

Increasing the gap between monotonic descriptive complexity and algorithmic probability

Adam Day

Victoria University of Wellington

29 June 2009

Investigate why an analogue of the coding theorem fails in a continuous setting.

More specifically:

- Look at two complexity measures:
 - K_m – monotonic descriptive complexity, and
 - KM – a complexity derived from a maximal c.e. semimeasure.
- Provide some intuition to as to why these complexities differ.
- Highlight some technical improvements in the proof that significantly improve the lower bound between these two complexities.

The Coding Theorem

Definition

A computably enumerable discrete semimeasure is a c.e. function $d : 2^{<\omega} \rightarrow \mathbb{R}^{\geq 0}$ such that $\sum_{\sigma \in 2^{<\omega}} d(\sigma) \leq 1$.

Let \mathcal{D} the set of all c.e. discrete semimeasures. It has a maximal element. i.e. there exists $D \in \mathcal{D}$ such that for all $d \in \mathcal{D}$ there is a c such that: $D(\sigma) \geq cd(\sigma)$.

Let U be a universal prefix-free machine.

- D can be obtained by defining $D(\sigma) = \mu(\{[\tau] : U(\tau) = \sigma\})$.
- Let $K(\sigma)$ be the length of the shortest τ such that $U(\tau) = \sigma$.

Theorem (The coding theorem – Levin, Chaitin)

$$K(\sigma) = -\log D(\sigma) \pm O(1)$$

The complexity KM

Idea: Consider measures on 2^ω the space of infinite binary strings.

Definition

A c.e. continuous semimeasure is a computably enumerable function $m : 2^{<\omega} \rightarrow \mathbb{R}^{\geq 0}$ such that:

- $m(\lambda) \leq 1$
- $m(\sigma) \geq m(\sigma 0) + m(\sigma 1)$

Like the discrete case there is a maximal c.e. continuous semimeasure M .

Definition (Levin)

Define $KM(\sigma) = -\log M(\sigma)$

Question: Is there a machine characterization of KM ?

Monotone machines

One way of understanding a monotone machine is to consider a machine M that:

- Must maintain continuity i.e. if $\tau_1 \preceq \tau_2$, $M(\tau_1) \preceq M(\tau_2)$,
- Can refine descriptions e.g. if a stage s , $M_s(01) = 0011$ then at a later stage t , we can redefine it by $M_t(01) = 0011000$.

Definition (Levin)

A *monotone machine* L is a computably enumerable set of pairs of finite binary strings $\langle \tau, \sigma \rangle$ such that if $\langle \tau_1, \sigma_1 \rangle, \langle \tau_2, \sigma_2 \rangle \in L$ and $\tau_1 \preceq \tau_2$, then $\sigma_1 \preceq \sigma_2$ or $\sigma_2 \preceq \sigma_1$.

Domain of the universal monotone machine

000	001	010	011	100	101	110	111
00		01		10		11	
0				1			
λ							

Universal monotone machine example

000 1111	001 100	010	011 01	100 00	101	110 011	111 0
00 1	01	10	11 01				
0	1 0						
λ							

Monotone complexity

There exists a universal monotone machine U .

Definition (Levin)

The monotonic descriptonal complexity of a string $\sigma \in 2^{<\omega}$ is:

$$K_m(\sigma) = \min\{|\tau| : U(\tau) \succeq \sigma\}$$

where $U(\tau) \succeq \sigma$ means there exists $\langle \tau, \sigma' \rangle \in U$ and $\sigma' \succeq \sigma$.

A reason to think K_m and KM are strongly related.

- Define $m_U(\sigma) = \mu(\{[\tau] : U(\tau) \succeq \sigma\})$, the measure of *all* descriptions of σ .
- Then $-\log m_U(\sigma) = KM(\sigma)$ to within an additive constant.

Question: Could it be that $K_m(\sigma) = KM(\sigma) \pm O(1)$?

The difference

Theorem (upper-bound)

$$K_m(\sigma) \leq KM(\sigma) + K(|\sigma|) + O(1)$$

For the space $\mathbb{N}^{<\omega}$, Gács proved that this upper-bound is tight.

For $2^{<\omega}$:

Theorem (Gács)

There exists a computable unbounded function A^{-1} , where A^{-1} is some version of the inverse Ackermann function such that for infinitely many σ ,

$$K_m(\sigma) > KM(\sigma) + A^{-1}(|\sigma|)$$

Hence K_m and KM are different, but the upper-bound could potentially be greatly improved.

Want to show that $A^{-1}(|\sigma|)$ can be replaced by $c \log \log(|\sigma|)$

Establishing a gap between K_m and KM

Objective: Given n , build a c.e. continuous semimeasure m such that:

- $m(\lambda) \leq 2^{-n}$
- There is some string σ such that $K_m(\sigma) > -\log m(\sigma)$

Hence if $\hat{m}(\sigma) = m(\sigma)2^n$, $K_m(\sigma) > -\log \hat{m}(\sigma) + n$.

We will develop a basic strategy to make a small difference between K_m and KM and then amplify it.

Basic strategy:

- Pick a string σ and $i, r \in \mathbb{N}$ with $i \leq r$.
- Increase $m(\sigma)$ by increments of 2^{-i} .
- Stop when the opponent uses a string of length $\leq r$ to describe σ (or something good happens).

Note: Once $m(\sigma)$ reaches 2^{-r} , the opponent must ensure $K_m(\sigma) \leq r$ i.e. use a string of length $\leq r$ to describe σ .

Example 1, $i = 3, r = 1$

000 τ	001 ρ	010	011	100	101	110	111
00		01		10		11	
0				1			
λ							

Domain of U at stage 0, τ, ρ incomparable

Example 1

At stage 1 set $m(\sigma) = 2^{-3}$

000 τ	001 ρ	010 σ	011	100	101	110	111
00		01		10		11	
0				1			
λ							

Domain of U at stage 1.

Example 1

At stage 2 set $m(\sigma) = 2^{-2}$

000 τ	001 ρ	010 σ	011	100	101	110	111
00		01 σ	10		11		
0				1			
λ							

Domain of U at stage 2.

Example 1

At stage 4 set $m(\sigma) = 2^{-1}$

000 τ	001 ρ	010 σ	011	100	101	110	111
00		01 σ	10		11		
0				1 σ			
λ							

Domain of U at stage 4.

Example 2

000 τ	001	010 ρ	011	100	101	110	111
00		01		10		11	
0				1			
λ							

Domain of U at stage 0.

Example 2

At stage 1 set $m(\sigma) = 2^{-3}$

000 τ	001	010 ρ	011	100 σ	101	110	111
00		01		10		11	
0				1			
λ							

Domain of U at stage 1.

Example 2

At stage 2 set $m(\sigma) = 2^{-2}$

000 τ	001	010 ρ	011	100 σ	101	110	111
00		01		10 σ		11	
0				1			
λ							

Domain of U at stage 2.

Example 2

At stage 4 set $m(\sigma) = 2^{-1}$

000 τ	001	010 ρ	011	100 σ	101	110	111
00		01		10 σ		11	
0				1 σ			
λ							

Domain of U at stage 4.

Example 2

Stop at this point. Make use of 'reservation'!

000 τ	001	010 ρ	011	100 σ	101	110	111
00		01		10		11	
0				1			
λ							

Domain of U at stage 1.

Example 2

Stop at this point. Make use of 'reservation'!

000 τ	001	010 ρ	011	100 σ	101	110	111
00 reserved for τ	01 reserved for ρ		10 reserved for σ		11		
0				1 reserved for σ			
λ							

Domain of U at stage 1.

Example 2

Stop at this point. Make use of 'reservation'!

000 τ	001	010 ρ	011	100 σ	101	110	111
00 reserved for τ	01 reserved for ρ		10 reserved for σ		11		
0 reserved for $\tau \cap \rho$				1 reserved for σ			
λ reserved for $\tau \cap \rho \cap \sigma$							

Domain of U at stage 1.

The idea of a reservation was introduced by Gács in his original proof.

Definition

We say a string τ is reserved for σ at stage s if at this stage:

- Some string comparable with τ describes an extension of σ .
- No string comparable with τ describes a string incomparable with σ .

This means $\langle \tau, \sigma \rangle$ could be enumerated into the machine at stage s but $\langle \tau, \rho \rangle$ cannot if σ and ρ are incomparable.

Exploiting reservations

If a reservation of length r is created for a string σ , then can exploit it by:

- Only increasing the measure on strings incomparable with σ . (Strings incomparable)
- Only using increments of measure of size at least 2^{-r} . (Increments exceed reservations)

These are the **platinum rules**.

- In we follow these rules, the opponent can do nothing *useful* with the reservation.
- Hard to quantify precisely – in this talk informally refer to as the gain of the algorithm.

Definition

A $(\sigma, \text{incr} = i, \text{reserve} = r)$ -algorithm is the following: increase the measure on σ in increments of 2^{-i} until the opponent reserves a string of length r for σ .

Say we want to run a (σ_1, i_1, r_1) -algorithm, followed by a (σ_2, i_2, r_2) -algorithm then to meet the platinum rules we must have:

- Strings incomparable $\Rightarrow \sigma_2$ and σ_1 incomparable
- Increments exceed reservations $\Rightarrow 2^{-i_2} \geq 2^{-r_1}$ or $i_2 \leq r_1$

Question: How can we run the algorithm recursively to increase the gain made?

Assume a $(\sigma, \text{incr} = i, \text{reserve} = r)$ -algorithm always increases the measure on σ by 2^{-r} .

The amplification problem

Idea: Instead of increasing the measure on σ directly, we run the algorithm on an extension of σ .

Example: Say we want to run make $(\sigma, \text{incr}=4, \text{reserve}=2)$ -algorithm. Take four incomparable extensions of σ : $\sigma_1, \sigma_2, \sigma_3$ and σ_4 then:

- Run a $(\sigma_1, \text{incr}=6, \text{reserve}=4)$ -algorithm.
- Run a $(\sigma_2, 6, 4)$ -algorithm (if no σ reservation of length 2).
- Run a $(\sigma_3, 6, 4)$ -algorithm (if no σ reservation of length 2).
- Run a $(\sigma_4, 6, 4)$ -algorithm (if no σ reservation of length 2).

At end $m(\sigma) \geq 4 \cdot 2^{-4} = 2^{-2}$ so opponent must provide a reservation.

Problem: We have violated platinum rule 2, increments exceed reservations.

Run algorithm on sets of strings

Definition

Σ is a prefix-free set of strings. A $(\Sigma, \text{incr} = i, \text{reserve} = r)$ -algorithm is the following:

- At each iteration increase the measure on those elements of Σ without an r -reservation in increments of 2^{-i} .
- Stop when the opponent reserves a string of length r for some fixed proportion of the elements of Σ .

Claim

Take any σ and let $\Sigma = \{\sigma\tau : |\tau| = k\}$.

A $(\Sigma, i + k, r + k)$ -algorithm gives about the same benefit as a (σ, i, r) -algorithm.

Solving the amplification problem

Example: We want to make a $(\{\sigma\}, \text{incr}=4, \text{reserve}=2)$ -algorithm.

Define:

- $\Sigma_0 = \{\sigma 1 \tau : |\tau| = 6\}$, $\Sigma_1 = \{\sigma 0 1 \tau : |\tau| = 4\}$
- $\Sigma_2 = \{\sigma 0 0 1 \tau : |\tau| = 2\}$, $\Sigma_3 = \{\sigma 0 0 0 1\}$

Now:

- Run a $(\Sigma_0, \text{incr}=12, \text{reserve}=10)$ -algorithm.
- Run a $(\Sigma_1, 10, 8)$ -algorithm (if no σ reservation of length 2).
- Run a $(\Sigma_2, 8, 6)$ -algorithm (if no σ reservation of length 2).
- Run a $(\Sigma_3, 6, 4)$ -algorithm (if no σ reservation of length 2).

At end $m(\sigma) = 2^{-2}$ so opponent must find a string of length ≤ 2 to describe σ .

Call this a $(\{\sigma\}, \text{incr}=12, \text{reserve}=2, \text{level}=2)$ -algorithm.

Solving the amplification problem

- Platinum rules met – the benefits produced by $(\Sigma_1, 12, 10)$, $(\Sigma_2, 10, 8)$, $(\Sigma_3, 8, 6)$, $(\Sigma_4, 6, 4)$ algorithms do not overlap with each other.
- Can show benefits from these algorithm do not overlap too much with benefit from overall $(\{\sigma\}, 12, 2, 2)$ -algorithm.

Conclusion: It works.

Spending enough measure

How do we ensure the algorithm spends enough measure?

- Divide the algorithm into two phase: gain and spend
- Run gain phase to establish enough reservations
- Run spend phase to ensure enough measure allocated.

This can be achieved while still obeying the platinum rules but adds technical difficulty.

Main result

Analysing the length of strings used by the algorithm gives the following result:

Theorem

If $c \in \mathbb{R}$, and $c < 1$, then there exist infinitely many $\sigma \in 2^{<\omega}$ such that:

$$K_m(\sigma) > KM(\sigma) + c \log \log |\sigma|.$$

But best known upper-bound is still:

Theorem

$$K_m(\sigma) \leq KM(\sigma) + K(|\sigma|) + O(1) \leq KM(\sigma) + 2 \log(|\sigma|) + O(1)$$

So more room for improvement!!