V such that

$$C_V(z) \le \min(C_{V_0}(z), C_{V_1}(z)) + O(1) \quad \text{for all } z$$

- "V is (almost) as good as each of V_0, V_1"

An easy exercise

- For every two decompressors V_0 and V_1 there exist some V such that

$$C_V(z) \le \min(C_{V_0}(z), C_{V_1}(z)) + O(1) \quad \text{for all } z$$

- "V is (almost) as good as each of V_0, V_1"
- $V(0x) = V_0(x)$ and $V(1x) = V_1(x)$

An easy exercise

- For every two decompressors V_0 and V_1 there exist some V such that

$$C_V(z) \leq \min(C_{V_0}(z), C_{V_1}(z)) + O(1) \quad \text{for all } z$$

- "V is (almost) as good as each of V_0, V_1"
- $V(0x) = V_0(x)$ and $V(1x) = V_1(x)$
- first we specify which decompressor to use, and then the short program for this decompressor

An easy exercise

- For every two decompressors V_0 and V_1 there exist some V such that

$$C_V(z) \le \min(C_{V_0}(z), C_{V_1}(z)) + O(1) \quad \text{for all } z$$

- "V is (almost) as good as each of V_0, V_1"
- $V(0x) = V_0(x)$ and $V(1x) = V_1(x)$
- first we specify which decompressor to use, and then the short program for this decompressor
- preserves computability

An easy exercise

- For every two decompressors V_0 and V_1 there exist some V such that

$$C_V(z) \le \min(C_{V_0}(z), C_{V_1}(z)) + O(1) \quad \text{for all } z$$

- "V is (almost) as good as each of V_0, V_1"
- $V(0x) = V_0(x)$ and $V(1x) = V_1(x)$
- first we specify which decompressor to use, and then the short program for this decompressor
- preserves computability
- "practical application": zipped file starts with a header that specifies compression method (2^k methods for k-bit header)

Optimal decompressors

Optimal decompressors

- For every sequence $V_0, V_1, \ldots$ of decompressors there exist some V such that $C_V(z) \leq C_{V_i}(z) + 2\log i + c$ for some c and for all i and z

Optimal decompressors

- For every sequence $V_0, V_1, \ldots$ of decompressors there exist some V such that $C_V(z) \le C_{V_i}(z) + 2\log i + c$ for some c and for all i and z
- V is almost as good as every V_i (and the price to pay is moderate, only $O(\log i)$)

Optimal decompressors

- For every sequence $V_0, V_1, \ldots$ of decompressors there exist some V such that $C_V(z) \leq C_{V_i}(z) + 2\log i + c$ for some c and for all i and z
- V is almost as good as every V_i (and the price to pay is moderate, only $O(\log i)$)
- proof: prepend V_i-programs by a self-delimited description of i (say, i in binary with all bits doubled, terminated by 01)

Optimal decompressors

- For every sequence $V_0, V_1, \ldots$ of decompressors there exist some V such that $C_V(z) \leq C_{V_i}(z) + 2\log i + c$ for some c and for all i and z
- V is almost as good as every V_i (and the price to pay is moderate, only $O(\log i)$)
- proof: prepend V_i-programs by a self-delimited description of i (say, i in binary with all bits doubled, terminated by 01)
- the computable enumeration of all computable V_i gives

Optimal decompressors

- For every sequence $V_0, V_1, \ldots$ of decompressors there exist some V such that $C_V(z) \le C_{V_i}(z) + 2\log i + c$ for some c and for all i and z
- V is almost as good as every V_i (and the price to pay is moderate, only $O(\log i)$)
- proof: prepend V_i-programs by a self-delimited description of i (say, i in binary with all bits doubled, terminated by 01)
- the computable enumeration of all computable V_i gives "Kolmogorov–Solomonoff theorem": *there exists an optimal computable decompressor that is almost as good as any other computable one.*

Optimal decompressors

- For every sequence $V_0, V_1, \ldots$ of decompressors there exist some V such that $C_V(z) \le C_{V_i}(z) + 2\log i + c$ for some c and for all i and z
- V is almost as good as every V_i (and the price to pay is moderate, only $O(\log i)$)
- proof: prepend V_i-programs by a self-delimited description of i (say, i in binary with all bits doubled, terminated by 01)
- the computable enumeration of all computable V_i gives "Kolmogorov–Solomonoff theorem": *there exists an optimal computable decompressor that is almost as good as any other computable one*.
- C_U for such an optimal U is called "algorithmic complexity" (or Kolmogorov complexity) and denoted by C

Optimal decompressors

- For every sequence $V_0, V_1, \ldots$ of decompressors there exist some V such that $C_V(z) \le C_{V_i}(z) + 2\log i + c$ for some c and for all i and z
- V is almost as good as every V_i (and the price to pay is moderate, only $O(\log i)$)
- proof: prepend V_i-programs by a self-delimited description of i (say, i in binary with all bits doubled, terminated by 01)
- the computable enumeration of all computable V_i gives "Kolmogorov–Solomonoff theorem": *there exists an optimal computable decompressor that is almost as good as any other computable one*.
- C_U for such an optimal U is called "algorithmic complexity" (or Kolmogorov complexity) and denoted by C
- "application": self-extracting archives

Some natural properties of C

Some natural properties of C

- $C(x) \leq |x| + O(1)$.

Some natural properties of C

- $C(x) \leq |x| + O(1)$.

  Indeed, compare the optimal decompressor U with $x \mapsto x$

Some natural properties of C

- $C(x) \leq |x| + O(1)$.
  Indeed, compare the optimal decompressor U with $x \mapsto x$
- "algorithmic transformations do not create new information": if A is some computable function, then $C(A(x)) \leq C(x) + c_A$ for some c_A and all x (here c_A depends on A but not on x)

Some natural properties of C

- $C(x) \leq |x| + O(1)$.
  Indeed, compare the optimal decompressor U with $x \mapsto x$
- "algorithmic transformations do not create new information": if A is some computable function, then $C(A(x)) \leq C(x) + c_A$ for some c_A and all x (here c_A depends on A but not on x)
- there is less than 2^n objects of complexity less than n.

Some natural properties of C

- $C(x) \leq |x| + O(1)$.

  Indeed, compare the optimal decompressor U with $x \mapsto x$
- "algorithmic transformations do not create new information": if A is some computable function, then $C(A(x)) \leq C(x) + c_A$ for some c_A and all x (here c_A depends on A but not on x)
- there is less than 2^n objects of complexity less than n.

  Indeed, the number of descriptions shorter than n does not exceed $1 + 2 + 4 + \ldots + 2^{n-1} < 2^n$

Some natural properties of C

- $C(x) \leq |x| + O(1)$.

  Indeed, compare the optimal decompressor U with $x \mapsto x$
- "algorithmic transformations do not create new information": if A is some computable function, then $C(A(x)) \leq C(x) + c_A$ for some c_A and all x (here c_A depends on A but not on x)
- there is less than 2^n objects of complexity less than n.

  Indeed, the number of descriptions shorter than n does not exceed $1 + 2 + 4 + \ldots + 2^{n-1} < 2^n$
- Some objects are highly compressible, e.g., $C(0^n) \leq \log n + c$

Some natural properties of C

- $C(x) \le |x| + O(1)$.

  Indeed, compare the optimal decompressor U with $x \mapsto x$
- "algorithmic transformations do not create new information": if A is some computable function, then $C(A(x)) \le C(x) + c_A$ for some c_A and all x (here c_A depends on A but not on x)
- there is less than 2^n objects of complexity less than n.

  Indeed, the number of descriptions shorter than n does not exceed $1 + 2 + 4 + \ldots + 2^{n-1} < 2^n$
- Some objects are highly compressible, e.g., $C(0^n) \le \log n + c$

  Indeed, consider the algorithmic transformation $\mathrm{bin}(n) \mapsto 0^n$

Some natural properties of C

- $C(x) \leq |x| + O(1)$.

  Indeed, compare the optimal decompressor U with $x \mapsto x$
- "algorithmic transformations do not create new information": if A is some computable function, then $C(A(x)) \leq C(x) + c_A$ for some c_A and all x (here c_A depends on A but not on x)
- there is less than 2^n objects of complexity less than n.

  Indeed, the number of descriptions shorter than n does not exceed $1 + 2 + 4 + \ldots + 2^{n-1} < 2^n$
- Some objects are highly compressible, e.g., $C(0^n) \leq \log n + c$

  Indeed, consider the algorithmic transformation $\mathrm{bin}(n) \mapsto 0^n$
- but most are not: the fraction of strings x of length n such that $C(x) < n - c$ is less than 2^{-c}

Some natural properties of C

- $C(x) \leq |x| + O(1)$.
  Indeed, compare the optimal decompressor U with $x \mapsto x$
- "algorithmic transformations do not create new information": if A is some computable function, then $C(A(x)) \leq C(x) + c_A$ for some c_A and all x (here c_A depends on A but not on x)
- there is less than 2^n objects of complexity less than n.
  Indeed, the number of descriptions shorter than n does not exceed $1 + 2 + 4 + \ldots + 2^{n-1} < 2^n$
- Some objects are highly compressible, e.g., $C(0^n) \leq \log n + c$
  Indeed, consider the algorithmic transformation $\mathrm{bin}(n) \mapsto 0^n$
- but most are not: the fraction of strings x of length n such that $C(x) < n - c$ is less than 2^{-c}
- Law of nature: *tossing* 8000 *coins, you get a sequence of* 1000 *bytes that has zip-compressed length at least* 900. Does it follow from the known laws of physics (and how if it does)?

Bad news

Bad news

- We defined the complexity function C, but in fact it is defined only up to $O(1)$-change

Bad news

- We defined the complexity function C, but in fact it is defined only up to $O(1)$-change

  ... unless you declare your favourite programming language to be "the right one"

Bad news

- We defined the complexity function C, but in fact it is defined only up to $O(1)$-change

  ... unless you declare your favourite programming language to be "the right one"

- So the questions "is $C(0^{1000}) < 15$"? or "what is bigger: $C(010)$ or $C(101)$" do not make sense

Bad news

- We defined the complexity function C, but in fact it is defined only up to $O(1)$-change

  ... unless you declare your favourite programming language to be "the right one"
- So the questions "is $C(0^{1000}) < 15$"? or "what is bigger: $C(010)$ or $C(101)$" do not make sense
- Theorem: function $C(\cdot)$ is not computable (and even does not have a computable lower bound)

Bad news

- We defined the complexity function C, but in fact it is defined only up to $O(1)$-change

  …unless you declare your favourite programming language to be "the right one"
- So the questions "is $C(0^{1000}) < 15$"? or "what is bigger: $C(010)$ or $C(101)$" do not make sense
- Theorem: function $C(\cdot)$ is not computable (and even does not have a computable lower bound)
- proof: if it were, the string x_n, "the first string that has complexity at least n", has complexity at least n and at most $O(\log n)$ at the same time (since it is obtained algorithmically from n)

Digression: Gödel and Chaitin

Digression: Gödel and Chaitin

- Gödel: not all true statements are provable

Digression: Gödel and Chaitin

- Gödel: not all true statements are provable
- Chaitin: not all true statements of the form "$C(x) > m$" where x is a specific string, and m is a specific number, are provable

Digression: Gödel and Chaitin

- Gödel: not all true statements are provable
- Chaitin: not all true statements of the form "$C(x) > m$" where x is a specific string, and m is a specific number, are provable
- moreover, they are provable only for m not exceeding some constant

Digression: Gödel and Chaitin

- Gödel: not all true statements are provable
- Chaitin: not all true statements of the form "$C(x) > m$" where x is a specific string, and m is a specific number, are provable
- moreover, they are provable only for m not exceeding some constant
- Why? If not, consider the function $m \mapsto y_m =$ (the first discovered string with complexity provably exceeding m)

Digression: Gödel and Chaitin

- Gödel: not all true statements are provable
- Chaitin: not all true statements of the form "$C(x) > m$" where x is a specific string, and m is a specific number, are provable
- moreover, they are provable only for m not exceeding some constant
- Why? If not, consider the function $m \mapsto y_m =$ (the first discovered string with complexity provably exceeding m)
- the complexity of y_m is at least m (assuming only true statements are provable)

Digression: Gödel and Chaitin

- Gödel: not all true statements are provable
- Chaitin: not all true statements of the form "$C(x) > m$" where x is a specific string, and m is a specific number, are provable
- moreover, they are provable only for m not exceeding some constant
- Why? If not, consider the function $m \mapsto y_m =$ (the first discovered string with complexity provably exceeding m)
- the complexity of y_m is at least m (assuming only true statements are provable)
- the complexity of y_m is at most $\log m + O(1)$ since it is obtained from m by an algorithmic transformation

Digression: Gödel and Chaitin

- Gödel: not all true statements are provable
- Chaitin: not all true statements of the form "$C(x) > m$" where x is a specific string, and m is a specific number, are provable
- moreover, they are provable only for m not exceeding some constant
- Why? If not, consider the function $m \mapsto y_m =$ (the first discovered string with complexity provably exceeding m)
- the complexity of y_m is at least m (assuming only true statements are provable)
- the complexity of y_m is at most $\log m + O(1)$ since it is obtained from m by an algorithmic transformation
- second order digression: axiomatic power of statements of this form

Good news

Good news

- Still the asymptotic considerations have sense

Good news

- Still the asymptotic considerations have sense
- e.g., one can define "effective Hausdorff dimension" of an individual infinite bit sequence $x_0x_1\ldots$ as $\liminf C(x_0\ldots x_{n-1})/n$

Good news

- Still the asymptotic considerations have sense
- e.g., one can define "effective Hausdorff dimension" of an individual infinite bit sequence $x_0x_1\ldots$ as $\liminf C(x_0\ldots x_{n-1})/n$
- (Hausdorff dimension for a singleton?!)

Good news

- Still the asymptotic considerations have sense
- e.g., one can define "effective Hausdorff dimension" of an individual infinite bit sequence $x_0 x_1 \ldots$ as $\liminf C(x_0 \ldots x_{n-1})/n$
- (Hausdorff dimension for a singleton?!)
- theorem: if $x = x_0 x_1 \ldots$ is obtained by independent trials of Bernoulli distribution $(p, 1-p)$, then with probability 1 the effective Hausdorff dimension of x is $H(p)$.

Good news

- Still the asymptotic considerations have sense
- e.g., one can define "effective Hausdorff dimension" of an individual infinite bit sequence $x_0 x_1 \ldots$ as $\liminf C(x_0 \ldots x_{n-1})/n$
- (Hausdorff dimension for a singleton?!)
- theorem: if $x = x_0 x_1 \ldots$ is obtained by independent trials of Bernoulli distribution $(p, 1-p)$, then with probability 1 the effective Hausdorff dimension of x is $H(p)$.
- finite version: if p is a frequence of 1s in a n-bit string x, then

$$C(x) \leq nH(p) + O(\log n)$$

Good news

- Still the asymptotic considerations have sense
- e.g., one can define "effective Hausdorff dimension" of an individual infinite bit sequence $x_0 x_1 \ldots$ as $\liminf C(x_0 \ldots x_{n-1})/n$
- (Hausdorff dimension for a singleton?!)
- theorem: if $x = x_0 x_1 \ldots$ is obtained by independent trials of Bernoulli distribution $(p, 1-p)$, then with probability 1 the effective Hausdorff dimension of x is $H(p)$.
- finite version: if p is a frequence of 1s in a n-bit string x, then

$$C(x) \le nH(p) + O(\log n)$$

- Even for genome (or a long novel) the notion of complexity has sense: different "natural" programming languages give complexities that are 10^2–10^5 apart (the length of a compiler)

Parallels with classical information theory

Parallels with classical information theory

- $H(X, Y) \leq H(X) + H(Y)$

Parallels with classical information theory

- $H(X, Y) \leq H(X) + H(Y)$
  computation: convexity of logarithms

Parallels with classical information theory

- $H(X,Y) \leq H(X) + H(Y)$

  computation: convexity of logarithms
- In AIT: if there is a short program computing x, and another short program computing y, they could be combined into a program that computes a pair (x,y) (some encoding of it)

Parallels with classical information theory

- $H(X, Y) \leq H(X) + H(Y)$

  computation: convexity of logarithms
- In AIT: if there is a short program computing x, and another short program computing y, they could be combined into a program that computes a pair (x, y) (some encoding of it)
- complexity of a pair = complexity of its encoding (change of the encoding is a computable transformation, so only $O(1)$-change in complexity)

Parallels with classical information theory

- $H(X, Y) \leq H(X) + H(Y)$
  computation: convexity of logarithms
- In AIT: if there is a short program computing x, and another short program computing y, they could be combined into a program that computes a pair (x, y) (some encoding of it)
- complexity of a pair = complexity of its encoding (change of the encoding is a computable transformation, so only $O(1)$-change in complexity)
- $C(x, y) \leq C(x) + C(y) + O(\log(C(x) + C(y))$

Parallels with classical information theory

- $H(X,Y) \leq H(X) + H(Y)$

  computation: convexity of logarithms
- In AIT: if there is a short program computing x, and another short program computing y, they could be combined into a program that computes a pair (x,y) (some encoding of it)
- complexity of a pair = complexity of its encoding (change of the encoding is a computable transformation, so only $O(1)$-change in complexity)
- $C(x,y) \leq C(x) + C(y) + O(\log(C(x) + C(y)))$
- logarithmic overhead needed to separate the programs

Parallels with classical information theory

- $H(X, Y) \leq H(X) + H(Y)$

  computation: convexity of logarithms
- In AIT: if there is a short program computing x, and another short program computing y, they could be combined into a program that computes a pair (x, y) (some encoding of it)
- complexity of a pair = complexity of its encoding (change of the encoding is a computable transformation, so only $O(1)$-change in complexity)
- $C(x, y) \leq C(x) + C(y) + O(\log(C(x) + C(y))$
- logarithmic overhead needed to separate the programs
- why so different arguments for parallel statements?

One more example

One more example

- $H(X,Y) = H(X) + H(Y|X)$

One more example

- $H(X, Y) = H(X) + H(Y|X)$
- parallel statement: $C(x, y) \approx C(x) + C(y|x)$

One more example

- $H(X, Y) = H(X) + H(Y|X)$
- parallel statement: $C(x, y) \approx C(x) + C(y|x)$

  ... but first we need to define $C(y|x)$

One more example

- $H(X,Y) = H(X) + H(Y|X)$
- parallel statement: $C(x,y) \approx C(x) + C(y|x)$

  ...but first we need to define $C(y|x)$
- $C(y|x) =$ the minimal length of a program that maps x to y

One more example

- $H(X, Y) = H(X) + H(Y|X)$
- parallel statement: $C(x, y) \approx C(x) + C(y|x)$

  . . . but first we need to define $C(y|x)$
- $C(y|x)$ = the minimal length of a program that maps x to y

  "conditional complexity"

One more example

- $H(X, Y) = H(X) + H(Y|X)$
- parallel statement: $C(x, y) \approx C(x) + C(y|x)$

  ...but first we need to define $C(y|x)$
- $C(y|x)$ = the minimal length of a program that maps x to y

  "conditional complexity"
- this statement is true with the same logarithmic precision

One more example

- $H(X, Y) = H(X) + H(Y|X)$
- parallel statement: $C(x, y) \approx C(x) + C(y|x)$

  ...but first we need to define $C(y|x)$
- $C(y|x) =$ the minimal length of a program that maps x to y

  "conditional complexity"
- this statement is true with the same logarithmic precision
- one direction ($\leq$): the same argument

One more example

- $H(X, Y) = H(X) + H(Y|X)$
- parallel statement: $C(x, y) \approx C(x) + C(y|x)$

  …but first we need to define $C(y|x)$
- $C(y|x) =$ the minimal length of a program that maps x to y

  "conditional complexity"
- this statement is true with the same logarithmic precision
- one direction ($\leq$): the same argument
- another direction more interesting: why looking for a short program that produces (x, y) we may assume w.l.o.g. it consists of two parts: first producing x and second transforming x to y?

Combinatorial versions – 1

Combinatorial versions – 1

- $H(X, Y) \leq H(X) + H(Y)$

Combinatorial versions – 1

- $H(X,Y) \le H(X) + H(Y)$
- $\log S(A) \le \log S(A_x) + \log S(A_y)$

- $H(X, Y) \leq H(X) + H(Y)$
- $\log S(A) \leq \log S(A_x) + \log S(A_y)$

  here $A \subset X \times Y$ is a two-dimensional set,

Combinatorial versions – 1

- $H(X, Y) \leq H(X) + H(Y)$
- $\log S(A) \leq \log S(A_x) + \log S(A_y)$

  here $A \subset X \times Y$ is a two-dimensional set,

  A_x and A_y are projections of A onto X and Y

Combinatorial versions – 1

- $H(X,Y) \le H(X) + H(Y)$
- $\log S(A) \le \log S(A_x) + \log S(A_y)$

  here $A \subset X \times Y$ is a two-dimensional set,

  A_x and A_y are projections of A onto X and Y

  and S stands for the "size" (cardinality in the discrete version, area/length in the continuous version)

Combinatorial versions – 2

Combinatorial versions – 2

- $H(X,Y) \le H(X) + H(Y|X)$

Combinatorial versions – 2

- $H(X,Y) \le H(X) + H(Y|X)$
- $\log S(A) \le \log S(A_x) + \log \max_x S(A_{y|x})$

Combinatorial versions – 2

- $H(X,Y) \leq H(X) + H(Y|X)$
- $\log S(A) \leq \log S(A_x) + \log \max_x S(A_{y|x})$
  here $A \subset X \times Y$ is a two-dimensional set,

Combinatorial versions – 2

- $H(X, Y) \leq H(X) + H(Y|X)$
- $\log S(A) \leq \log S(A_x) + \log \max_x S(A_{y|x})$
  here $A \subset X \times Y$ is a two-dimensional set,
  $A_x \subset X$ is the projection of A onto X

Combinatorial versions – 2

- $H(X, Y) \le H(X) + H(Y|X)$
- $\log S(A) \le \log S(A_x) + \log \max_x S(A_{y|x})$

  here $A \subset X \times Y$ is a two-dimensional set,

  $A_x \subset X$ is the projection of A onto X

  $A_{y|x} \subset Y$ is the x-th "vertical section" of A where the X-coordinate is fixed

Combinatorial versions – 2

- $H(X,Y) \leq H(X) + H(Y|X)$
- $\log S(A) \leq \log S(A_x) + \log \max_x S(A_{y|x})$

  here $A \subset X \times Y$ is a two-dimensional set,

  $A_x \subset X$ is the projection of A onto X

  $A_{y|x} \subset Y$ is the x-th "vertical section" of A where the X-coordinate is fixed

  and S stands for the "size"

Combinatorial versions – 2

- $H(X, Y) \leq H(X) + H(Y|X)$
- $\log S(A) \leq \log S(A_x) + \log \max_x S(A_{y|x})$

  here $A \subset X \times Y$ is a two-dimensional set,

  $A_x \subset X$ is the projection of A onto X

  $A_{y|x} \subset Y$ is the x-th "vertical section" of A where the X-coordinate is fixed

  and S stands for the "size"
- In other words: if A_x is of size at most 2^l and all sections $A_{y|x}$ are of size at most 2^m, then A is of size at most 2^{l+m}.

Combinatorial versions – 2

- $H(X, Y) \leq H(X) + H(Y|X)$
- $\log S(A) \leq \log S(A_x) + \log \max_x S(A_{y|x})$

  here $A \subset X \times Y$ is a two-dimensional set,

  $A_x \subset X$ is the projection of A onto X

  $A_{y|x} \subset Y$ is the x-th "vertical section" of A where the X-coordinate is fixed

  and S stands for the "size"
- In other words: if A_x is of size at most 2^l and all sections $A_{y|x}$ are of size at most 2^m, then A is of size at most 2^{l+m}.
- now closer to the algorithmic statement: to specify an element (x, y) of A, we may first use l bits to specify x and then m bits to specify y inside x-section $A_{y|x}$

Combinatorial versions – 3

Combinatorial versions – 3

- $H(X) + H(Y|X) \leq H(X,Y)$

Combinatorial versions – 3

- $H(X) + H(Y|X) \leq H(X,Y)$
- combinatorial statement not so obvious: A may have small size, at the same time its projection A_x can be rather large and some section $A_{y|x}$ can be rather large

Combinatorial versions – 3

- $H(X) + H(Y|X) \le H(X,Y)$
- combinatorial statement not so obvious: A may have small size, at the same time its projection A_x can be rather large and some section $A_{y|x}$ can be rather large
- in other words: the *average* size of a section may be much less than the *maximal* size

Combinatorial versions – 3

- $H(X) + H(Y|X) \leq H(X,Y)$
- combinatorial statement not so obvious: A may have small size, at the same time its projection A_x can be rather large and some section $A_{y|x}$ can be rather large
- in other words: the *average* size of a section may be much less than the *maximal* size
- correct version: if $S(A) \leq 2^{l+m}$, then A can be represented as $A' \cup A''$ where (1) A' has small projection: $S(A'_x) \leq 2^l$

Combinatorial versions – 3

- $H(X) + H(Y|X) \leq H(X, Y)$
- combinatorial statement not so obvious: A may have small size, at the same time its projection A_x can be rather large and some section $A_{y|x}$ can be rather large
- in other words: the *average* size of a section may be much less than the *maximal* size
- correct version: if $S(A) \leq 2^{l+m}$, then A can be represented as $A' \cup A''$ where (1) A' has small projection: $S(A'_x) \leq 2^l$ and (2) all sections of A'' are small: $S(A''_{y|x}) \leq 2^m$ for every x.

Combinatorial versions – 3

- $H(X) + H(Y|X) \leq H(X, Y)$
- combinatorial statement not so obvious: A may have small size, at the same time its projection A_x can be rather large and some section $A_{y|x}$ can be rather large
- in other words: the *average* size of a section may be much less than the *maximal* size
- correct version: if $S(A) \leq 2^{l+m}$, then A can be represented as $A' \cup A''$ where (1) A' has small projection: $S(A'_x) \leq 2^l$ and (2) all sections of A'' are small: $S(A''_{y|x}) \leq 2^m$ for every x.
- proof: let A' be the union of all sections that are larger than 2^m, and A'' be the rest (the union of all small sections)

Algorithmic complexity version

Algorithmic complexity version

- $C(x, y) \geq C(x) + C(y|x)$ (ignoring logarithmic overhead)

Algorithmic complexity version

- $C(x, y) \geq C(x) + C(y|x)$ (ignoring logarithmic overhead)
- reformulation: if $C(x, y) < l + m$, then either $C(x) < l$ or $C(y|x) < m$.

Algorithmic complexity version

- $C(x,y) \geq C(x) + C(y|x)$ (ignoring logarithmic overhead)
- reformulation: if $C(x,y) < l + m$, then either $C(x) < l$ or $C(y|x) < m$.
- enumerate pairs (x,y) with $C(x,y) < l + m$; there are at most 2^{l+m} such pairs

Algorithmic complexity version

- $C(x, y) \geq C(x) + C(y|x)$ (ignoring logarithmic overhead)
- reformulation: if $C(x, y) < l + m$, then either $C(x) < l$ or $C(y|x) < m$.
- enumerate pairs (x, y) with $C(x, y) < l + m$; there are at most 2^{l+m} such pairs
- while for a given x at most 2^m pairs (x, y) with this x and different y's are discovered, each of these y can be specified by its ordinal number (at most m bits) assuming x is known

Algorithmic complexity version

- $C(x,y) \geq C(x) + C(y|x)$ (ignoring logarithmic overhead)
- reformulation: if $C(x,y) < l + m$, then either $C(x) < l$ or $C(y|x) < m$.
- enumerate pairs (x,y) with $C(x,y) < l + m$; there are at most 2^{l+m} such pairs
- while for a given x at most 2^m pairs (x,y) with this x and different y's are discovered, each of these y can be specified by its ordinal number (at most m bits) assuming x is known
- for some pairs (x,y) this does not work: there are more than 2^m pairs with this x. But there are at most 2^l "bad" x, and each of them can be specified by its ordinal number (at most l bits)

Algorithmic complexity version

- $C(x, y) \geq C(x) + C(y|x)$ (ignoring logarithmic overhead)
- reformulation: if $C(x, y) < l + m$, then either $C(x) < l$ or $C(y|x) < m$.
- enumerate pairs (x, y) with $C(x, y) < l + m$; there are at most 2^{l+m} such pairs
- while for a given x at most 2^m pairs (x, y) with this x and different y's are discovered, each of these y can be specified by its ordinal number (at most m bits) assuming x is known
- for some pairs (x, y) this does not work: there are more than 2^m pairs with this x. But there are at most 2^l "bad" x, and each of them can be specified by its ordinal number (at most l bits)
- these cases correspond to $C(y|x) \leq m$ and $C(x) \leq l$ (plus logarithmic overhead) respectively

Romashchenko's theorem

Romashchenko's theorem

- More than informal analogy

Romashchenko's theorem

- More than informal analogy
- Linear inequalities for entropies:

$$\lambda_{XYZ}H(X,Y,Z)+\lambda_{XY}H(X,Y)+\lambda_{XZ}H(X,Z)+\lambda_{YZ}H(Y,Z)+ \lambda_X H(X)+\lambda_Y H(Y)+\lambda_Z H(Z)\ge 0$$

Romashchenko's theorem

- More than informal analogy
- Linear inequalities for entropies:

$$\lambda_{XYZ}H(X,Y,Z)+\lambda_{XY}H(X,Y)+\lambda_{XZ}H(X,Z)+\lambda_{YZ}H(Y,Z)+ \lambda_X H(X)+\lambda_Y H(Y)+\lambda_Z H(Z)\ge 0$$

- e.g., $H(X)+H(X,Y,Z)\le H(X,Y)+H(X,Z)$ (expanded version of $I(Y:Z|X)\ge 0$)

Romashchenko's theorem

- More than informal analogy
- Linear inequalities for entropies:

$$\lambda_{XYZ}H(X,Y,Z)+\lambda_{XY}H(X,Y)+\lambda_{XZ}H(X,Z)+\lambda_{YZ}H(Y,Z)+\lambda_X H(X)+\lambda_Y H(Y)+\lambda_Z H(Z)\ge 0$$

- e.g., $H(X)+H(X,Y,Z)\le H(X,Y)+H(X,Z)$ (expanded version of $I(Y:Z|X)\ge 0$)
- the description of all true inequalities of this type (the dual cone to the set of entropy tuples) is an open difficult problem for >3 variables

Romashchenko's theorem

- More than informal analogy
- Linear inequalities for entropies:

$$\lambda_{XYZ} H(X,Y,Z) + \lambda_{XY} H(X,Y) + \lambda_{XZ} H(X,Z) + \lambda_{YZ} H(Y,Z) + \\ \lambda_X H(X) + \lambda_Y H(Y) + \lambda_Z H(Z) \geq 0$$

- e.g., $H(X) + H(X,Y,Z) \leq H(X,Y) + H(X,Z)$ (expanded version of $I(Y:Z|X) \geq 0$)
- the description of all true inequalities of this type (the dual cone to the set of entropy tuples) is an open difficult problem for > 3 variables
- Romashchenko: exactly the same inequalities are true for Kolmogorov complexities

Romashchenko's theorem

- More than informal analogy
- Linear inequalities for entropies:

$$\lambda_{XYZ} H(X,Y,Z) + \lambda_{XY} H(X,Y) + \lambda_{XZ} H(X,Z) + \lambda_{YZ} H(Y,Z) + \lambda_X H(X) + \lambda_Y H(Y) + \lambda_Z H(Z) \geq 0$$

- e.g., $H(X) + H(X,Y,Z) \leq H(X,Y) + H(X,Z)$ (expanded version of $I(Y:Z|X) \geq 0$)
- the description of all true inequalities of this type (the dual cone to the set of entropy tuples) is an open difficult problem for > 3 variables
- Romashchenko: exactly the same inequalities are true for Kolmogorov complexities
- similar statement is true for combinatorial analogs (Yeung uniform sets, or splitting as explained above)

Shannon coding theorem

Why algorithmic information theory is natural?

Shannon coding theorem

Why algorithmic information theory is natural?

Shannon coding theorem

Why algorithmic information theory is natural?

- X: random variable with finite range

Shannon coding theorem

Why algorithmic information theory is natural?

- X: random variable with finite range

  $(X_1, \ldots, X_n)$ values of n independent copies of X

Shannon coding theorem

Why algorithmic information theory is natural?

- X: random variable with finite range

  $(X_1, \ldots, X_n)$ values of n independent copies of X

  want to encode $(X_1, \ldots, X_n)$ by m bits (so that decoding works with high probability)

Shannon coding theorem

Why algorithmic information theory is natural?

- X: random variable with finite range
  $(X_1, \ldots, X_n)$ values of n independent copies of X
  want to encode $(X_1, \ldots, X_n)$ by m bits (so that decoding works with high probability)
- Shannon: possible if $m \geq nH(X)$

Shannon coding theorem

Why algorithmic information theory is natural?

- X: random variable with finite range

  $(X_1, \ldots, X_n)$ values of n independent copies of X

  want to encode $(X_1, \ldots, X_n)$ by m bits (so that decoding works with high probability)
- Shannon: possible if $m \geq nH(X)$
- X is serialized, but encoding/decoding is arbitrary: half-way to algorithmic information theory

Shannon coding theorem

Why algorithmic information theory is natural?

- X: random variable with finite range
  $(X_1, \ldots, X_n)$ values of n independent copies of X
  want to encode $(X_1, \ldots, X_n)$ by m bits (so that decoding works with high probability)
- Shannon: possible if $m \geq nH(X)$
- X is serialized, but encoding/decoding is arbitrary: half-way to algorithmic information theory
- algorithmic reformulation: with high probability [under product distribution] the complexity of the string $(X_1, \ldots, X_n)$ is close to $nH(X)$

Multisource algorithmic information theory

Multisource algorithmic information theory

- Common information: X, Y two random variables

Multisource algorithmic information theory

- Common information: X, Y two random variables $(X_1, Y_1), \ldots, (X_n, Y_n)$: serialization

Multisource algorithmic information theory

- Common information: X, Y two random variables
  $(X_1, Y_1), \ldots, (X_n, Y_n)$: serialization
  we want to communicate $(X_1, \ldots, X_n)$ to Alice and
  $(Y_1, \ldots, Y_n)$ to Bob

Multisource algorithmic information theory

- Common information: X, Y two random variables
  $(X_1, Y_1), \ldots, (X_n, Y_n)$: serialization
  we want to communicate $(X_1, \ldots, X_n)$ to Alice and
  $(Y_1, \ldots, Y_n)$ to Bob
  by sending some common (broadcast) message to both, and
  two separate messages to Alice and Bob

Multisource algorithmic information theory

- Common information: X, Y two random variables
  $(X_1, Y_1), \ldots, (X_n, Y_n)$: serialization
  we want to communicate $(X_1, \ldots, X_n)$ to Alice and
  $(Y_1, \ldots, Y_n)$ to Bob
  by sending some common (broadcast) message to both, and two separate messages to Alice and Bob
  question: how long should be these messages? if the lengths are bounded by c (common), a and b (separate), what are the conditions on a, b, c that make this possible?

Multisource algorithmic information theory

- Common information: X, Y two random variables
  $(X_1, Y_1), \dots, (X_n, Y_n)$: serialization
  we want to communicate $(X_1, \dots, X_n)$ to Alice and
  $(Y_1, \dots, Y_n)$ to Bob
  by sending some common (broadcast) message to both, and two separate messages to Alice and Bob
  question: how long should be these messages? if the lengths are bounded by c (common), a and b (separate), what are the conditions on a, b, c that make this possible?
- necessary conditions $a + b + c \geq nH(X, Y)$, $a + c \geq H(X)$, $b + c \geq H(Y)$

Multisource algorithmic information theory

- Common information: X, Y two random variables
  $(X_1, Y_1), \ldots, (X_n, Y_n)$: serialization
  we want to communicate $(X_1, \ldots, X_n)$ to Alice and
  $(Y_1, \ldots, Y_n)$ to Bob
  by sending some common (broadcast) message to both, and two separate messages to Alice and Bob
  question: how long should be these messages? if the lengths are bounded by c (common), a and b (separate), what are the conditions on a, b, c that make this possible?
- necessary conditions $a + b + c \geq nH(X, Y)$, $a + c \geq H(X)$, $b + c \geq H(Y)$ are in general not sufficient

Multisource algorithmic information theory

- Common information: X, Y two random variables $(X_1, Y_1), \ldots, (X_n, Y_n)$: serialization
  we want to communicate $(X_1, \ldots, X_n)$ to Alice and $(Y_1, \ldots, Y_n)$ to Bob
  by sending some common (broadcast) message to both, and two separate messages to Alice and Bob
  question: how long should be these messages? if the lengths are bounded by c (common), a and b (separate), what are the conditions on a, b, c that make this possible?
- necessary conditions $a + b + c \geq nH(X, Y)$, $a + c \geq H(X)$, $b + c \geq H(Y)$ are in general not sufficient
- but why should be restrict ourselves to n independent copies? Let x, y be a random pair of incident point and line on a plane over $\mathbb{F}_p$. What is the (a, b, c)-profile of it?

- Common information: X, Y two random variables
  $(X_1, Y_1), \dots, (X_n, Y_n)$: serialization
  we want to communicate $(X_1, \dots, X_n)$ to Alice and
  $(Y_1, \dots, Y_n)$ to Bob
  by sending some common (broadcast) message to both, and two separate messages to Alice and Bob
  question: how long should be these messages? if the lengths are bounded by c (common), a and b (separate), what are the conditions on a, b, c that make this possible?
- necessary conditions $a+b+c \geq nH(X,Y)$, $a+c \geq H(X)$, $b+c \geq H(Y)$ are in general not sufficient
- but why should be restrict ourselves to n independent copies? Let x, y be a random pair of incident point and line on a plane over $\mathbb{F}_p$. What is the (a, b, c)-profile of it?
- a combinatorial question about covering of the set of incident pairs by combinatorial rectangles

Slepyan–Wolf (a special case) and Muchnik

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?
- SW: about $nH(Y|X)$ bits are necessary and sufficient

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?
- SW: about $nH(Y|X)$ bits are necessary and sufficient

  (Shannon achieves this if Alice knows X)

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?
- SW: about $nH(Y|X)$ bits are necessary and sufficient (Shannon achieves this if Alice knows X)
- algorithmic version: Alice knows string X; Bob knows string Y. A message M is needed such that

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?
- SW: about $nH(Y|X)$ bits are necessary and sufficient (Shannon achieves this if Alice knows X)
- algorithmic version: Alice knows string X; Bob knows string Y. A message M is needed such that (1) $C(M|X) \approx 0$ (the message M does not contain information that Alice does not have)

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?
- SW: about $nH(Y|X)$ bits are necessary and sufficient

  (Shannon achieves this if Alice knows X)
- algorithmic version: Alice knows string X; Bob knows string Y. A message M is needed such that (1) $C(M|X) \approx 0$ (the message M does not contain information that Alice does not have) and (2) $C(X|Y, M) \approx 0$ (the message M together with Y contain all the information needed to reconstruct X).

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?
- SW: about $nH(Y|X)$ bits are necessary and sufficient

  (Shannon achieves this if Alice knows X)
- algorithmic version: Alice knows string X; Bob knows string Y. A message M is needed such that (1) $C(M|X) \approx 0$ (the message M does not contain information that Alice does not have) and (2) $C(X|Y, M) \approx 0$ (the message M together with Y contain all the information needed to reconstruct X).
- what is the minimal length of such an M?

Slepyan–Wolf (a special case) and Muchnik

- a pair (X, Y) of dependent variables and n independent copies $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Alice knows $(X_1, \ldots, X_n)$; Bob known $(Y_1, \ldots, Y_n)$ and wants to know $(X_1, \ldots, X_n)$ too
- how many bits needs Alice to send to Bob?
- SW: about $nH(Y|X)$ bits are necessary and sufficient

  (Shannon achieves this if Alice knows X)
- algorithmic version: Alice knows string X; Bob knows string Y. A message M is needed such that (1) $C(M|X) \approx 0$ (the message M does not contain information that Alice does not have) and (2) $C(X|Y, M) \approx 0$ (the message M together with Y contain all the information needed to reconstruct X).
- what is the minimal length of such an M?
  Andrej Muchnik: about $C(Y|X)$ bits are necessary and sufficient. [Related to SW but not a corollary or vice versa]

Versions of algorithmic complexity

Several versions of “Kolmogorov complexity”:

Versions of algorithmic complexity

Several versions of "Kolmogorov complexity":

- plain complexity (as defined above, denoted sometimes by K, C, KS,...) [Solomonoff, Kolmogorov, Chaitin]

Versions of algorithmic complexity

Several versions of "Kolmogorov complexity":

- plain complexity (as defined above, denoted sometimes by K, C, KS,...) [Solomonoff, Kolmogorov, Chaitin]
- prefix complexity (only prefix decompressors are considered: their domain should not contain a string and its prefix at the same time; denoted sometimes by K, H, KP...) [Levin, Chaitin]

Versions of algorithmic complexity

Several versions of "Kolmogorov complexity":

- plain complexity (as defined above, denoted sometimes by K, C, KS,...) [Solomonoff, Kolmogorov, Chaitin]
- prefix complexity (only prefix decompressors are considered: their domain should not contain a string and its prefix at the same time; denoted sometimes by K, H, KP...) [Levin, Chaitin]
- decision complexity (the program does not produce x but can compute bit x_i for every given i; denoted sometimes by KR, KD,...) [Loveland]

Versions of algorithmic complexity

Several versions of "Kolmogorov complexity":

- plain complexity (as defined above, denoted sometimes by K, C, KS,...) [Solomonoff, Kolmogorov, Chaitin]
- prefix complexity (only prefix decompressors are considered: their domain should not contain a string and its prefix at the same time; denoted sometimes by K, H, KP...) [Levin, Chaitin]
- decision complexity (the program does not produce x but can compute bit x_i for every given i; denoted sometimes by KR, KD,...) [Loveland]
- monotone complexity (both the program and the output are considered as prefixes of infinite sequences, denoted sometimes by KM, Km,...) [Levin]

Versions of algorithmic complexity

Several versions of "Kolmogorov complexity":

- plain complexity (as defined above, denoted sometimes by K, C, KS,...) [Solomonoff, Kolmogorov, Chaitin]
- prefix complexity (only prefix decompressors are considered: their domain should not contain a string and its prefix at the same time; denoted sometimes by K, H, KP...) [Levin, Chaitin]
- decision complexity (the program does not produce x but can compute bit x_i for every given i; denoted sometimes by KR, KD,...) [Loveland]
- monotone complexity (both the program and the output are considered as prefixes of infinite sequences, denoted sometimes by KM, Km,...) [Levin]
- a priori probability (discrete and continuous; the first one leads to prefix complexity, the second one gives a new notion of complexity, sometimes denoted by KM, KA,...) [Levin, Chaitin]

So what?

Natural questions:

So what?

Natural questions:

- why so many versions?

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

Brief answers:

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

Brief answers:

- many versions since inputs and outputs can be considered with different structures (topologies): discrete and continuous (as prefixes of infinite sequences): this gives four versions

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

Brief answers:

- many versions since inputs and outputs can be considered with different structures (topologies): discrete and continuous (as prefixes of infinite sequences): this gives four versions (even eight for conditional complexities where also topology on conditions is important)

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

Brief answers:

- many versions since inputs and outputs can be considered with different structures (topologies): discrete and continuous (as prefixes of infinite sequences): this gives four versions (even eight for conditional complexities where also topology on conditions is important)
- not "right" versus "wrong", just different (different ones are more suitable in different cases)

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

Brief answers:

- many versions since inputs and outputs can be considered with different structures (topologies): discrete and continuous (as prefixes of infinite sequences): this gives four versions (even eight for conditional complexities where also topology on conditions is important)
- not "right" versus "wrong", just different (different ones are more suitable in different cases)
- information theorists: no need to care (only log-difference)

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

Brief answers:

- many versions since inputs and outputs can be considered with different structures (topologies): discrete and continuous (as prefixes of infinite sequences): this gives four versions (even eight for conditional complexities where also topology on conditions is important)
- not "right" versus "wrong", just different (different ones are more suitable in different cases)
- information theorists: no need to care (only log-difference)
  recursion theorists: yes, they do!

So what?

Natural questions:

- why so many versions?
- which versions is the right one (the best)?
- do we really care?

Brief answers:

- many versions since inputs and outputs can be considered with different structures (topologies): discrete and continuous (as prefixes of infinite sequences): this gives four versions (even eight for conditional complexities where also topology on conditions is important)
- not "right" versus "wrong", just different (different ones are more suitable in different cases)
- information theorists: no need to care (only log-difference)

  recursion theorists: yes, they do!

  the translation between a priori probability and complexity is of philosophical importance (and a technical tool)

Discrete a priori probability

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms
- randomized algorithm P without input: being started, outputs a natural number (or binary string: we identify them) and stops

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms
- randomized algorithm P without input: being started, outputs a natural number (or binary string: we identify them) and stops
- may hang (no output)

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms
- randomized algorithm P without input: being started, outputs a natural number (or binary string: we identify them) and stops
- may hang (no output)
- $p_i = \Pr[P \text{ outputs } i]$

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms
- randomized algorithm P without input: being started, outputs a natural number (or binary string: we identify them) and stops
- may hang (no output)
- $p_i = \Pr[P \text{ outputs } i]$
- $\sum_i p_i \leq 1$

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms
- randomized algorithm P without input: being started, outputs a natural number (or binary string: we identify them) and stops
- may hang (no output)
- $p_i = \Pr[P \text{ outputs } i]$
- $\sum_i p_i \leq 1$
- sum may be less than 1 if non-termination has positive probability

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms
- randomized algorithm P without input: being started, outputs a natural number (or binary string: we identify them) and stops
- may hang (no output)
- $p_i = \Pr[P \text{ outputs } i]$
- $\sum_i p_i \le 1$
- sum may be less than 1 if non-termination has positive probability
- p_i are "lower semicomputable" (can be approximated from below effectively)

Discrete a priori probability

- we considered arbitrary functions as decompessors, but then restricted ourselves to computable ones
- now we consider arbitrary distributions and then restrict ourselves to output distributions of randomized algorithms
- randomized algorithm P without input: being started, outputs a natural number (or binary string: we identify them) and stops
- may hang (no output)
- $p_i = \Pr[P \text{ outputs } i]$
- $\sum_i p_i \leq 1$
- sum may be less than 1 if non-termination has positive probability
- p_i are "lower semicomputable" (can be approximated from below effectively) and every lower semicomputable converging series with sum ≤ 1 is the output distribution of some P

Universal randomized algorithm and a priori distribution

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if

$$\exists\varepsilon\,\forall i\ \ \Pr[P' \text{ outputs } i] \geq \varepsilon \Pr[P \text{ outputs } i]$$

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if
$$\exists \varepsilon\, \forall i \;\; \Pr[P' \text{ outputs } i] \geq \varepsilon \Pr[P \text{ outputs } i]$$
- series $\sum p'_i$ is an upper bound for $\sum p_i$ (up to a constant)

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if
$$\exists\varepsilon\,\forall i\ \ \Pr[P' \text{ outputs } i] \geq \varepsilon \Pr[P \text{ outputs } i]$$
- series $\sum p'_i$ is an upper bound for $\sum p_i$ (up to a constant)
- universal (optimal, "most diverse") randomized algorithm: "choose a randomized algorithm at random and then simulate it"

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if
$$\exists \varepsilon\, \forall i \;\; \Pr[P' \text{ outputs } i] \ge \varepsilon \Pr[P \text{ outputs } i]$$
- series $\sum p'_i$ is an upper bound for $\sum p_i$ (up to a constant)
- universal (optimal, "most diverse") randomized algorithm: "choose a randomized algorithm at random and then simulate it"
- "biggest" ("least convergent") lower semicomputable series

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if
$$\exists\varepsilon\,\forall i\;\; \Pr[P' \text{ outputs } i] \ge \varepsilon \Pr[P \text{ outputs } i]$$
- series $\sum p'_i$ is an upper bound for $\sum p_i$ (up to a constant)
- universal (optimal, "most diverse") randomized algorithm: "choose a randomized algorithm at random and then simulate it"
- "biggest" ("least convergent") lower semicomputable series (weighted sum of all)

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if
$$\exists\varepsilon\,\forall i\ \ \Pr[P' \text{ outputs } i] \ge \varepsilon \Pr[P \text{ outputs } i]$$
- series $\sum p'_i$ is an upper bound for $\sum p_i$ (up to a constant)
- universal (optimal, "most diverse") randomized algorithm: "choose a randomized algorithm at random and then simulate it"
- "biggest" ("least convergent") lower semicomputable series (weighted sum of all)
- discrete a priori probability of i = the probability to get i as an output of some fixed universal randomized algorithm

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if
  $$\exists\varepsilon\,\forall i\ \ \Pr[P' \text{ outputs } i] \geq \varepsilon \Pr[P \text{ outputs } i]$$
- series $\sum p'_i$ is an upper bound for $\sum p_i$ (up to a constant)
- universal (optimal, "most diverse") randomized algorithm: "choose a randomized algorithm at random and then simulate it"
- "biggest" ("least convergent") lower semicomputable series (weighted sum of all)
- discrete a priori probability of i = the probability to get i as an output of some fixed universal randomized algorithm
- defined up to $O(1)$ factor

Universal randomized algorithm and a priori distribution

- let P and P' be two randomized algorithms of that type
- P' is "better" (more "diverse") if
$$\exists \varepsilon\, \forall i \;\; \Pr[P' \text{ outputs } i] \geq \varepsilon \Pr[P \text{ outputs } i]$$
- series $\sum p'_i$ is an upper bound for $\sum p_i$ (up to a constant)
- universal (optimal, "most diverse") randomized algorithm: "choose a randomized algorithm at random and then simulate it"
- "biggest" ("least convergent") lower semicomputable series (weighted sum of all)
- discrete a priori probability of i = the probability to get i as an output of some fixed universal randomized algorithm
- defined up to $O(1)$ factor
- denoted sometimes by $\mathbf{m}(i)$ (where i is an integer – or the corresponding string)

Discrete a priori probability and prefix complexity

Let $K(i)$ be a prefix complexity of an integer i (the length of the shortest "self-delimited" program that produces i

Discrete a priori probability and prefix complexity

Let $K(i)$ be a prefix complexity of an integer i (the length of the shortest "self-delimited" program that produces i

Theorem (Levin, Chaitin)

$$\mathbf{m}(i) = 2^{-K(i)+O(1)}$$

Discrete a priori probability and prefix complexity

Let $K(i)$ be a prefix complexity of an integer i (the length of the shortest "self-delimited" program that produces i

Theorem (Levin, Chaitin)

$$\mathbf{m}(i) = 2^{-K(i)+O(1)}$$

- an infinite algorithmic version of Kraft inequality for prefix codes

Discrete a priori probability and prefix complexity

Let $K(i)$ be a prefix complexity of an integer i (the length of the shortest "self-delimited" program that produces i

Theorem (Levin, Chaitin)

$$\mathbf{m}(i) = 2^{-K(i)+O(1)}$$

- an infinite algorithmic version of Kraft inequality for prefix codes
- relates two philosophically different properties of an object x:

Discrete a priori probability and prefix complexity

Let $K(i)$ be a prefix complexity of an integer i (the length of the shortest "self-delimited" program that produces i

Theorem (Levin, Chaitin)

$$\mathbf{m}(i) = 2^{-K(i)+O(1)}$$

- an infinite algorithmic version of Kraft inequality for prefix codes
- relates two philosophically different properties of an object x: (1) how difficult is an object x to describe (complexity)

Discrete a priori probability and prefix complexity

Let $K(i)$ be a prefix complexity of an integer i (the length of the shortest "self-delimited" program that produces i

Theorem (Levin, Chaitin)

$$\mathbf{m}(i) = 2^{-K(i)+O(1)}$$

- an infinite algorithmic version of Kraft inequality for prefix codes
- relates two philosophically different properties of an object x: (1) how difficult is an object x to describe (complexity) and (2) how plausible x is as an output of an unknown random process

Discrete a priori probability and prefix complexity

Let $K(i)$ be a prefix complexity of an integer i (the length of the shortest "self-delimited" program that produces i

Theorem (Levin, Chaitin)

$$\mathbf{m}(i) = 2^{-K(i)+O(1)}$$

- an infinite algorithmic version of Kraft inequality for prefix codes
- relates two philosophically different properties of an object x: (1) how difficult is an object x to describe (complexity) and (2) how plausible x is as an output of an unknown random process
- one direction ($\geq$) is easy: universal decompressor applied to a sequence of random bits is a random process, and if i has a program of length n, then the probability to bump into it is at least 2^{-n}

Game proof of Levin–Chaitin theorem

Game proof of Levin–Chaitin theorem

Game:

Game proof of Levin–Chaitin theorem

Game:

- two players A and B alternate

Game:

- two players A and B alternate
- B approximates from below the terms m_i of some converging series $\sum m_i \leq 1$ (at every step t giving some rational lower bounds $m_i[t]$ that increases with t; then m_i is defined as $\lim_{t\to\infty} m_i[t]$)

Game proof of Levin–Chaitin theorem

Game:

- two players A and B alternate
- B approximates from below the terms m_i of some converging series $\sum m_i \leq 1$ (at every step t giving some rational lower bounds $m_i[t]$ that increases with t; then m_i is defined as $\lim_{t\to\infty} m_i[t]$)
- A may declare at each step that some string x is a "description" for some integer i (in other words, A enumerates some pairs of type (string, integer));

Game proof of Levin–Chaitin theorem

Game:

- two players A and B alternate
- B approximates from below the terms m_i of some converging series $\sum m_i \le 1$ (at every step t giving some rational lower bounds $m_i[t]$ that increases with t; then m_i is defined as $\lim_{t\to\infty} m_i[t]$)
- A may declare at each step that some string x is a "description" for some integer i (in other words, A enumerates some pairs of type (string, integer)); strings appearing in these pairs should form a prefix-free set (one is not a prefix of another)

Game proof of Levin–Chaitin theorem

Game:

- two players A and B alternate
- B approximates from below the terms m_i of some converging series $\sum m_i \le 1$ (at every step t giving some rational lower bounds $m_i[t]$ that increases with t; then m_i is defined as $\lim_{t\to\infty} m_i[t]$)
- A may declare at each step that some string x is a "description" for some integer i (in other words, A enumerates some pairs of type (string, integer)); strings appearing in these pairs should form a prefix-free set (one is not a prefix of another)
- game is infinite; A wins in the limit if every i has description of size at most $\log(1/m_i) + 4$. (Here 4 is large enough constant.)

Game proof of Levin–Chaitin theorem

Game:

- two players A and B alternate
- B approximates from below the terms m_i of some converging series $\sum m_i \leq 1$ (at every step t giving some rational lower bounds $m_i[t]$ that increases with t; then m_i is defined as $\lim_{t\to\infty} m_i[t]$)
- A may declare at each step that some string x is a "description" for some integer i (in other words, A enumerates some pairs of type (string, integer)); strings appearing in these pairs should form a prefix-free set (one is not a prefix of another)
- game is infinite; A wins in the limit if every i has description of size at most $\log(1/m_i) + 4$. (Here 4 is large enough constant.)

Claim: A has a computable winning strategy.

Game proof – 2

Two questions:

Game proof – 2

Two questions:

- how A wins the game?

Game proof – 2

Two questions:

- how A wins the game?
- why it is enough for the proof of Levin–Chaitin theorem?

Game proof – 2

Two questions:

- how A wins the game?
- why it is enough for the proof of Levin–Chaitin theorem?

The second question: let A play against the approximations for a priori probability (recall it is lower semicomputable); the pairs generated by A form a graph of a computable prefix-free decompressor, so they provide a bound for prefix complexity

The first question (more technical):

Game proof – 2

Two questions:

- how A wins the game?
- why it is enough for the proof of Levin–Chaitin theorem?

The second question: let A play against the approximations for a priori probability (recall it is lower semicomputable); the pairs generated by A form a graph of a computable prefix-free decompressor, so they provide a bound for prefix complexity

The first question (more technical):

- assume w.l.o.g. that approximations $m_i[t]$ are all of the form 2^{-k} (we lose only some $O(1)$ factor);

Game proof – 2

Two questions:

- how A wins the game?
- why it is enough for the proof of Levin–Chaitin theorem?

The second question: let A play against the approximations for a priori probability (recall it is lower semicomputable); the pairs generated by A form a graph of a computable prefix-free decompressor, so they provide a bound for prefix complexity

The first question (more technical):

- assume w.l.o.g. that approximations $m_i[t]$ are all of the form 2^{-k} (we lose only some $O(1)$ factor);
- the sum of all these approximations is bounded by 2 (again a constant factor); divide them by 2;

Game proof – 2

Two questions:

- how A wins the game?
- why it is enough for the proof of Levin–Chaitin theorem?

The second question: let A play against the approximations for a priori probability (recall it is lower semicomputable); the pairs generated by A form a graph of a computable prefix-free decompressor, so they provide a bound for prefix complexity

The first question (more technical):

- assume w.l.o.g. that approximations $m_i[t]$ are all of the form 2^{-k} (we lose only some $O(1)$ factor);
- the sum of all these approximations is bounded by 2 (again a constant factor); divide them by 2;
- cover the interval $[0, 1]$ from left to right by the intervals of these lengths and then choose a maximal binary (Cantor space) interval inside.

in case we reached this place...

Thanks for the patience!

in case we reached this place...

Thanks for the patience!
textbook (Uspensky, Vereshchagin, S):
www.lirmm.fr/~ashen/kolmbook-eng.pdf