Probablistic proofs of the existence of computable objects

Andrei Rumyantsev, Alexander Shen
Moscow State University, LIRMM CNRS, ИППИ РАН
Supported in part by NAFIT ANR project

Philosophy

Philosophy

- Probabilistic proof: we show that some property is true for a random object with positive probability, and conclude that objects with this property do exist

Philosophy

- Probabilistic proof: we show that some property is true for a random object with positive probability, and conclude that objects with this property do exist
- Nonconstructive existence proof

Philosophy

- Probabilistic proof: we show that some property is true for a random object with positive probability, and conclude that objects with this property do exist
- Nonconstructive existence proof
- Can we prove in this way the existence of a computable object with some property?

Philosophy

- Probabilistic proof: we show that some property is true for a random object with positive probability, and conclude that objects with this property do exist
- Nonconstructive existence proof
- Can we prove in this way the existence of a computable object with some property?
- Yes (in a sense)

Probabilistic proof: an example

Probabilistic proof: an example

- 0/1 $n \times n$ matrices

Probabilistic proof: an example

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected

Probabilistic proof: an example

- $0/1$ $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones

Probabilistic proof: an example

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors

Probabilistic proof: an example

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors
- random matrix: all bits are independent fair coin tossings

Probabilistic proof: an example

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors
- random matrix: all bits are independent fair coin tossings
- given minor is uniform with probability $2 \cdot 2^{-k^2}$

Probabilistic proof: an example

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors
- random matrix: all bits are independent fair coin tossings
- given minor is uniform with probability $2 \cdot 2^{-k^2}$
- at most $n^k = 2^{k \log n}$ ways to select rows/columns

Probabilistic proof: an example

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors
- random matrix: all bits are independent fair coin tossings
- given minor is uniform with probability $2 \cdot 2^{-k^2}$
- at most $n^k = 2^{k \log n}$ ways to select rows/columns
- at most $2^{2k \log n}$ ways to select minors

Probabilistic proof: an example

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors
- random matrix: all bits are independent fair coin tossings
- given minor is uniform with probability $2 \cdot 2^{-k^2}$
- at most $n^k = 2^{k \log n}$ ways to select rows/columns
- at most $2^{2k \log n}$ ways to select minors
- condition: $k^2 - 1 > 2k \log n$

General scheme of a probabilistic proof

General scheme of a probabilistic proof

- Random process (a machine with random bit generator)

General scheme of a probabilistic proof

- Random process (a machine with random bit generator)
- generates objects according to some distribution

General scheme of a probabilistic proof

- Random process (a machine with random bit generator)
- generates objects according to some distribution
- we prove that the probability to get a “bad” object is strictly less than 1

General scheme of a probabilistic proof

- Random process (a machine with random bit generator)
- generates objects according to some distribution
- we prove that the probability to get a “bad” object is strictly less than 1
- conclusion: good objects exist

General scheme of a probabilistic proof

- Random process (a machine with random bit generator)
- generates objects according to some distribution
- we prove that the probability to get a "bad" object is strictly less than 1
- conclusion: good objects exist

To speak about computability, we need infinite objects

General scheme of a probabilistic proof

- Random process (a machine with random bit generator)
- generates objects according to some distribution
- we prove that the probability to get a “bad” object is strictly less than 1
- conclusion: good objects exist

To speak about computability, we need infinite objects (binary sequences)

Randomized algorithm and its output distribution

Randomized algorithm and its output distribution

- Machine M has access to fair coin

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$
- $m(x) \geq m(x0) + m(x1)$ for all binary strings x

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$
- $m(x) \geq m(x0) + m(x1)$ for all binary strings x
- any m with these properties corresponds to some M

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$
- $m(x) \geq m(x0) + m(x1)$ for all binary strings x
- any m with these properties corresponds to some M
- measures $m(x) = m(x0) + m(x1)$ correspond to machines that generate infinite sequences almost surely

Existence of computable objects

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable
Proof:

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$
- each element of this set can be extended uniquely (or cannot be extended at all)

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$
- each element of this set can be extended uniquely (or cannot be extended at all)
- ω can be reconstructed starting from its prefix in the set

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$
- each element of this set can be extended uniquely (or cannot be extended at all)
- ω can be reconstructed starting from its prefix in the set

Probably not very useful in proving the existence of computable objects

Existence of computable objects II

Existence of computable objects II

- closed set in the Cantor space

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free
- not closed: computable or non-computable

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free
- not closed: computable or non-computable
- If some randomized machine M with probability 1 generates a sequence in some closed set S, then S contains a computable element

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free
- not closed: computable or non-computable
- If some randomized machine M with probability 1 generates a sequence in some closed set S, then S contains a computable element
- proof: construct ω bit by bit in such a way that each prefix of ω has positive probability

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free
- not closed: computable or non-computable
- If some randomized machine M with probability 1 generates a sequence in some closed set S, then S contains a computable element
- proof: construct ω bit by bit in such a way that each prefix of ω has positive probability

This will be used but some more general machines are needed

Lovasz local lemma (special case)

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-2} neighbors, then CNF is satisfiable

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-2} neighbors, then CNF is satisfiable
- neighbors: clauses that have common variables

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-2} neighbors, then CNF is satisfiable
- neighbors: clauses that have common variables
- classical proof uses induction to prove some bound on conditional probabilities

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-2} neighbors, then CNF is satisfiable
- neighbors: clauses that have common variables
- classical proof uses induction to prove some bound on conditional probabilities
- Moser's proof that uses Kolmogorov complexity

Infinite Lovasz local lemma

Infinite Lovasz local lemma

- countably many variables

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them
- and has at most 2^{m-2} neighbors

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down i-th clause

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause
- and lists all clauses that involve *j*-th variable

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause
- and lists all clauses that involve *j*-th variable
- Computable LLL: such a CNF has a computable satisfying assignment

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down i-th clause
- and lists all clauses that involve j-th variable
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set;

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause
- and lists all clauses that involve *j*-th variable
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set; it is enough to construct a machine that generates satisfying assignments with probability 1;

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause
- and lists all clauses that involve *j*-th variable
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set; it is enough to construct a machine that generates satisfying assignments with probability 1; such a machine can be extracted from Moser–Tardos algorithm for finding a solution for finite LLL;

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-2} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause
- and lists all clauses that involve *j*-th variable
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set; it is enough to construct a machine that generates satisfying assignments with probability 1; such a machine can be extracted from Moser–Tardos algorithm for finding a solution for finite LLL; but this is *rewriting* machine

Rewriting machines

Rewriting machines

- Machine has a random bit generator

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε
- output distribution is still computable:

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε
- output distribution is still computable:
- $m(x)$ = the probability that output starts with x

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε
- output distribution is still computable:
- $m(x)$ = the probability that output starts with x
- can be computed with any given precision

Rewriting machines

- Machine has a random bit generator
- and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε
- output distribution is still computable:
- $m(x)$ = the probability that output starts with x
- can be computed with any given precision
- paradox: the same class of distributions

so it is enough to construct a rewriting machine that solves LLL with probability 1

Moser–Tardos probabilistic machine

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1
- they give an estimate for convergence speed

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1
- they give an estimate for convergence speed
- so $N(i, \varepsilon)$ can be computed

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1
- they give an estimate for convergence speed
- so $N(i, \varepsilon)$ can be computed
- Q.E.D.

Remarks

Remarks

- Breakthrough: Moser–Tardos algorithm

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm
- layerwise computable mappings = almost everywhere defined mappings that correspond to rewriting machines with effective convergence

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm
- layerwise computable mappings = almost everywhere defined mappings that correspond to rewriting machines with effective convergence
- algorithmic randomness approach: layerwise computable mapping can be computed given the sequence and an upper bound for its randomness deficiency (Hoyrup, Rojas)

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm
- layerwise computable mappings = almost everywhere defined mappings that correspond to rewriting machines with effective convergence
- algorithmic randomness approach: layerwise computable mapping can be computed given the sequence and an upper bound for its randomness deficiency (Hoyrup, Rojas)
- Another application: let $\alpha < 1$ and let A be a decidable set of strings that contains at most $2^{\alpha n}$ strings of length n; then there exists a computable sequence α and c such that α has no α-factors longer than c.