Probabilistic Constructions of Computable Objects and a Computable Version of Lovász Local Lemma.

Andrei Rumyantsev, Alexander Shen*

January 17, 2013

Abstract

A nonconstructive proof can be used to prove the existence of an object with some properties without providing an explicit example of such an object. A special case is a probabilistic proof where we show that an object with required properties appears with some positive probability in some random process. Can we use such arguments to prove the existence of a *computable* infinite object? Sometimes yes: following [8], we show how the notion of a layerwise computable mapping can be used to prove a computable version of Lovász local lemma.

1 Probabilistic generation of infinite sequences

We want to show that one can use probabilistic arguments to prove the existence of inifinite objects (say, infinite sequences of zeros and ones) with some properties. So we should discuss first in which sense a probabilistic algorithm can generate such a sequence. The most natural approach: consider a Turing machine that has an output write-only tape where it can print bits sequentially. Using fair coin tosses, the machine generates a (finite or infinite) sequence of output bits. The output distribution of such a machine T is some probability distribution Q on the set of all finite and infinite sequences. Distributions of this class are determined by their values on cones Σ_x (of all finite and infinite extensions of a binary string x). The function $q(x) = Q(\Sigma_x)$ that corresponds to the measure Q satisfies the following conditions:

$$q(\Lambda) = 1; \quad q(x) \geq q(x0) + q(x1) \text{ for all strings } x.$$

(Here Λ denotes the empty string.) Any non-negative real function that satisfies these conditions corresponds to some probability distribution on the set of finite and infinite binary sequences. The output distributions of probabilistic machines correspond to functions q that are *lower semicomputable*; this means that some algorithm, given a binary string x, computes a non-decreasing sequence of rational numbers that converges to $q(x)$.

Now we are ready to look at the classical result of de Leeuw – Moore – Shannon – Shapiro: *if some individual sequence ω appears with positive probability as an output of a probabilistic Turing machine, this sequence is computable.* Indeed, let this probability be

*LIRMM CNRS & UM 2, Montpellier; on leave from IITP RAS, Moscow. Supported by ANR NAFIT-008-01/2 grant. E-mail: azrumuyan@gmail.com, alexander.shen@lirmm.fr

some $\varepsilon > 0$; take a rational threshold r such that $r < \varepsilon < 2r$, and consider some prefix w of ω such that $q(w) < 2r$. (Such a prefix exists, since $q(x)$ for prefixes of ω converges to ε.) Starting from w, we can compute the next prefix of ω by finding a son where q exceeds r. The correct son satisfies this condition, and no branching is possible: if for two sons the value exceeds r, then it should exceed $2r$ for the father.

This result can be interpreted as follows: *if our task is to produce some specific infinite sequence of zeros and ones, randomization does not help* (at least if we ignore the computational resources). However, if our goal is to produce *some* sequence with required properties, randomization can help. A trivial example: to produce a noncomputable sequence with probability 1 it is enough to output the random bits.

All these observations are well known, see, e.g., the classical paper of Zvonkin and Levin [9]. For a less trivial example, let us consider another result (proved by N.V. Petri) mentioned in this paper: *there exists a probabilistic machine that with positive probability generates a sequence ω such that* (1) *ω contains infinitely many ones;* (2) *the function $i \mapsto$ the position of i-th one in ω has no computable upper bound.* (In the language of recursion theory, this machine generates with positive probability a characteristic sequence of a hyperimmune set.) A nice proof of this result was given by Peter Gacs; it is reproduced in the next section (we thank M. Bondarko for some improvements in the presentation).

2 Fireworks and hyperimmune sequences

Consider the following "real-life" problem. We come to a shop where fireworks are sold. After we buy one, we can test it in place (then we know whether it was good or not, but it is not usable anymore, so we have to buy a new one after that), or go home, taking the untested firework with us. We look for a probabilistic strategy that with 99% probability either finds a bad firework during the testing (so we can sue the shop and forget about the fireworks) or takes home a good one.

Solution: take a random number k in 0..99 range, make k tests (less if the failure happens earlier); if all k tested fireworks were good, take home the next one. The seller does not get any information from our behavior: he sees only that we are buying and testing the fireworks; when we take the next one home instead of testing, it is too late for him to do anything. So his strategy is reduced to choosing some number K of good fireworks sold before the bad one. He wins only if $K = k$, i.e., with probability at most 1%.

Another description of the same strategy: we take the first firework home with probability 1/100 and test it with probability 99/100; if it is good, we take the second one with probability 1/99 and test it with probability 98/99, etc.

Why this game is relevant? Assume we have a program of some computable function f and want to construct probabilistically a total function g not bounded by f if f is total. (It is convenient to consider a machine that constructs a total integer-valued function and then use this function to determine the numbers of zeros between consecutive ones.) We consider $f(0), f(1), \ldots$ as "fireworks"; $f(i)$ is a good one if computation $f(i)$ terminates. First we buy $f(0)$; with probability 1/100 we "take" it and with probability 99/100 we "test" it. *Taking* $f(0)$ means that we run this computation until it terminates and then let $g(0) := f(0) + 1$. If this happens, we may relax and let all the other values of g be zeros. (If the computation does not terminate, i.e., if we take a bad firework, we are out of luck.) *Testing* $f(0)$ means that we run this computation and at the same time let $g(0) := 0$, $g(1) := 0$, etc. until the computation terminates. If $f(0)$ is undefined, g will be zero function, and this is OK since we do not care about non-total f. But if $f(0)$ is defined,

at some point testing stops, we have some initial fragment of zeros $g(0), g(1), \ldots, g(u)$, and then consider $f(u+1)$ as the next firework bought and test [take] it with probability 98/99 [resp. 1/99]. For example, if we decide to test it, we run the computation $f(u+1)$ until it terminates, and then let $g(u+1) := f(u+1) + 1$. And so on.

In this way we can beat one computable function f with probability arbitrarily close to 1. To construct with positive probability a function not bounded by *any* total computable function, we consider all the functions as functions of two natural arguments (tables), and use ith row to beat ith potential upper bound with probability $1 - \varepsilon 2^{-i}$. To beat the upper bound, it is enough to beat it in some row, so we can deal with all the rows in parallel, and get error probability at most $\varepsilon \sum_i 2^{-i} = \varepsilon$.

3 Computable elements of closed sets

Let us return now to the original question: can we use probabilistic arguments to construct a computable sequence with some property? As we have seen, if we are able to construct a probabilistic machine that generates some *specific* sequence with positive probability, we can then conclude that this specific sequence is computable. However, we do not know arguments that follow this scheme, and it is difficult to imagine how one can describe a specific sequence that it is actually computable, and prove that it has positive probability — without actually constructing an algorithm that computes it.

Here is another statement that may be easier to apply:

Proposition 1. *If F is some closed set of infinite sequences, and if output of some probabilistic machine belongs to F with probability 1, then F contains a computable element.*

Proof. Indeed, consider the output distribution and take a computable branch along which the probabilities remains positive (this is possible since the measure function is lower semicomputable). We get some computable sequence ω. If $\omega \notin F$, then some prefix of ω has no extensions in F (recall that F is closed). This prefix has positive probability by construction, so our machine cannot generate elements in F with probability 1. This contradiction shows that $\omega \in F$. □

In the following sections we give a specific example when this approach (in a significantly modified form) can be used.

4 Lovász Local Lemma

Let $\mathcal{X}$ be a sequence of mutually independent random variables; each variable has a finite range. (In the simplest case x_i are independent random bits.) Consider some family $\mathcal{A}$ of *forbidden events*; each of them depends on a finite set of variables, denoted $\mathrm{vbl}(A)$ (for an event A). Informally speaking, Lovász Local Lemma (LLL) guarantees that forbidden events do not cover the entire probability space if each of them has small probability and the dependence between the events (caused by common variables) is limited.

To make the statement exact, we need to introduce some terminology and notation. Two events A and B are *disjoint* if they do not share variables, i.e., if $\mathrm{vbl}(A) \cap \mathrm{vbl}(B) = \varnothing$. For every $A \in \mathcal{A}$ let $\Gamma(A)$ be the open (punctured) neighborhood of A, i.e., the set of all events $E \in \mathcal{A}$ that have common variables with A, except for A itself.

Proposition 2 (Finite version of LLL, [1]). *Consider a finite family $\mathcal{A}$ of forbidden events. Assume that for every event $A \in \mathcal{A}$ a rational number $x(A) \in (0,1)$ is fixed such that*

$$\Pr[A] \le x(A) \prod_{E \in \Gamma(A)} (1 - x(E)),$$

for all $A \in \mathcal{A}$. Then

$$\Pr[\textit{none of the forbidden events happens}] \ge \prod_{A \in \mathcal{A}} (1 - x(A)) \qquad (*)$$

This bound can be proved by an induction argument; note that the product in the right hand side of $(*)$ is positive (though can be very small), and therefore there exists an assignment that avoids all the forbidden events. This existence result can be extended to infinite families:

Proposition 3 (Infinite version of LLL). *Consider an infinite family $\mathcal{A}$ of forbidden events. Assume that each $A \in \mathcal{A}$ has only finitely many neighbors in $\mathcal{A}$, and that for every event $A \in \mathcal{A}$ a rational number $x(A) \in (0,1)$ is fixed such that*

$$\Pr[A] \le x(A) \prod_{E \in \Gamma(A)} (1 - x(E)),$$

for all $A \in \mathcal{A}$. Then there exists an assignment that avoids all the forbidden events $A \in \mathcal{A}$.

Proof. This is just a combination of finite LLL and compactness argument. Indeed, each event from $\mathcal{A}$ is open the the product topology; if the claim is false, these events cover the entire (compact) product space, so there exists a finite subset of events that covers the entire space, which contradicts the finite LLL. □

Our goal is to make this theorem effective. To formulate the computable version of LLL, assume that we have a countable sequence of independent random variables $\mathcal{X} = x_0, x_1, \ldots$, the range of x_i is $\{0, 1, \ldots, n_i - 1\}$, and n_i and the probability distribution of x_i is computable given i. Then we consider a sequence of events, $\mathcal{A} = \{A_0, A_1, \ldots\}$. We assume that these events are effectively presented, i.e., for a given j one can compute the list of all the variables in $\mathrm{vbl}(A_j)$, and the event itself (i.e., the list of tuples that belong to that event). Moreover, we assume that for each variable x_i only finitely many events involve this variable, and the list of those events can be computed given i.

Theorem 1 (Computable version of LLL). *Suppose there is a rational constant $\varepsilon \in (0,1)$ and a computable assignment of rational numbers $x : \mathcal{A} \to (0,1)$ such that*

$$\Pr[A] \le (1 - \varepsilon) x(A) \prod_{E \in \Gamma(A)} (1 - x(E)),$$

for all $A \in \mathcal{A}$. Then there exists a computable assignment that avoids all $A \in \mathcal{A}$.

Note that the computability restrictions look quite naturally and that we only need to make the upper bounds for probability just a bit stronger (adding some constant factor $1 - \varepsilon$). It should not be a problem for typical applications of LLL; usually this stronger bound on $\Pr[A]$ can be easily established.

This theorem is the main result of the paper (it was published as an `arxiv` prepint [8]). To prove it we use Proposition 1 and construct a computable probability distribution on

infinite sequences such that every forbidden event $A \in \mathcal{A}$ has zero probability. The construction of this distribution is based on the Moser–Tardos probabilistic algorithm [4] for finding an assignment that avoids all the bad events in the finite version of LLL. However, we need first to extend the class of probabilistic machines.

5 Rewriting machines and layerwise computability

Now we change the computational model and make the output tape rewritable: the machine can change several times the contents of a cell, and only the last value matters. (We may assume that ith cell may contain integers in $0 \ldots n_i - 1$ range, and that initially all cells contain zeros.) The last value is defined if a cell changes its value finitely many times during the computation. Of course, for some values of random bits it may happen that some cell gets infinitely many changes. We say that the output sequence of the machine is not defined in this case.

If the output sequence is defined with probability 1, we get an almost everywhere defined mapping from Cantor space (of random coin sequences) into the space of all assignments (input sequence is mapped to the limit output sequence). This mapping defines the output distribution on the assignments (the image of the uniform distribution on fair coin bits). This distribution may be not computable (e.g., for every lower semicomputable real α it is easy to generate $0000\ldots$ with probability α and $1111\ldots$ with probability $1-\alpha$). However, we get a computable output distribution if we impose some restrictions on the machine.

The restriction: *for every output cell i and for every rational $\delta > 0$ one can effectively compute integer $N = N(i, \delta)$ such that the probability of the event "the contents of cell i changes after step N", taken over all possible random bit sequences, is bounded by δ.*

Proposition 4. *In this case the limit content of the output is well defined with probability* 1*, and the output distribution on the sequences of zeros and ones is computable.*

Proof. Indeed, to approximate the probability of the event "output starts with z" for a sequence z of length k with error at most δ, we find $N(i, \delta/k)$ for k first cells (i.e., for $i = 0, 1, \ldots, k-1$). Then we take n greater than all these values of N and simulate first n steps of the machine for all possible combinations of random bits. □

An almost everywhere defined mappings of Cantor space defined by machines with described properties, are called *layerwise computable*. Initially they appeared in the context of algorithmic randomness [2]. One can show that such a mapping is defined on all Martin-Löf random inputs. Moreover, it can be computed by a machine with write-only output tape if the machine additionally gets the value of randomness deficiency for the input sequence. This property can be used (and was originally used) as an equivalent definition of layerwise computable mapping. (The word "layerwise" reflects this idea: the mapping is computable on all "randomness levels".)

This aspect, however, is not important for us now; all we need is to construct a layerwise computable mapping whose output distribution avoids all the forbidden events with probability 1, and then apply Proposition 1.

6 Rewriting machine for computable LLL

We recall first Moser–Tardos probabilistic algorithm that finds a satisfying assignment for the finite LLL. (We do not repeat the argument here and assume that the reader is familiar with [4]: some estimates from this paper are needed and we assume that the reader knows their proofs from [4].)

Having only finitely many events, we have only finitely many essential variables. The algorithm starts by assigning random values to all variables. Then, while there are some non-satisfied conditions (=some bad events happen), the algorithm takes one of these events and resamples all the variables that appear in this event (assigning fresh random values to them).

There is some freedom in this algorithm: the bad event for next resampling can be chosen in an arbitrary deterministic (or even probabilistic) way. Moser and Tardos have proven that with probability 1 this algorithm terminates (making all the conditions satisfied), and terminates rather fast. In particular, they have shown that the expected number of resampling operations for some bad event A is bounded by $x(A)/(1-x(A))$.

We modify this algorithm for the case of infinitely many variables and events and get a probabilistic machine that with probability 1 layerwise-computes a satisfying assignment. Then Propositions 1 and 4 guarantee the existence of a computable satisfying assignment, and this finished the proof of Theorem 1.

Let us introduce some priority on the conditions (bad events). For each condition A we look at the variables it involves, and take the variable with maximal index, denoted by $\max \mathrm{vbl}(A)$. Then we reorder all the conditions in such a way that

$$\max \mathrm{vbl}(A_0) \le \max \mathrm{vbl}(A_1) \le \max \mathrm{vbl}(A_2) \le \dots$$

(Recall that each variable is used only in finitely many conditions, so we can make the rearrangement in a computable way. This rearrangement is not unique.)

Then the algorithm works exactly as before, choosing the first violated condition (in this ordering).

An important observation: for some n consider all the conditions that depend on variables $x_0, x_1, \dots, x_n$ only. These conditions form a prefix in our ordering. Therefore, while not all of them are satisfied, we will not consider any other conditions, so our infinite algorithm will behave (up to some point) like a Moser–Tardos finite algorithm. This implies the bound $x(A)/(1-x(A))$ for an average number of resampling operations for condition A, so the expected total number of resamples for this finite algorithm is finite. We come to the following conclusion:

Lemma 1. *With probability* 1 *our algorithm will at some point satisfy all the conditions depending on* $x_0, \dots, x_n$.

Therefore, with probability 1 the actions of the infinite probabilistic algorithm can be split into stages: at ith stage we resample conditions that depend on $x_0, \dots, x_i$ only, until all of them are satisfied. Let $x_0^i, \dots, x_i^i$ be the values of the variables $x_0, \dots, x_n$ at the end of the ith stage, i.e., at the first moment when all the conditions depending only on $x_0, \dots, x_i$ are satisfied.

These x_i^j (for $i \le j$) are random variables defined with probability 1 (due to Lemma 1). The values $x_0^i, \dots, x_0^i$ form a satisfying assignment for all the conditions that depend only on them. However, these values are not guaranteed to be "final": when we start to work with other variables, this may lead to changes in the previous variables. So, e.g., x_i^{j+1} can differ from x_i^j. However, the stabilization happens with probability 1:

Lemma 2. *For every i with probability 1 the sequence*

$$x_i^i, x_i^{i+1}, x_i^{i+2}, \ldots$$

stabilizes.

Moreover, for every variable with probability 1 there exists some moment in our algorithm such that after this moment it will never be changed. (This is formally a stronger statement since a variable can change during some stage but return to its previous value at the end of the stage.)

Proof of Lemma 2. It is enough to show that for every i and sufficiently large j the probability of them event "x_i is changed after stage j" is small. To show this, we need to refer to the details of Moser–Tardos argument. Consider all the events that involve the variable x_i. Then consider all the neighbors of these events, all neighbors of their neighbors, etc. (m times for some large m). Let j be the maximal index of a variable that appears in all these events (up to distance m).

We claim that *for every event A that involves x_i, the probability of being resampled after stage j does not exceed* $(1-\varepsilon)^m$. (Recall that $(1-\varepsilon)$ is a safety margin factor that we have, compared to classical LLL.) Indeed, consider such a resample and its tree (constructed as in [4]). This tree should contain some event that involves variable with index greater than j (since a new resample became necessary after all variables up to x_j have satisfactory values). The choice of j guarantees then that the size of the tree is at least m, and the sum of probabilities of all such trees is bounded by $(1-\varepsilon)^m x(A)/(1-x(A))$, for the same reasons as in Moser–Tardos proof. By a suitable choice of m we can make this probability as small as we wish. Lemma 2 is proven. □

Note that at this stage we have already shown the existence of an assignment that satisfies all the conditions, since such an assignment is produced by our algorithm with probability 1. But to finish the argument, we need to establish a computable bound for convergence speed.

Lemma 3. *The convergence in Lemma 2 is effective: for every i and for every δ one can compute some $N(i,\delta)$ such that the probability of the event "x_i will change after $N(i,\delta)$ steps of the algorithm" is less than δ.*

Proof of Lemma 3. The estimate in the proof of Lemma 2 gives some bound in terms of the number of stages. At the same time we know the bounds for the expected length of each stage, and can use Chebyshev inequality. Lemma 3 is proven. □

This finishes the proof of Theorem 1.

7 Corollaries

We conclude the paper by showing some special cases where complutable LLL can be applied.

A standard illustration for LLL is the following result: *a CNF where all clauses contain m different variables and each clause has at most 2^{m-2} neighbors, is always satisfiable.* Here neighbors are clauses that have common variables.

Indeed, we let $x(A) = 2^{-(m-2)}$ and note that

$$2^{-m} \le 2^{-(m-2)}[(1 - 2^{-(m-2)})^{2^{m-2}}],$$

since the expression in square brackets is approximately $1/e > 1/2^2$.

This was about finite CNFs; now we may consider *effectively presented* infinite CNFs. This means that we consider CNFs with countably many variables and clauses (numbered by natural numbers); we assume that for given i we can compute the list of clauses where ith variable appears, and for given j we can compute jth clause.

Corollary 1. *For every effectively presented infinite CNF where each clause contains m different variables and every clause has at most 2^{m-2} neighbors, one can find a computable assignment that satisfies it.*

Proof. Indeed, the same choice of $x(A)$ works, if we choose ε small enough (say, $\varepsilon = 0.1$). □

Similar argument can be applied in the case where there are clauses of different sizes. The condition now is as follows: for every variable there are at most $2^{\alpha n}$ clauses of size that involve this variable, where $\alpha \in (0, 1)$ is some constant. Note that here we do not assume that every variable appears in finitely many clauses, so the notion of effectively presented infinite CNF should be extended. Now we assume that for each i and for each n one can compute the list of clauses of size n that include x_i.

Corollary 2. *For every $\alpha \in (0, 1)$ there exists some N such that every effective infinite CNF where each variable appears in at most $2^{\alpha n}$ clauses of size n (for every n) and all clauses have size at least N, has a computable satisfying assignment.*

Proof. Let us consider first a special case when each variable appears only in finitely many clauses. Then we are in the situation covered by Theorem 1, and we need only to choose the values of $x(A)$. These values will depend on the size of the clause A: let us choose

$$x(A) = 2^{-\beta k}$$

for clauses of size k, where β is some constant. In fact, any constant between α and 1 will work, so we can use, e.g., $\beta = (1 + \alpha)/2$. So we need to check (for clauses of some size k) that

$$2^{-k} \le 2^{-\beta k} \prod_{B \in \Gamma(A)} (1 - 2^{-\beta \# B})$$

Note that for every of k variables in A there are at most $2^{\alpha m}$ clauses of size m that involve it. So together there are at most $k2^{\alpha m}$ neighbors of size m. So it is enough to show that

$$2^{-k} \le 2^{-\beta k} \prod_{m \ge N} (1 - 2^{-\beta m})^{k2^{\alpha m}}$$

Using that $(1 - h)^s \ge 1 - hs$ and taking kth roots, we see that it is enough to show that

$$2^{-1} \le 2^{-\beta}(1 - \sum_{m \ge N} 2^{\alpha m} 2^{-\beta m})$$

Since the series $\sum 2^{(\alpha-\beta)m}$ is converging, this is guaranteed for large N.

So we have proven Corollary 2 for the special case when each variable appear only in finitely many clauses (and we can compute the list of those clauses).

The general case is easily reducible to this special one. Indeed, fix some $\rho > 0$ and delete from each clause ρ-fraction of its variables with minimal indices. The CNF becomes only harder to satisfy. But if ρ is small enough, the conditions of the theorem (the number of clauses with m variables containing a given variable is bounded by $2^{\alpha n}$) are still true for some $\alpha' \in (\alpha, 1)$, because the deletion makes the size of clauses only slightly smaller and decreases the set of clauses containing a given variable. And in this modified CNF each variable appears only in clauses of limited size (it is deleted from all large enough clauses). □

Let us note an important special case. Assume that F is a set of binary strings that contains (for every n) at most $2^{\alpha n}$ strings of size n. Then one can use LLL to prove the existence of an infinite (or bi-infinite) sequence ω and a number N such that ω does not have substrings in F of length greater than N. There are several proofs of this statement; the most direct one is given in [3]; one may also use LLL or Kolmogorov complexity, see [5, 6].

Joseph Miller noted that his proof (given in [3]) can be used to show that for a decidable F (with this property) one can find a computable ω that avoids long substrings in F. Konstantin Makarychev extended this argument to bi-infinite strings (personal communication). Now this result becomes a special case of Corollary 2: places in the sequence correspond to variables, each forbidden string gives a family of clauses (one per each starting position), and there is at most $n2^{\alpha n}$ clauses of size n that involve given position (and this number is bounded by $2^{\alpha' n}$ for slightly bigger α' and large enough n).

Moreover, we can do the same for 2-dimensional case: having a decidable set F of rectangular patterns that contains at most $2^{\alpha n}$ different patterns of size (=area) n, one can find a number N and computable 2D configuration (a mapping $\mathbb{Z}^2 \to \{0, 1\}$) that does not contain patterns from F of size N or more. (It is not clear how to get this result directly, without using Moser–Tardos technique.)

Authors are grateful to Lance Fortnow who suggested to apply Moser–Tardos technique to get the infinite computable version of LLL.

References

[1] Erdös P., Lovász L., Problems and results of 3-chromatic hypergraphs and some related questions. In: A. Hajnal, R. Rado, V.T. Sós, Eds. Infinite and Finite Sets. North-Holland, *II*, 609–627.

[2] Hoyrup M., Rojas C., Applications of Effective Probability Theory to Martin-Löf Randomness. *Lectures Notes in Computer Science*, 2009, v. 5555, p. 549–561.

[3] Miller J., *Two notes on subshifts*. Available from `http://www.math.wisc.edu/ jmiller/downloads.html`

[4] Moser R.A., Tardos G., A constructive proof of the general Lovász Local Lemma, *Journal of the ACM*, 2010, 57(2), 11.1–11.15. See also: `http://arxiv.org/abs/0903.0544`.

[5] Rumyantsev A., Forbidden Substrings, Kolmogorov Complexity and Almost Periodic Sequences, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer*

Science, Marseille, France, February 23–25, 2006. Lecture Notes in Computer Science, 3884, Springer, 2006, p. 396–407.

[6] Rumyantsev A., Kolmogorov Complexity, Lovász Local Lemma and Critical Exponents. *Computer Science in Russia*, 2007, Lecture Notes in Computer Science, 4649, Springer, 2007, p. 349-355.

[7] Rumyantsev A., *Everywhere complex sequences and probabilistic method*, `arxiv.org/abs/1009.3649`

[8] Rumyantsev A., *Infinite computable version of Lovasz Local Lemma*, `arxiv.org/abs/1012.0557`

[9] Zvonkin A. K., Levin L. A., The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms, Uspekhi Matematicheskikh Nauk, 1970, v. 25, no. 6 (156), p. 85–127.