Probabilistic algorithms in computability theory

Laurent Bienvenu (Laboratoire J-V Poncelet, Moscow)

Computability, Complexity and Randomness
Moscow, September 2013

1. Introduction

Motivation

mathoverflow

Questions Tags Users Badges Unanswered

Can randomness add computability?

13

3

I have been looking at Church's Thesis, which asserts that all intuitively computable functions are recursive. The definition of recursion does not allow for randomness, and some people have suggested exceptions to Church's Thesis based on generating random strings. For example, using randomness one can generate strings of arbitrarily high Kolmogorov complexity but this is not possible with recursion alone.

However, these exceptions are not true *functions*. They generate multiple outputs, which collectively have some property. A recursive function takes an inputs and outputs a single unique answer. So some people do not consider these random coin-flips to be true exceptions to Church's Thesis.

My question is whether it is possible to use randomness to get something which is still essentially deterministic like a function, but is non-recursive.

For example, if we had a sequence of functions $F_i^r(n)$, which are recursive functions relative to a random tape oracle r, which have the property that for some function $g(n)$, we have $F_i^r(n) = g(n)$ with probability approaching 1 as $n \to \infty$ (the probability taken over the random tapes). Furthermore, g would not be recursive itself.

Here, I am suggesting relativizing to r, rather than having r as an input, because one might need arbitrarily many random cells. This *could* be thought of as allowing to "intuitively compute" g.

Motivation

- It is obvious that by computable means only, one cannot generate something non-computable (duh).

Motivation

- It is obvious that by computable means only, one cannot generate something non-computable (duh).
- If we additionally have access to a source of randomness, can we achieve more than what we could by computable means alone?

Motivation

- It is obvious that by computable means only, one cannot generate something non-computable (duh).
- If we additionally have access to a source of randomness, can we achieve more than what we could by computable means alone?

Note: this is a computability-theoretic question. As usual in computability theory, the basic objects will be infinite binary sequences and computable = computable by Turing machine.

For the moment: source of randomness = source of (infinitely many) independent random bits with distribution (1/2,1/2).

The issue of reproducibility

There are essentially two ways to understand this question, depending on whether we care about reproducibility.

The issue of reproducibility

There are essentially two ways to understand this question, depending on whether we care about reproducibility.

If we do not, then the answer is trivial: take an infinite binary sequence x at random, and print x. With probability 1 you have generated a non-computable sequence! (there are only countably many computable sequences)

The issue of reproducibility

There are essentially two ways to understand this question, depending on whether we care about reproducibility.

If we do not, then the answer is trivial: take an infinite binary sequence x at random, and print x. With probability 1 you have generated a non-computable sequence! (there are only countably many computable sequences)

The issue is that if we were to repeat this process, we would almost surely obtain some $x' \neq x$.

The issue of reproducibility

So the next question is:

Is there a non-computable sequence x **which can be probabilistically computed with positive probability?**

By this we mean: is there an algorithm (machine) M with access to a random source r and such that

$$\mathbb{P}[M(r) = x] > 0?$$

The issue of reproducibility

This question is the computability-theoretic analogue of the celebrated open question of computational complexity:

$$\mathsf{BPP} \stackrel{?}{=} \mathsf{P}$$

Derandomization in computability

The answer is negative: one can always derandomize.

Derandomization in computability

The answer is negative: one can always derandomize.

Theorem (De Leeuwe, Moore, Shannon, Shapiro - Sacks)

If there is an algorithm (machine) M with access to a random source r such that $\mathbb{P}[M(r) = x] > 0$*, then x is a computable sequence.*

Derandomization in computability

The answer is negative: one can always derandomize.

Theorem (De Leeuwe, Moore, Shannon, Shapiro - Sacks)

If there is an algorithm (machine) M with access to a random source r such that $\mathbb{P}[M(r) = x] > 0$*, then x is a computable sequence.*

Looks like this theorem is the end of the story, but it is not.

2. When randomness helps

Solving problems probabilistically

Randomness can be useful to solve problems which have no computable solution.

Solving problems probabilistically

Randomness can be useful to solve problems which have no computable solution.

Setting: mass problems. Given a **set** $\mathcal{C}$ of infinite sequences, is it possible to get some element of $\mathcal{C}$:
- deterministically?
- probabilistically (but non-reproducibly)?

Solving problems probabilistically

Randomness can be useful to solve problems which have no computable solution.

Setting: mass problems. Given a **set** $\mathcal{C}$ of infinite sequences, is it possible to get some element of $\mathcal{C}$:
- deterministically?
- probabilistically (but non-reproducibly)?

There are some classes $\mathcal{C}$ for which the answer to the first question is **no**, and the second is **yes** (obvious example: $\mathcal{C}$ = set of non-computable sequences).

Solving problems probabilistically

A much less obvious example:

Theorem (Kurtz 1981 - Kautz 1991)

Let C *be the set of functions from* $\mathbb{N}$ *to* $\mathbb{N}$ *(encoded as binary sequences) which are dominated by no computable function. There exists a probabilistic algorithm M such that* $\mathbb{P}[M(r) \in C] > 0$.

[For the experts: this is a corollary of the fact that no two 2-random sequence is computably dominated]

Solving problems probabilistically

A much less obvious example:

Theorem (Kurtz 1981 - Kautz 1991)

Let C be the set of functions from $\mathbb{N}$ to $\mathbb{N}$ (encoded as binary sequences) which are dominated by no computable function. There exists a probabilistic algorithm M such that $\mathbb{P}[M(r) \in C] > 0$.

[For the experts: this is a corollary of the fact that no two 2-random sequence is computably dominated]

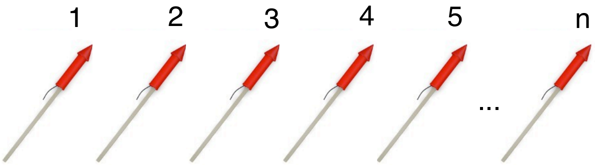
But what does the algorithm look like?

Fireworks

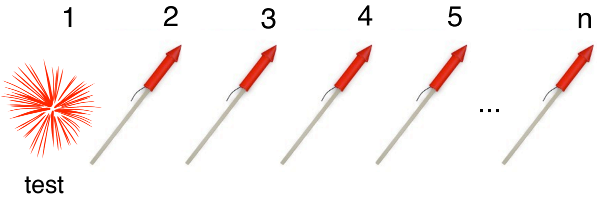
The algorithm was made more explicit by Gács and Shen (2012), using an amusing analogy: the **fireworks shop**.

- Suppose we walk into a fireworks shop.
- The fireworks sold there are very cheap so we are suspicious they could be defective.
- Since they are so cheap, we can ask the owner to test a few before buying one.
- Our goal: **either** buy a good one (untested) and take it home **or** get the owner to fail a test (and then sue him).

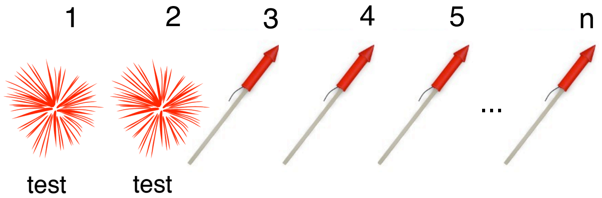
Fireworks



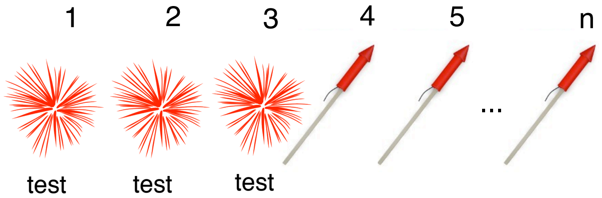
Fireworks



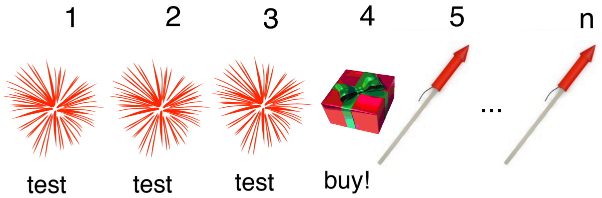
Fireworks



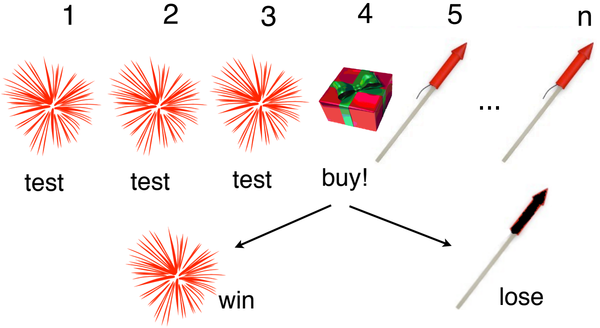
Fireworks



Fireworks



Fireworks



Fireworks

Clearly there is no deterministic strategy which works in all cases.

Fireworks

Clearly there is no deterministic strategy which works in all cases.

There is however a good probabilistic strategy, which wins with probability at least $\frac{n}{n+1}$ in all cases, where n is the number of fireworks:

Fireworks

Clearly there is no deterministic strategy which works in all cases.

There is however a good probabilistic strategy, which wins with probability at least $\frac{n}{n+1}$ in all cases, where n is the number of fireworks:

1. Pick a number k at random between 1 and $n+1$
2. Test the $k-1$ first fireworks
3. Buy the k-th one (unless $k=n+1$)

Fireworks

To see that the strategy succeeds with probability at least $\frac{n}{n+1}$, notice that the only bad case for us is when we pick the first bad one (convention: $(n+1)$-th fireworks is bad).

Fireworks

To see that the strategy succeeds with probability at least $\frac{n}{n+1}$, notice that the only bad case for us is when we pick the first bad one (convention: $(n+1)$-th fireworks is bad).

Indeed if we pick one before the first bad one, it will be good, so we win, and if we pick one after the first bad one, the first bad one will have failed the test.

Fireworks and the Kurtz-Kautz theorem

Back to the proof: we want to construct a function *f* which is not dominated by any (total) computable one.

Fireworks and the Kurtz-Kautz theorem

Back to the proof: we want to construct a function *f* which is not dominated by any (total) computable one.

First, list all the partial computable functions $\varphi_1, \varphi_2, \varphi_3, \ldots$

Fireworks and the Kurtz-Kautz theorem

Back to the proof: we want to construct a function *f* which is not dominated by any (total) computable one.

First, list all the partial computable functions $\varphi_1, \varphi_2, \varphi_3, \ldots$

We apply a fireworks strategy for each φ_i.

Fireworks and the Kurtz-Kautz theorem

- For each i, pick a number k_i between 1 and 2^{i+1}
- Repeat $k_i - 1$ times:
  - Pick the first fresh number w (on which f is not defined yet)
  - Define $f(w) = 0$
  - Test whether $\varphi_i(w)$ terminates. [While waiting, take care of other strategies for other φ_j].
- If previous loop terminates, pick a final fresh w, pause all other strategies, wait for $\varphi_i(w)$ to terminate, and if it does, define $f(w) = \varphi_i(w) + 1$.

Fireworks and the Kurtz-Kautz theorem

Just like in the fireworks game, the only bad case is when the $\varphi_i(w)$ is defined for all *w*'s picked during stage 2 (loop), and undefined on the *w* picked during stage 3 (final pick).

Fireworks and the Kurtz-Kautz theorem

Just like in the fireworks game, the only bad case is when the $\varphi_i(w)$ is defined for all w's picked during stage 2 (loop), and undefined on the w picked during stage 3 (final pick).

Thus with probability at least $1 - 2^{-i-1}$, we guarantee that f is not dominated by φ_i.

Fireworks and the Kurtz-Kautz theorem

Just like in the fireworks game, the only bad case is when the $\varphi_i(w)$ is defined for all *w*'s picked during stage 2 (loop), and undefined on the *w* picked during stage 3 (final pick).

Thus with probability at least $1 - 2^{-i-1}$, we guarantee that *f* is not dominated by φ_i.

Over all *i*, this gives a probability of success of at least $1 - \sum_i 2^{-i-1} > 0$.

Fireworks and the Kurtz-Kautz theorem

The fireworks technique is quite powerful, and can be used to solve even more difficult problems. For example, one can use the same argument to build a 1-generic.

[1-generic = infinite binary sequence which meets or (strongly) avoids every c.e. set of finite strings.]

Fireworks and the Kurtz-Kautz theorem

What have we gained?

Fireworks and the Kurtz-Kautz theorem

What have we gained?

- An intuitive understanding of the construction

Fireworks and the Kurtz-Kautz theorem

What have we gained?

- An intuitive understanding of the construction
- ... which allows, via a careful analysis, to strengthen Kautz's original results and solve open questions:

Theorem (Bienvenu, Porter)

Every Demuth random computes a 1-generic.

The typical Turing degree

A deeper phenomenon has been discovered recently by Barmpalias, Day and Lewis.

Theorem (Barmpalias, Day, Lewis)

Let T be a Turing functional. With probability 1 over x:
- either $T(x)$ is undefined
- or $T(x)$ is computable
- or $T(x)$ computes a 1-generic

The typical Turing degree

A deeper phenomenon has been discovered recently by Barmpalias, Day and Lewis.

Theorem (Barmpalias, Day, Lewis)

Let T be a Turing functional. With probability 1 over x:
- either $T(x)$ is undefined
- or $T(x)$ is computable
- or $T(x)$ computes a 1-generic

[For the experts: it suffices to take x 2-random]

The typical Turing degree

The hidden reason behind this is that when $T(x)$ is not computable, it still contains some randomness:

- Lemma (Bienvenu, Porter / Folklore?) If x is randomly distributed, $T(x)$ [up to small modification] follows a **0'**-computable (=limit computable) probability distribution.
- Kautz-Levin-Demuth: if a probability distribution is computable, then one can extract pure (uniform) randomness from it (modulo atoms).
- Here, the distribution is merely **0'**-computable, so we can only extract randomness in a limit-computable way.

The typical Turing degree

This is like playing the fireworks game with a “limit coin”... and it is enough! Just restart the game if the coin changes its value.

The typical Turing degree

This is like playing the fireworks game with a "limit coin"... and it is enough! Just restart the game if the coin changes its value.

Therefore:

Theorem (Bienvenu, Porter)

If x is distributed according to a $\mathbf{0'}$-computable distribution μ, then μ-almost surely, x is either an atom of μ, or can be used to compute a 1-generic.

Random algorithms and math. theorems

Random algorithms and math. theorems

Consider a mathematical theorem of type

$$\forall X \, \exists Y \, \Phi(X, Y)$$

where X and Y can be encoded as infinite binary sequences.

Random algorithms and math. theorems

Consider a mathematical theorem of type

$$\forall X \, \exists Y \, \Phi(X, Y)$$

where X and Y can be encoded as infinite binary sequences.

Examples:

- Bolzano-Weierstrass: For every sequence of reals in $[0, 1]$, there exists a converging subsequence.
- König's lemma: Every finitely branching tree with infinitely many nodes has an infinite path.
- Ramsey's theorem: For every coloring of the pairs of integers with k colors, there exists an infinite, monochromatic, set of integers.
- ...

Random algorithms and math. theorems

This gives rise to natural mass problems: for a given X, consider the problem

$$\mathcal{C}_X = \{Y \mid \Phi(X, Y)\}$$

Random algorithms and math. theorems

This gives rise to natural mass problems: for a given X, consider the problem

$$\mathcal{C}_X = \{Y \mid \Phi(X,Y)\}$$

Question: when X is computable, can one generate an element of $\mathcal{C}_X$:

- deterministically?
- probabilistically?

Random algorithms and math. theorems

Once there **are** problems for which no computable solution exists but such a solution can be found probabilstically:

Rainbow Ramsey Theorem: for all coloring of pairs of integers with possibly infinitely many colors, such that every color is used at most k times, there exists an infinite subset of $\mathbb{N}$ which is a rainbow (no color appears more than once).

Random algorithms and math. theorems

Once there **are** problems for which no computable solution exists but such a solution can be found probabilstically:

Rainbow Ramsey Theorem: for all coloring of pairs of integers with possibly infinitely many colors, such that every color is used at most k times, there exists an infinite subset of $\mathbb{N}$ which is a rainbow (no color appears more than once).

There is a computable coloring for which there is no computable rainbow (easy), but...

Theorem (Csima-Mileti, 2009)

Given a computable coloring, there exists a probabilistic algorithm which produces an infinite rainbow with positive probability.

3. When randomness does not help

Randomness does not help... often

As one might expect, randomness does not often help: when a problem has no computable solution, it is usually the case that one cannot generate a solution with a probabilistic algorithm.

Randomness does not help... often

As one might expect, randomness does not often help: when a problem has no computable solution, it is usually the case that one cannot generate a solution with a probabilistic algorithm.

Perhaps one of the most famous mass problems is the set of consistent completions of Peano arithmetic. We know from Gödel's theorem than there is no computable such object, and:

Randomness does not help... often

As one might expect, randomness does not often help: when a problem has no computable solution, it is usually the case that one cannot generate a solution with a probabilistic algorithm.

Perhaps one of the most famous mass problems is the set of consistent completions of Peano arithmetic. We know from Gödel's theorem than there is no computable such object, and:

Theorem (Jockusch, Soare, 1972)

No probabilistic algorithm can generate a consistent completion of PA.

Randomness does not help... often

Another interesting example: shift-complex sequences. Levin showed that there exists an infinite binary sequence X and a constant c such that for every substring σ of X,

$$\mathrm{K}(\sigma) \geq 0.99|\sigma| - c$$

Randomness does not help... often

Another interesting example: shift-complex sequences. Levin showed that there exists an infinite binary sequence X and a constant c such that for every substring σ of X,

$$\mathrm{K}(\sigma) \geq 0.99|\sigma| - c$$

Theorem (Rumyantsev, 2011)

No probabilistic algorithm can generate a shift-complex sequence (not even for any positive α instead of 0.99).

Wait and defeat

There is a common idea to prove such theorems: a “wait and defeat” strategy, which is the dual of a majority vote argument:

Wait and defeat

There is a common idea to prove such theorems: a “wait and defeat” strategy, which is the dual of a majority vote argument:

- Run the probabilistic algorithm to see how different oracles “vote”

Wait and defeat

There is a common idea to prove such theorems: a “wait and defeat” strategy, which is the dual of a majority vote argument:

- Run the probabilistic algorithm to see how different oracles “vote”
- Then use the universality of the class to defeat the majority of voters

Random algorithms and math. theorems

The same can be done for mathematical theorems
(theorem $\forall X\, \exists Y\, \Phi(X, Y) \longrightarrow$ class $\mathcal{C}_X$).

Random algorithms and math. theorems

The same can be done for mathematical theorems (theorem $\forall X\, \exists Y\, \Phi(X, Y) \longrightarrow$ class $\mathcal{C}_X$).

Bienvenu, Patey, Shafer: many, many examples of theorems such that there is no probabilistic algorithm for $\mathcal{C}_X$. See Ludovic's talk...

Random algorithms and math. theorems

The same can be done for mathematical theorems
(theorem $\forall X\, \exists Y\, \Phi(X, Y) \longrightarrow$ class $\mathcal{C}_X$).

Bienvenu, Patey, Shafer: many, many examples of theorems such that there is no probabilistic algorithm for $\mathcal{C}_X$. See Ludovic's talk...

For such results, due to (in general) the lack of universality, one needs to diagonalize against **all** probabilistic algorithms. For this, one arranges the wait-and-defeat technique in a priority construction (most of the time with finite injury).

Other directions

Other directions

- Deep Π^0_1 classes (Bienvenu, Porter, Taveneaux)

Other directions

- Deep Π^0_1 classes (Bienvenu, Porter, Taveneaux)
- What is a typical outcome of a probabilistic algorithm? (randomness w.r.t. to lower-semicomputable semimeasures)

Other directions

- Deep Π^0_1 classes (Bienvenu, Porter, Taveneaux)
- What is a typical outcome of a probabilistic algorithm? (randomness w.r.t. to lower-semicomputable semimeasures)
- Invariant degrees (V'Yugin, Levin – recent work by Hölzl and Porter)