HAL
open science

Diagonally non-computable functions and fireworks

Laurent Bienvenu, Ludovic Patey

To cite this version:

Laurent Bienvenu, Ludovic Patey. Diagonally non-computable functions and fireworks. Information and Computation, 2017, 253 (Part 1), pp.64-77. 10.1016/j.ic.2016.12.008 . lirmm-01693182

HAL Id: lirmm-01693182

https://hal-lirmm.ccsd.cnrs.fr/lirmm-01693182v1

Submitted on 1 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Diagonally non-computable functions and fireworks

Laurent Bienvenu Ludovic Patey

September 10, 2016

Abstract

A set $\mathcal{C}$ of reals is said to be negligible if there is no probabilistic algorithm which generates a member of $\mathcal{C}$ with positive probability. Various classes have been proven to be negligible, for example the Turing upper-cone of a non-computable real, the class of coherent completions of Peano Arithmetic or the class of reals of minimal degrees. One class of particular interest in the study of negligibility is the class of diagonally non-computable (DNC) functions, proven by Kučera to be non-negligible in a strong sense: every Martin-Löf random real computes a DNC function. Ambos-Spies et al. showed that the converse does not hold: there are DNC functions which compute no Martin-Löf random real. In this paper, we show that the set of such DNC functions is in fact non-negligible. More precisely, we prove that for every sufficiently fast-growing computable h, every 2-random real computes an h-bounded DNC function which computes no Martin-Löf random real. Further, we show that the same holds for the set of reals which compute a DNC function but no bounded DNC function. The proofs of these results use a combination of a technique due to Kautz (which, following a metaphor of Shen, we like to call a 'fireworks argument') and bushy tree forcing, which is the canonical forcing notion used in the study of DNC functions.

1 Background

1.1 Negligibility, Levin-V'yugin algebra and DNC functions

Let $\mathcal{C} \subseteq 2^\omega$ be a class of infinite binary sequences (a.k.a. *reals*) and, consider the set

$$\{X \in 2^\omega : X \text{ Turing-computes some element of } \mathcal{C}\}$$

By Kolmogorov's 0-1 law theorem, its (Lebesgue) measure is either 0 or 1. If it is equal to 0, the class $\mathcal{C}$ is said to be *negligible*, a terminology due to Levin and V'yugin [17]. Equivalently, this means that there is no (infinite) probabilistic algorithm which generates a member of $\mathcal{C}$ with positive probability.

Various classes have been proven to be negligible, for example the Turing upper-cone of a non-computable A [5] (or a countable class of such A's), the class of coherent completions of Peano Arithmetic [10], the class of minimal degrees [18], the class of shift-complex sequences [20, 12], etc. Likewise, many classes have been showed to be non-negligible (or 'typical'). Obviously classes of positive measure, such as the class of Martin-Löf random reals, are all non-negligible. Much more interesting examples were given by Kautz [11][1]. He

[1] Very similar ideas were used by V'Yugin in [22, 23]

showed that the following are non-negligible: the class of hyperimmune sets, the class of 1-generic reals, and the class of sets of CEA degrees. All three results are variations of the same technique. However, the way Kautz presents his technique is quite abstract and in some sense hides its true spirit, namely that what is being used is a probabilistic algorithm. In the paper [19], Rumyantsev and Shen give a more explicit and intuitive explanation of the underlying algorithm, using the metaphor of a fireworks shop in which a customer is trying to either buy a box of good fireworks, or expose the vendor by opening a flawed one, and uses a probabilistic algorithm to maximize his chances of success. Following the tradition of colourful terminology in computability theory (Lerman's pinball machine, Nies' decanter argument...), we propose to refer to this method as a *fireworks argument*, the precise template of which will be recalled in the next section.

The dichotomy between negligibility and non-negligibility has received quite a lot of attention in recent years. We refer the reader to [2, 3, 21] for a panorama of the existing results in this direction.

One class of particular interest in the study of negligibility is the class of diagonally non-computable functions, or 'DNC functions' for short. It consists of the total functions $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) \neq \varphi_n(n)$ for all n, where (φ_n) is a standard enumeration of partial computable functions from $\mathbb{N}$ to $\mathbb{N}$. By definition, there is no computable DNC function. On the other hand, as showed by Kučera [15], the class of DNC functions is non-negligible. If we take the point of view of probabilistic algorithms, this is clear: since for all n there is only one value for $f(n)$ to avoid (namely, $\varphi_n(n)$ if it is defined), so by picking $f(n)$ at random between 0 and some large integer (e.g. 2^n), we ensure a positive probability of success. The situation becomes more interesting when one restricts the class DNC to the subclass

$$\mathrm{DNC}_h = \{f : \mathbb{N} \to \mathbb{N} \mid f \text{ is a DNC function and } (\forall n) f(n) < h(n)\}$$

where h is a given function, typically a computable one. The faster h grows, the easier it is to obtain an element of DNC_h. And indeed, depending on the growth rate of h the class DNC_h can be negligible or non-negligible (an unpublished result due to J. Miller, see [3] for a proof).

This notion relativizes to an arbitrary oracle X: a DNC^X function is a function f such that $f(n) \neq \varphi_n^X(n)$ for all n. Likewise, we set

$$\mathrm{DNC}_h^X = \{f : \mathbb{N} \to \mathbb{N} \mid f \text{ is a } \mathrm{DNC}^X \text{ function and } (\forall n) f(n) < h(n)\}$$

Of course, the stronger the oracle X, the harder it is to compute a DNC^X function.

In this paper, we study the role of DNC functions in the setting of the Levin-V'yugin algebra, which is the algebra of Turing invariant Borel sets 'modulo negligibility'. That is, for two Turing-invariant sets $\mathcal{A}$ and $\mathcal{B}$ we write $\mathcal{A} \sqsubseteq \mathcal{B}$ if $\mathcal{A} \smallsetminus \mathcal{B}$ is negligible. We say that $\mathcal{A}$ and $\mathcal{B}$ are equivalent if $\mathcal{A} \sqsubseteq \mathcal{B}$ and $\mathcal{B} \sqsubseteq \mathcal{A}$. We call an equivalence class for this equivalence relation a *Levin-V'yugin degree*.

Despite having been introduced some time ago and being a very natural notion, little work has been done on the Levin-V'yugin degrees, except for the seminal papers [17, 22, 23] and

some ongoing work by Hölzl and Porter. It is during discussions with the authors of the latter that a question arose. Kučera's result discussed above shows that reals of Martin-Löf random degree are also of DNC degree, so $\mathbf{mlr} \sqsubseteq \mathbf{dnc}$, where $\mathbf{mlr}$ is the set of sequences Turing-equivalent to a Martin-Löf random real, and $\mathbf{dnc}$ those equivalent to a DNC function. On the other hand, it is well-known that there are DNC functions which do not Turing-compute any Martin-Löf random real (see subsection 1.3 below). But does this result translate in the setting of V'yugin's algebra? More precisely, is it then the case that $\mathbf{mlr} \sqsubset \mathbf{dnc}$ (i.e., that $\mathbf{dnc} \smallsetminus \mathbf{mlr}$ is non-negligible)? In this paper, we answer this question in the affirmative. Namely, we prove:

Theorem 1 (Main theorem) For every sufficiently fast-growing computable h, every 2-random (i.e., Martin-Löf random relative to $\emptyset'$) real Z computes some $f \in \mathrm{DNC}_h$ which does not compute any Martin-Löf random real.

Not only is this result interesting in its own right, but its proof is particularly instructive. It combines fireworks arguments with bushy tree forcing, a forcing notion used in many recent papers to study the properties of DNC functions [1, 4, 7, 9, 13]. To our knowledge, our proof is the most elaborate use of a fireworks argument to date. It illustrates quite convincingly the power of the technique, and is likely to yield further applications in the future.

Also interesting is the fact that if we want to study functions which are DNC relative to some oracle X, we can state a stronger theorem than what we would get from a straightforward relativization of Theorem 1.

Theorem 2 (Main theorem, relativized) For any real X and a sufficiently fast-growing computable h, every real Z which is both X-random and 2-random computes a function $f \in \mathrm{DNC}_h^X$ which itself computes no Martin-Löf random real.

A straightforward relativization of Theorem 1 would require Z to be X'-random, and would give a function f which does not compute any X-Martin-Löf random real, but it could still compute a Martin-Löf random real. Note that taking $X = \emptyset'$ gives us a stronger result than Theorem 1: Every 2-random real Z computes some $f \in \mathrm{DNC}_h^{\emptyset'}$ which is does not compute any Martin-Löf random real.

We finally remark *en passant* that '2-random' in the hypothesis of Theorem 2 cannot be substituted for 'Martin-Löf random', as shown by the following easy proposition.

Proposition 3 There is a Martin-Löf random real which computes no member of $\mathbf{dnc} \smallsetminus \mathbf{mlr}$.

Proof. Take any hyperimmune-free Martin-Löf random Z. Because of hyperimmune-freeness, everything Turing-computed by Z is in fact tt-computed by Z. Moreover, every non-computable real which is tt-computed by a Martin-Löf random real is also of Martin-Löf random degree [6]. In particular, every DNC function it computes is of Martin-Löf random degree. □

Remark. Readers who are experts in algorithmic randomness may not fully be satisfied with this last proposition, and rightfully so. Indeed, there is a wealth of algorithmic randomness

notions between Martin-Löf randomness and 2-randomness, and it would be interesting to know precisely which levels of randomness it is sufficient for a real to have in order to compute a member of $\mathbf{dnc} \smallsetminus \mathbf{mlr}$. A more precise answer is the following: weak-2-randomness is not sufficiently strong, but Demuth randomness is. For the first part of this assertion, observe that in the proof of Proposition 3, one can take Z to be weak-2-random (indeed there are hyperimmune-free weak-2-randoms). The second part is more involved and requires a fine analysis of fireworks arguments which will be done in a forthcoming paper by Christopher Porter and the first author. However, the crude '2-randomness' bound we use in this paper is sufficient for our main goal, which is to prove the non-negligibility of $\mathbf{dnc} \smallsetminus \mathbf{mlr}$.

1.2 Notation and terminology

Unless otherwise specified, a *string* is a finite sequence of integers. We denote the set of strings by $\omega^{<\omega}$, by λ the empty string and by $|\sigma|$ the length of a string σ. Unless specified otherwise, a *sequence* is an infinite list of integers. The set of sequences is denoted by ω^ω. We will sometimes need to consider *binary* sequences (which we also call *reals*), the set of which we denote by 2^ω. The n-th element of a string or sequence Z is denoted by $Z(n-1)$ and $Z{\restriction}n$ denotes the finite sequence consisting of the first n values of Z. A string τ is a *prefix* of a string σ (we also say that σ *extends* τ), noted $\tau \preceq \sigma$, if $|\sigma| \geq |\tau|$ and $\sigma{\restriction}|\tau| = \tau$.

A sequence $Z \in \omega^\omega$ is said to be a DNC function (resp. X-DNC function) if for all n, $Z(n) \neq \varphi_n(n)$ (resp. $Z(n) \neq \varphi_n^X(n)$), where (φ_n) is a standard enumeration of partial computable functions from $\mathbb{N}$ to $\mathbb{N}$ with oracle.

A *tree* $T \subseteq \omega^{<\omega}$ is a set of strings closed downwards under the prefix relation, i.e., if $\sigma \in T$ and $\tau \preceq \sigma$ then $\tau \in T$. Members of a tree are often referred to as nodes. A node σ is a *child* (or *immediate extension*) of a node τ in a tree T if $\tau \preceq \sigma$, τ and σ are nodes of T, and $|\sigma| = |\tau| + 1$. A *leaf* of a tree T is a node with no immediate extension in T. A *path in a tree* T is a sequence f such that every initial segment of f is in T. The class of paths of a tree T forms a closed set denoted by $[\![T]\!]$.

1.3 Bushy trees

In this section we present the notion of bushy tree and its main properties. Roughly speaking, bushiness is a purely combinatorial property, which states that a tree is 'sufficiently fast branching', in a way that guarantees the existence of a DNC path through the tree. The idea of bushy tree was invented by Kumabe, who used it to construct a DNC function of minimal Turing degree (see [16] for an exposition of this result), which in particular shows that there is a DNC function which computes no Martin-Löf random real (as no Martin-Löf random real can have minimal degree). Since then, bushy trees have been successfully applied to the study of DNC functions. We refer the reader to the excellent survey of Khan and Miller [13].

Definition 1 (Bushy tree) Fix a function h and a string $\sigma \in \omega^{<\omega}$. A tree T is h-bushy (resp. *exactly h-bushy*) above σ if every $\tau \in T$ is comparable with σ and whenever $\tau \succeq \sigma$ is not a leaf of T, it has at least (resp. exactly) $h(|\tau|)$ immediate children. We call σ the *stem* of T.

Definition 2 (Big set, small set) Fix a function h and some string $\sigma \in \omega^{<\omega}$. A set $B \subseteq \omega^{<\omega}$ is *h-big above σ* if there exists a finite tree T which is h-bushy above σ and such that all leaves of T are in B. If no such tree exists, B is said to be *h-small above σ*.

Bushy tree forcing consists generally of building decreasing sequences of infinite bushy trees $T_0 \supseteq T_1 \supseteq T_2 \dots$ where T_i is h-bushy over σ_i for some string σ_i. Each stem σ_i is an initial segment of the constructed sequence. During the construction we maintain a set B of "bad" extensions, i.e., of strings to avoid. This set must remain g-small above σ_i at any stage for some function g. Bushy tree forcing is especially convenient for building DNC functions. Let B_{DNC} be the set of strings which are not initial segments of any DNC function:

$$B_{\mathrm{DNC}} = \left\{\sigma \in \omega^{<\omega} : (\exists e < |\sigma|)\, \varphi_e(e) \downarrow = \sigma(e)\right\}$$

One can easily see that B_{DNC} is 2-small above λ.

The following three lemmas are at the core of every bushy tree argument. We state them without proof and refer the reader to [13] for details.

Lemma 1 (Concatenation) Fix a function h. Suppose that $A \subseteq \omega^{<\omega}$ is h-big above a string σ. Let $(S_\tau)_{\tau \in A}$ be a family of subsets of $\omega^{<\omega}$. If $S_\tau \subseteq \omega^{<\omega}$ is h-big above τ for every $\tau \in A$, then $\bigcup_{\tau \in A} S_\tau$ is h-big above σ.

The concatenation property is often used in the following contrapositive form: If we are given a finite tree T h-bushy above some string σ and a "bad" set B of extensions to avoid which is h-small above σ, then there exists a leaf τ of T such that B is still h-small above τ. In particular, if a set A is h-big above σ, there exists an extension τ of σ which is in A and such that B is still h-small above τ.

Lemma 2 (Smallness additivity) Suppose that $B_1, B_2, \dots, B_n$ are subsets of $\omega^{<\omega}$, g_1, g_2, ..., g_n are functions, and $\sigma \in \omega^{<\omega}$. If B_i is g_i-small above σ for all i, then $\bigcup_i B_i$ is $(\sum_i g_i)$-small above σ.

Lemma 3 (Small set closure) We say that $B \subseteq \omega^{<\omega}$ is *g-closed* if whenever B is g-big above a string ρ then $\rho \in B$. Accordingly, the *g-closure* of any set $B \subseteq \omega^{<\omega}$ is the set $C = \{\tau \in \omega^{<\omega} : B \text{ is } g\text{-big above } \tau\}$. If B is g-small above a string σ, then its closure is also g-small above σ.

As explained in [13], Lemma 3 is very useful during our constructions. We are given a "bad" set B of nodes, which is g-small above σ where σ is a partial approximation of the object we are constructing. We want to extend σ still avoiding B and in particular preserving g-smallness of B. Lemma 3 enables us to consider w.l.o.g. that if ρ is an extension of σ which does not preserves g-smallness of B, then ρ is already on B.

The next lemma is very simple and yet central in this paper. It expresses the fact that if a set B is sufficiently small in an h-bushy tree T, then there is only a small probability that

a random path of the tree meets B (has a member of B as prefix). By "random path", we mean the probability distribution over paths induced by a downward random walk where one starts at the root and at each step goes from a node to one of its children, all children being given the same probability of being picked.

Lemma 4 Fix two functions positive functions g and h with $g \leq h$. If T is an infinite tree h-bushy above λ and $B \subseteq T$ is a g-small above λ, then the probability that a random path of T avoids B is greater than

$$\prod_{i \in \omega} \left(1 - \frac{g(i)}{h(i)}\right)$$

Note that this quantity is positive if and only if $\sum g(i)/h(i) < \infty$, due to the identity $\prod_{i\in\omega}(1-e_i) = \exp\left(-\sum_{i\in\omega}\ln(1-e_i)\right)$ and the asymptotic estimate $-\ln(1-x) = x + o(x)$.

Proof. Without loss of generality, we can assume that B is g-closed (otherwise, take its closure). We prove by induction over n that the probability of having avoided B by the time we reach depth n is at least $\prod_{i=0}^{n-1}\left(1-g(i)/h(i)\right)$ (a quantity equal to 1 for $n = 0$, by convention). The lemma immediately follows from this fact.

The base case $n = 0$ is trivial as λ is the only such node and B is g-small above λ. Assume it is true for some depth n. Suppose we have reached a node $\rho \in T$ of length n such that B is g-small above ρ. By h-bushiness of T, ρ has at least $h(n)$ immediate extensions. If B were g-big above $g(n)$-many immediate extensions of ρ then B would be also g-big above ρ, by g-closedness. Hence B is g-small above at least $h(n) - g(n)$ immediate extensions of ρ. It follows easily that, conditional to having reached ρ, the probability to avoid B at the next level is at least $1 - g(n)/h(n)$. This finishes the induction. □

Remark. Lemma 4 makes no computability assumption on T and B. However, when T is computable, taking a path of T at random can be performed using a probabilistic algorithm, which will then produce a path avoiding B with probability at least $\prod_{i=0}^{n-1}\left(1-g(i)/h(i)\right)$ (and this still makes no computability assumption about B).

Now we see how randomness helps us compute DNC functions: take a computable function h such that $\prod_i(1 - 2/h(i)) > 0$ (which is equivalent to $\sum_i 1/h(i) < \infty$; for example, $h(n) = (n+1)^2$ will do) and take an h-bushy tree T. Now, take a path of T at random. Since, as we saw above, B_{DNC} is 2-small in T, the previous lemma tells us that the probability to get a path of the tree which avoids B, and thus is a DNC function, is at least $\prod_i(1-2/h(i))$, thus is positive.

1.4 Fireworks arguments

A fireworks argument can be seen as "probabilistic forcing" for Σ_1 properties. It is best illustrated by the following theorem, due to Kautz: there exists a probabilistic algorithm which with positive probability produces a 1-generic real (more precisely, every 2-random real computes a 1-generic real). Let us present the argument in a more abstract way so as to better fit the setting of the next section, where we will (implicitly) force with bushy trees. Let

$(\mathbb{P}, \leq)$ be a computable partial order, and let W_1, W_2, ... be a list of uniformly c.e. subsets of $\mathbb{P}$. We want to get a probabilistic algorithm to generate an infinite list $p_0 \geq p_1 \geq p_2 \geq ...$ such that for every set W_i, the requirement $\mathcal{R}_i$ holds, where $\mathcal{R}_i$ says that there exists a j such that either $p_j \in W_i$ holds or for every $q \leq p_j$, $q \notin W_i$. (For example, to get Kautz's result, on takes for $\mathbb{P}$ the set of finite strings, where $\sigma \leq \tau$ if σ extends τ and the W_i are all c.e. sets of strings. In this paper, $\mathbb{P}$ will typically consist of a set of finitely represented bushy trees, and $T' \leq T$ will mean that T' is a subset of T).

Suppose we have already built the sequence of p_j up to some p_k, and we want to satisfy the requirement $\mathcal{R}_i$. If we did not care about the effectivity of the construction, we could easily satisfy the requirement by distinguishing two cases:

> Case 1 (the Π_1 case): there is no $p' \leq p_k$ such that $p' \in W_i$. In this case nothing needs to be done, $\mathcal{R}_i$ is already satisfied!
>
> Case 2 (the Σ_1 case): there is some $p' \leq p_k$ such that $p' \in W_i$. Here it suffices to search for such a p' (which can be done effectively since W_i is c.e.), and set $p_{k+1} = p'$ after which $\mathcal{R}_i$ is satisfied.

Although both cases only require effective actions (do nothing or effectively search for a p', respectively), the problem is that one cannot computably distinguish between them, as being in Case 2 is only a c.e. event (hence the name 'Σ_1 case'). And indeed in general there is no deterministic algorithm to build a sequence of p_j satisfying all requirements (otherwise one could, for example, computably build a 1-generic).

There *is* however, a *probabilistic algorithm* which builds such a sequence of p_j's with positive probability. It works as follows. We start with any $p_0 \in \mathbb{P}$. Next, for each i, we pick an integer n_i at random between 1 and $N(i)$, where N is a fixed computable function (the faster N grows, the higher the probability of success of the algorithm will be). Moreover, for each i, we let c_i be a counter, initialized at 0, which will count how many wrong passive guesses we have made for requirement $\mathcal{R}_i$.

By "passive guess", we mean that in the construction of the p_j's, we assume at some step k that we are in the Π_1 case, i.e., that no $q \leq p_k$ is in W_i. It is a passive guess because as we saw, if it is indeed true, requirement $\mathcal{R}_i$ is already satisfied and no particular action is needed. Of course, this guess may be incorrect but since W_i is c.e., if it is incorrect we will discover this at some later stage k' of the algorithm. When this happens, we make a new assumption that there are no $q \leq p_{k'}$ in W_i and so on. If at any point we make a correct passive guess, requirement $\mathcal{R}_i$ is satisfied. There is however a danger that *all* the passive guesses we make for requirement $\mathcal{R}_i$ turn out to be wrong. What we do is use the number n_i as a cap on how many times we allow ourselves to make a wrong passive guess for $\mathcal{R}_i$. If for some i the cap n_i is reached at stage k, we then make the opposite guess ("active guess"), i.e., that there *is* a $q \leq p_k$ such that $q \in W_i$ holds, try to find such a q and take it as our p_{k+1}, thus satisfying requirement $\mathcal{R}_i$. This guess is "active" because we need to find such a q before doing anything else. But at least, as we said above, since W_i is a c.e. set, such a q can be effectively found if it exists. Then we take $p_{k+1} = q$.

This active/passive guessing strategy is still not guaranteed to work, as one bad case remains: if we make an *incorrect active guess* for some $\mathcal{R}_i$, we then get stuck while waiting for a q in W_i which we will never find. However, this is the only bad case: if it does not happen for any $\mathcal{R}_i$, then the algorithm succeeds in producing a sequence $p_0 \geq p_1 \geq p_2 \geq ...$ as wanted. Indeed, for every i, either it makes a good passive guess for $\mathcal{R}_i$ that never turns out to be wrong, meaning that for some p_s, no $q \leq p_s$ is in W_i, or it makes a good active guess that some $q \leq p_s$ is in W_i, eventually finds such a q, and take it as an extension.

Why can the probability of success of the algorithm be made arbitrarily close to 1? The reason is the following key observation: For all i, if all $\{n_j : j \neq i\}$ are fixed, then there is at most one value of the random variable n_i for which we get stuck in a loop while trying to satisfy requirement $\mathcal{R}_i$. Indeed, suppose we get stuck having chosen a value n_i. This means that we made $n_i - 1$ incorrect passive guesses and then one incorrect active guess. Any other choice $n_i' < n_i$ would have been fine, because our n_i'-th guess would then have been a correct active guess and a q in W_i. And any other choice $n_i' > n_i$ would also have been fine, as in this case our n_i-th guess would have been a correct passive guess. Thus, the probability to get stuck because of requirement $\mathcal{R}_i$ is at most $1/N(i)$, giving a total probability of success of the algorithm of at least $1 - \sum_i 1/N(i)$, which can be made arbitrarily close to 1 for a well chosen N.

Now the last thing we need to check is how much randomness (in the sense of algorithmic randomness) is needed for this probabilistic algorithm to work. Let us explain what we mean. A probabilistic algorithm is nothing but a Turing functional Γ with access to a 'random' (in the classical sense) oracle $R \in 2^\omega$. This is the R used by the algorithm to make its random decisions. What we have argued above is that the failure set of the algorithm

$$U = \{R \mid \Gamma^R \text{ is either undefined or fails to satisfy some } \mathcal{R}_i \}$$

has probability < 1 (by 'undefined' we mean that the algorithm does not produce an infinite sequence of p_j's).

In fact, this probability can be made arbitrarily small, therefore fireworks arguments do not give us one algorithm but a uniform family of algorithms: For any given integer m, one can design, uniformly in m, a probabilistic algorithm which fails with probability at most 2^{-m} (it suffices to choose the function N such that $\sum 1/N(i) < 2^{-m}$). Call Γ_m the corresponding algorithm, and consider

$$U_m = \{R \mid \Gamma_m^R \text{ is either undefined or fails to satisfy some } \mathcal{R}_i \}$$

which is of measure at most 2^{-m}. The set $\bigcap_m U_m$, which is the set of R's on which *all* the algorithms Γ_m fail is a null set. This means that if R is 'sufficiently random', in the sense of effective randomness, it does not belong to all U_m and thus some algorithm Γ_m succeeds using R. Which level of algorithmic randomness is actually needed? One should observe that every U_m is in fact an effectively closed set relatively to $\emptyset'$. Indeed, as we have seen, the only case the algorithm Γ_m fails is when it waits in vain for a q extending some condition p and belonging to some W_i. If such a situation happens, it does so at some finite stage, i.e., having used only a finite initial segment of R, hence U_m is open. Moreover, testing

whether the algorithm Γ_m is stuck at a given stage can be done using $\emptyset'$: indeed, the predicate $[(\forall q \leq p)\ q \notin W_i]$ is a Π_1-predicate uniformly in p and i.

Thus, $(U_m)_{m\in\mathbb{N}}$ is a $\emptyset'$-Martin-Löf test, which shows that every 2-random real computes, via some functional Γ_m, an infinite sequence $p_0 \geq p_1 \geq p_2 \geq ...$ such that for every i there is a j such that either $p_j \in W_i$ holds or for every $q \leq p_j$, $q \notin W_i$, as wanted.

2 Main result

We shall now see how to combine fireworks arguments with bushy tree forcing to prove Theorem 2. We first provide an informal presentation of the proof. Full details will be given in the next section.

2.1 Proof overview

For this construction, we will need a hierarchy of very fast growing computable functions

$$g_0 \ll g_1 \ll g_2 \ll \dots$$

($g \ll g'$ is an informal notation: it means that g' grows 'much faster' than g) and another fast-growing function h (which is meant to grow faster that all the g_i but with certain restrictions). At this point, we do not specify precisely what functions g_i and h we take. We will see during the construction which properties they need to have to make the argument work.

Contrary to most bushy tree arguments, the whole construction will happen within a single tree T, which is exactly h-bushy:

$$T = \{\sigma \in \omega^{<\omega}\ :\ (\forall i)\ \sigma(i) < h(i)\}$$

Typically, a bushy tree forcing argument constructs a sequence $T_0 \supseteq T_1 \supseteq \dots$ of bushy trees, and the path obtained by forcing is in the intersection of all of those. We will not need such a sequence in our argument. However, some steps of the construction can be understood as "locally" taking a subtree of T. What we keep from other bushy tree arguments is the idea of maintaining during the construction a small set of bad strings to be avoided. But again, there is a difference in our construction: to build a DNC^X function by forcing, one usually starts with the initial small bad set

$$B_{\mathrm{DNC}} = \left\{\sigma \in \omega^{<\omega} : (\exists e < |\sigma|)\ \Phi_e^X(e) \downarrow = \sigma(e)\right\}$$

We will not do this as we need our bad set of strings to be c.e. at all times. Our fireworks argument will (with high probability) build an A which does not compute any Martin-Löf random real. It is only an *a posteriori* analysis of the construction which will allow us to conclude that A is also DNC^X with high probability. In the absence of any other requirement, we would just build A value by value, picking for each n the value of $A(n)$ at random between 0 and $h(n)-1$. This would give us a probability of avoiding B_{DNC} of at least $\prod_n(1-1/h(n))$, which is positive for h fast-growing. But of course there are other requirements our construction needs to meet, namely all the requirements of the form:

$$\mathcal{R}_{\Gamma,d}\text{: either } \Gamma^A \text{ is partial or there is an } n \text{ such that } K(\Gamma^A\restriction n) < n-d$$

where d is an integer, Γ is a Turing functional from $\omega^{<\omega}$ to 2^ω, and K is the prefix-free Kolmogorov complexity function. (Here we use the Levin-Schnorr theorem that a real Y is Martin-Löf random if and only if for some constant d, and all n, $K(Y\restriction n) \geq n-d$).

Let us see how we would ideally like to satisfy such a (single) requirement. Suppose we have already built some string $\sigma \in T$ of length k and consider the set

$$S = \{\tau \in T : |\Gamma^\tau| \geq h(k-1)\}$$

where $|\Gamma^\tau| \geq h(k-1)$ means that the Turing reduction Γ produces at least $h(k)$ bits of output on input τ, and distinguish two cases:

- Case 1: S is g_k-small above σ. In this case there is essentially nothing to worry about. We can just continue to build A by making random choices. The probability that we hit the set S at some point is small, namely it is at most $1-\prod_{i\geq k+1}(1-g_k(i)/h(i))$ (by Lemma 4), and recall that $h(i) \geq g_{k+1}(i) \gg g_k(i)$ for $i \geq k+1$. And if we do not hit S, then Γ^A will end up being partial, therefore satisfying $\mathcal{R}_{\Gamma,d}$.
- Case 2: S is g_k-big above σ. Each element $\tau \in S$ is such that $|\Gamma^\tau| \geq h(k-1)$, therefore we can decompose S as

  $$S = \bigcup_{|\rho|=h(k-1)} S_\rho \quad \text{where} \quad S_\rho = \{\tau \in S \mid \Gamma^\tau \succeq \rho\}$$

  There are $2^{h(k-1)}$ strings of length $h(k-1)$, therefore by Lemma 2, there must be a ρ^* such that S_{ρ^*} is $\big(g_k/2^{h(k-1)}\big)$-big above σ (note that in this expression, g_k is a function while $h(k-1)$ is just an integer). Since being big is a Σ_1-property, such a ρ^* can be found effectively knowing σ and Γ, and thus the first such ρ^* found with this effective search must satisfy

  $$K(\rho^*) \leq K(\sigma) + K(\Gamma) + c \leq 2\sum_{i\leq k-1} \log h(i) + K(\Gamma) + c' \tag{1}$$

  for some fixed constants c, c' (the last term is due to the fact that σ is a list of $k-1$ integers such that the i-th integer is less than $h(i)$, therefore has complexity less than $2\log h(i)$). Since $|\rho^*| = h(k-1)$ we have

  $$K(\rho^*) - |\rho^*| \leq 2\sum_{i\leq k-1} \log h(i) - h(k-1) + K(\Gamma) + c'$$

  If h grows fast enough, and k is sufficiently large then this last inequality implies $K(\rho^*) \leq |\rho^*| - d$. Thus, any A which passes through a node in S^*_ρ satisfies requirement $\mathcal{R}_{\Gamma,d}$. Moreover, since S_{ρ^*} is $\big(g_k/2^{h(k-1)}\big)$-big above σ, this means by definition that there is a finite $\big(g_k/2^{h(k-1)}\big)$-bushy tree $T' \subseteq T$ of stem σ all of whose leaves are in S_{ρ^*}. Then, what we can do is effectively find the tree T' and temporarily restrict our

random walk to T' (picking at each step the next value at random among nodes of T') until we reach a leaf of T'. This guarantees the satisfaction of $\mathcal{R}_{\Gamma,d}$ and the probability that we hit B_{DNC} during this temporary restriction is less than $1 - \prod_{i \geq k}(1 - 2^{h(k-1)}/g_k(i))$, which can be made small if g_k is well-chosen.

Of course, the problem is that, having built σ, we cannot effectively determine whether we are in Case 1 or in Case 2. This is where the fireworks argument comes into play. We are going to pick a number $n = n(\Gamma, d)$ between 1 and some large $N = N(\Gamma, d)$, and assume up to n times that we are in Case 1 (the Π_1 case), and if proven wrong n times, we will then wait until proven that we are in Case 2 (the Σ_1 case), and if so, implement the above strategy for Case 2. As with other fireworks arguments, the probability that we decide to wait at the wrong moment is at most $1/N$, which we can thus make arbitrarily small.

These considerations are enough to give us a strategy which ensures, with arbitrarily high probability, the satisfaction of a *single* requirement $\mathcal{R}_{\Gamma,d}$ while avoiding the set B_{DNC} with high probability. However, there is a subtle point to address when we try to satisfy several requirements in parallel. Indeed, what can happen is that the strategy for a first requirement has made the assumption that some set S is g_k-small in T - and thus small probability of being hit - while a strategy for a second requirement needs to make a temporary restriction to a tree T'. While the probability to hit S was small for a random walk within T, it could happen that the random walk restricted to T' has a much greater probability to hit S. This is what the assumption $g_i \ll g_j$ for $i < j$ takes care of: Whenever a strategy needs to make such a restriction to a $\left(g_k/2^{h(k-1)}\right)$-bushy subtree, we will have that $g_k/2^{h(k-1)}$ grows much faster than the g_j's previously considered in the proof, and thus, if a set S is g_j-small, it will still be unlikely that we hit S while choosing a path of a $\left(g_k/2^{h(k-1)}\right)$-bushy tree at random for any $k > j$.

2.2 The full algorithm and formal proof of correctness

We now state the precise theorem regarding the probabilistic algorithm discussed above. The analysis of the level of effective randomness required will be done separately.

Theorem 4 Let $X \in 2^\omega$ and h be a sufficiently fast-growing computable function. For every rational $\varepsilon > 0$, one can effectively design a probabilistic algorithm (= Turing machine with random oracle) which, with probability at least $1 - \varepsilon$, produces an $A \in \omega^\omega$ such that (1) A is DNC_h^X and (2) A computes no Martin-Löf random real.

Let $m = m(\varepsilon)$ (we will argue at the end of the proof that choosing m large enough guarantees that the algorithm succeeds with probability at least $1 - \varepsilon$). Let us first define the functions g_k and h we alluded to above. Set g_0 to be the function defined by $g_0(n) = 2^{n+m}$. Then, for all $k > 1$, inductively define

$$g_k(i) = \begin{cases} 1 \text{ if } i < k \\ g_{k-1}(i) \cdot 2^{g_{k-1}(k-1)+i+m} \text{ otherwise} \end{cases}$$

Finally define for all k

$$h(k) = g_k(k)$$

Let us now give the details of the algorithm. First, number all the requirements $\mathcal{R}_{\Gamma,d}$ and call $\mathcal{R}_i$ the i-th requirement. As usual, we organize them in a way that all requirements receive attention infinitely often, and only one requirement receives attention at any given stage. We will see during the verification that a small extra assumption should be added, namely that every requirement should be considered for the first time at some 'late enough' stage.

Stage 0: Initialization. The first thing we do is pick for all i a number n_i at random between 1 and 2^{i+m}. Then, we initialize σ to be the empty string. For all i, set a counter c_i originally equal to 0.

Loop (to be repeated indefinitely). Suppose that the values $\sigma(0), ..., \sigma(k-1)$ of σ are already defined (the last one being between 0 and $h(k-1)-1$). Assume some requirement $\mathcal{R}_i = \mathcal{R}_{\Gamma,d}$ receives attention.

(a) If this requirement receives attention for the first time, we make for this requirement the assumption that the set

$$S = \{\tau \in T : |\Gamma^\tau| \geq h(k-1)\}$$

is g_k-small above the current σ (again, note that this is a Σ_1 assumption so if it is false it will be discovered to be so at some finite stage).

(b) If it does not receive attention for the first time, we check whether the current assumption made for this requirement still appears to be true at stage k.

(1) If it does, we maintain this assumption and simply pick the value of $\sigma(k)$ at random between 0 and $h(k)-1$.

(2) If the assumption is discovered to be false, we increase our 'error counter' c_i by 1.

(i) If the new value of c_i remains less than n_i, we forget our previous assumption for requirement $\mathcal{R}_i$ and make a new assumption: we now assume that the set $S = \{\tau \in T : |\Gamma^\tau| \geq h(k-1)\}$ is g_k-small above (the current) σ.

(ii) If the new value of c_i is equal to n_i, we then wait until we find, for some ρ of length $h(k-1)$, a set $S_\rho = \{\tau \in S \mid \Gamma^\tau \succeq \rho\}$ which is $\left(g_k/2^{h(k-1)}\right)$-big above σ. When this happens, i.e., when we find a finite subtree T' of T of stem σ which is $\left(g_k/2^{h(k-1)}\right)$-bushy above σ and all of whose leaves are in S_ρ, we choose the next values of σ by a downward random walk restricted to T' until we reach a leaf of T' (note that we may never find such a tree in which case our algorithm gets stuck at this stage and thus fails to even return an infinite sequence). When a leaf is reached, we mark the i-th requirement as satisfied.

The sequence A returned by the algorithm is the minimal element of $\omega^{\leq\omega}$ extending all the values taken by σ throughout the algorithm. We now turn to the verification of our algorithm, which we have already done for the most part in Section 2.1. We do this via a series of claims.

Claim 1. The probability that the algorithm gets stuck at some substage of type (b.2.ii) (waiting to find a big tree T' which does not exist) is at most 2^{-m+1}.

Proof. This is standard fireworks calculation: all other randomly chosen values being fixed, there at most one value of n_i which causes the algorithm to get stuck at (b.2.ii) because of requirement $\mathcal{R}_i$. And since n_i is chosen randomly between 1 and 2^{i+m}, the probability that the algorithm gets stuck at (a.2.ii) because of requirement $\mathcal{R}_i$ is at most 2^{-i-m}. Thus, over all requirements, this gives a probability of at most 2^{-m+1}. □

Claim 2. Conditionally to our algorithm returning an infinite sequence, the probability that all requirements are met is at least $1 - 2^{-m+2}$.

Proof. Let us look at a given requirement $\mathcal{R}_i = \mathcal{R}_{\Gamma,d}$. This requirement receives attention infinitely often until satisfied. This means that if the algorithm does not get stuck, one of the following happens

- at some point it makes for $\mathcal{R}_i$ a correct assumption during substep (a) or (b.2.i) or,
- the i-th requirement causes the algorithm to enter some substep (b.2.ii) but a tree T' is found thereafter.

These two cases are mutually exclusive. In the first case, for some k a set

$$S = \{\tau \in T : |\Gamma^\tau| \geq h(k-1)\}$$

is correctly assumed to be g_k-small above the current σ. For any later stage of the algorithm (i.e., at stages $i \geq k$), the value of $\sigma(i)$ is chosen at random among $\tilde{h}(i)$ values, where $\tilde{h}(i)$ is either equal to $h(i)$ or to $g_{k'}(i)/2^{h(k'-1)}$ for some $k' > k$ in case some other strategy has caused a temporary restriction of the tree. The latter quantity is the smaller of the two, and by definition of the g_k's, it follows that $\tilde{h}(i) \geq 2^{i+m} g_k(i)$.

By the calculations of the proof of Lemma 4, it follows that in this case, the probability of hitting a node of S at some later stage is at most

$$1 - \prod_{i\geq k}\left(1 - \frac{g_k(i)}{\tilde{h}(i)}\right) \leq 1 - \prod_{i\geq k}\left(1 - \frac{g_k(i)}{2^{i+m} g_k(i)}\right) \leq \sum_{i\geq k} 2^{-i-m} \leq 2^{-k-m+1}$$

In the second case the requirement $\mathcal{R}_i$ is always satisfied. Indeed, in this case, we find a string ρ^* of length $h(k-1)$ and finite tree T' whose leaves are contained in $S_{\rho^*} = \{\tau \in S \mid$

$\Gamma^\tau \succeq \rho^*\}$ and then make a random walk within T' until we reach a leaf. As explained in the previous section, we then have

$$K(\rho^*) \leq K(\sigma) + K(\Gamma) + c \leq 2 \sum_{i \leq k-1} \log h(i) + c' + K(\Gamma)$$

for some fixed constants c, c'. But $|\rho^*| = h(k-1)$, so

$$K(\rho^*) - |\rho^*| \leq 2 \sum_{i \leq k-1} \log h(i) + K(\Gamma) - h(k-1) + c'' \tag{2}$$

for some fixed c''. By construction of h, $h(k-1) - 2\sum_{i \leq k-1} \log h(i)$ tends to ∞, thus for k large enough (and this 'large enough' can be found computably), the value of above expression is less than $-d$, which means that the requirement $\mathcal{R}_{\Gamma,d}$ is satisfied as soon as we reach a leaf of T'. We thus add the technical extra assumption that requirement $\mathcal{R}_{\Gamma,d}$ is only allowed to receive attention at stage k if in the above expression the right-hand side is smaller than $-d$. This essentially changes nothing since it only prevents every requirement to receive attention for finitely many stages.

Given a requirement $\mathcal{R}_{\Gamma,d}$, we say that *stage k is good for $\mathcal{R}_{\Gamma,d}$* when after having built $\sigma{\restriction}k$, in the next iteration of the loop, $\mathcal{R}_{\Gamma,d}$ receives attention and either a true assumption is made at steps (a) or (b.2), or stage (b.2.ii) is reached and a tree T' is found. To every requirement corresponds exactly one good stage (and no two requirements have a good stage in common). As we have just argued, if k is the good stage for a requirement, the probability that the requirement is satisfied, conditional to the algorithm returning an infinite sequence, is at least $1 - 2^{-k-m+1}$. Over all requirements, this gives a probability of at least $1 - \sum_k 2^{-k-m+1} = 1 - 2^{-m+2}$. □

Claim 3. The probability that we hit a node of B_{DNC} during the algorithm is at most 2^{-m+1}.

Proof. There is at most one 'bad' value of $\sigma(n)$ the algorithm can choose (namely, $\varphi_n^X(n)$, if it is defined). Whenever a value $\sigma(n)$ is chosen at random for some n, it is either chosen at random between 0 and $h(n) - 1$ or, in case of a temporary restriction to a subtree, between 0 and $g_k(n)/2^{h(k-1)} - 1$ for some $k \leq n$ (in case of a temporary restriction of the tree). Both quantities are at least 2^{n+m}, by construction. This gives a total probability of at most $\sum_n 2^{-n-m} = 2^{-m+1}$ of hitting B_{DNC}. □

The theorem immediately follows from the three claims: the probability that an infinite sequence A is returned and all requirement are satisfied is at least $1 - 2^{-m+1} - 2^{-m+2} - 2^{-m+1} = 1 - 2^{-m+3}$.

2.3 How much randomness do we need?

It remains to conduct, like in Section 1.4, an analysis of the level of algorithmic randomness needed to make the algorithm work. The attentive reader will notice that there are two uses

of randomness in the construction: the first one to choose the sequence (n_i) which will make the fireworks argument work (i.e., the algorithm won't get stuck), and the second one which helps choosing a node of the tree at random during the construction. For a given k, consider the algorithm Γ_k with probability of success at least $1 - 2^{-k}$. There are three ways in which it can fail:

1. It could get stuck at some stage b.2.ii.
2. It could hit a node which belongs to B_{DNC}.
3. It could make at some stage k a true assumption that a set S is g_k-small, but nonetheless hit a node of S later on (when it hits a node of the set S corresponding to a wrong assumption, this is not a problem because the assumption will be discovered to be wrong later on and a new assumption will be made for the requirement).

Technically, the occurrence of the third case does not necessarily mean that the algorithm has failed, but if neither of these three cases occur the algorithm succeeds, as explained above. The total probability of such events is at most 2^{-k}. Moreover, if any event of the above three types happens, it does so at some finite stage, thus after having used only finitely many bits of the random oracle. The open set of oracles that cause events of type 1 and 3 to happen can be effectively enumerated relatively to $\emptyset'$. Indeed, for the first type this is exactly what is explained in Section 1.4, namely that $\emptyset'$ can check at any given given stage whether the algorithm is stuck at stage b.2.ii. The third type can also be checked using $\emptyset'$: indeed, the sets we assume to be g_k-small are c.e. sets and since g_k is computable uniformly in k, the smallness can be checked using $\emptyset'$ and checking whether a node chosen at some stage of the algorithm is in S can also be done effectively in $\emptyset'$ (since S is c.e.). Thus we can design a $\emptyset'$-Martin-Löf test $(\mathcal{U}_k^{\emptyset'})_{k\in\mathbb{N}}$ such that $\mathcal{U}_k^{\emptyset'}$ covers the set of oracles which make the algorithm Γ_k fail because of cases 1 or 3.

The second type of failure is even easier to analyse. The set B_{DNC} is X-c.e., so the set of oracles which cause the algorithm to hit a node of B_{DNC} is effectively open relative to X. Thus we can design a X-Martin-Löf test $(\mathcal{V}_k^X)_{k\in\mathbb{N}}$ such that $\mathcal{V}_k^X$ covers the set of oracles which make the algorithm Γ_k fail because of case 2.

This finishes the proof of Theorem 2: if Z is X-random and 2-random, for k large enough it will be outside $\mathcal{U}_k^{\emptyset'}$ and outside $\mathcal{V}_k^X$, hence the algorithm Γ_k will succeed on input Z.

3 Further results

The proof of Theorem 2 can be adapted to prove more results on the class of DNC functions in V'Yugin's algebra. For example, in this proof, we construct a real of DNC^X degree which computes no Martin-Löf random real, but we do so using Kolmogorov complexity in a rather liberal way. By being very slightly more precise we can get a stronger result.

Theorem 5 Let $X \in 2^\omega$. For every fixed computable function h_0, for every sufficiently fast-growing h, every real Z which is both X-Martin-Löf random and 2-random computes a DNC_h^X function which computes no DNC_{h_0} function.

This is a stronger theorem than Theorem 2 because, as we saw in Section 1.1, when h_0 is sufficiently fast-growing, every Martin-Löf random real computes a DNC_{h_0} function. Again, the fact that $\mathbf{dnc}_{h_0}$ is strictly contained in $\mathbf{dnc}_{h_1}$ when h_1 grows sufficiently faster than h_0 is known (see [13]), but our theorem shows that this separation holds in V'Yugin's algebra as well.

The first thing to do to adapt our previous proof is to use the relationship between Kolmogorov complexity and DNC functions discovered by Kjos-Hanssen et al. [14]. For a function $h : \mathbb{N} \to \mathbb{N}$, call *$h$-complex* a real $A \in 2^\omega$ such that $K(X{\restriction}h(n)) \geq n - O(1)$. Call a real *complex* if it is h-complex for some computable h. Kjos-Hanssen et al. proved that a real computes a DNC function with computable bound if and only if it computes a complex real. More precisely: for any computable h_0, for any computable h_1 which grows sufficiently faster than h_0, if a real A computes a DNC_{h_0} function, it computes an h_1-complex real, and if A computes an h_0-complex real, it computes a DNC_{h_1} function.

Proof of Theorem 5. By the correspondence between DNC functions and complex reals, it suffices to show the following: For every fixed computable function h_0, for every sufficiently fast-growing h, every real Z which is both X-Martin-Löf random and 2-random computes a DNC_h^X function which computes no h_0-complex function. We modify the proof of Theorem 2 as follows. The requirements now become:

$$\mathcal{R}_{\Gamma,d}\text{: either } \Gamma^A \text{ is partial or there is an } n \text{ such that } K(\Gamma^A{\restriction}h_0(n)) < n - d$$

The new functions g_i's are defined by

$$g_k(i) = \begin{cases} 1 \text{ if } i < k \\ g_{k-1}(i) \cdot 2^{h_0(g_{k-1}(k-1))+i+m} \text{ otherwise} \end{cases}$$

and again, $h(k) = g_k(k)$ for all k. The sets S considered in Step (a) of the algorithm are now

$$S = \{\tau \in T : |\Gamma^\tau| \geq h_0(h(k-1))\}$$

The rest of the construction remains the same. The estimate (1) is left unchanged by this modification, but we now have $|\rho^*| \geq h_0(h(k-1))$. Together with (1), for k large enough, this guarantees the satisfaction of $\mathcal{R}_{\Gamma,d}$ (where Γ is the reduction with respect to which ρ^* is defined). □

Using another adaptation of the proof of Theorem 2, we can also transfer to V'Yugin's algebra the following result, due to Miller (see [13, section 3]): there exists a DNC function which computes no complex real. Namely, the following holds.

Theorem 6 Let $X \in 2^\omega$. Every real Z which is both X-Martin-Löf random and 2-random computes a DNC^X function which computes no DNC_h function for any computable h.

Although the general structure is similar, this second adaptation is not straightforward and quite a number of important changes are needed. The first thing to notice is that there is obviously no hope to conduct the whole construction in an h-bushy tree for a computable h since we want A to compute no complex real, which is equivalent to computing no computably bounded DNC function. Thus we will need to work in the full tree ω^ω and at each level k, choose dynamically the interval $[0, h(k)]$ for the random choice of $\sigma(k)$.

For this construction, the requirements are of the form:

$\mathcal{R}_{\Gamma,\varphi,d}$: Γ^A is partial, or φ is partial, or there is an n such that $K(\Gamma^A{\restriction}\varphi(n)) \leq n - d$

where the Γ's are still Turing functionals from ω^ω to 2^ω, and the φ's are partial computable functions from $\mathbb{N}$ to $\mathbb{N}$. How can we satisfy a single requirement $\mathcal{R}_{\Gamma,\varphi,d}$? Again, suppose some string σ of length k has already been built, and consider the set

$$S = \left\{\tau \in 2^{<\omega} : |\Gamma^\tau| \geq \varphi(r+d)\right\}$$

where r is an upper bound for the Kolmogorov complexity of $(\Gamma, d, \varphi, \sigma)$ (which can be found computably since there are computable upper bounds of prefix-free Kolmogorov complexity). By convention, this set is empty if $\varphi(r+d)$ is undefined. Again, let us analyze the different cases and how to succeed in each of them.

- Case 1: $\varphi(r+d)$ is undefined. Then the requirement is satisfied vacuously.
- Case 2: $\varphi(r+d)$ is defined and S is $\left(2^{\varphi(r+d)} \cdot g_0\right)$-small above σ, where g_0 is the function defined in previous constructions. In this case, it suffices to choose a function $g_1 \gg 2^{\varphi(r+d)} \cdot g_0$ and for all $k' \geq k$ pick the value of $\sigma(k')$ at random between 0 and $g_1(k')$. By smallness of S, using Lemma 4 as usual, we will avoid the set S with high probability, thus satisfying requirement $\mathcal{R}_{\Gamma,\varphi,d}$.
- Case 3: $\varphi(r+d)$ is defined and S is $\left(2^{\varphi(r+d)} \cdot g_0\right)$-big above σ. Each element $\tau \in S$ is such that $|\Gamma^\tau| \geq \varphi(r+d)$, therefore we can decompose S as

  $$S = \bigcup_{|\rho|=\varphi(r+d)} S_\rho \quad \text{where} \quad S_\rho = \{\tau \in S \mid \Gamma^\tau \succeq \rho\}$$

  and since there are $2^{\varphi(r+d)}$ strings of length $\varphi(r+d)$, there must be a ρ^* such that S_{ρ^*} is g_0-big above σ and such a ρ^* can be found effectively knowing $(\Gamma, d, \varphi, \sigma)$, hence by the choice of r,

  $$K(\rho^*) \leq r$$

  We have $|\rho^*| = \varphi(r+d)$, so for m large enough, this guarantees $K(\rho^*{\restriction}\varphi(r+d)) \leq r$, thus satisfying requirement $\mathcal{R}_{\Gamma,\varphi,d}$ with $n = r + d$.

We want to use a fireworks argument to help us choose between these three cases, but some care is needed since we no longer have a Σ_1/Π_1 dichotomy. Indeed Case 2 is neither Σ_1 nor Π_1. The solution is to introduce a priority between passive guesses. We will first make a

number of assumptions that $\varphi(r+d)$ is undefined at different stages. If all of these assumptions turn out to be wrong (= the error counter reaches its cap), we will then make the assumption that $\varphi(r+d)$ is defined at the current stage, wait for the value $\varphi(r+d)$ to be defined, and only then make *one* assumption that the current S is $\left(2^{\varphi(r+d)} \cdot g_0\right)$-small above the current σ. If proven wrong, we will begin another round of assumptions that $\varphi(r+d)$ is undefined (using a new error counter with a new cap) before making a new 'Case 2' assumption. Finally, when many of these 'Case 2' assumptions are proven to be wrong, we make one last assumption, a 'Case 3' assumption, and if everything goes well we will satisfy the requirement $\mathcal{R}_{\Gamma,\varphi,d}$. Thus, for any given requirement, our sequence of assumptions will look like this:

C1, C1, ..., C1, C2, C1, ..., C1, C2, C1, ..., C1, C2, ..., C2, C1, C1, ..., C1, C3

(where Ci=Case i) unless one of the C1/C2 assumptions is never proven to be wrong, in which case we succeed. This time there are two possible ways for the algorithm to get stuck: either wrongly assume in Case 2 that $\varphi(r+d)$ is defined, and then wait forever for it to converge, or like in the previous proofs, get stuck because of a wrong C3 assumption, waiting in vain to find a big subtree with leaves in a given c.e. set. The probability of either of these events happening can be made arbitrarily small by a fireworks argument.

The idea to handle several requirements at the same time is similar to what was done in our previous constructions, but this time is dynamic: Before making a C2/C3 assumption of type 'the following set S is small/big', we need to dynamically decide what 'big/small' should mean. What we do is first look at what other smallness assumptions are currently being made for other requirements. If there are l current assumptions of type 'S_i is g_i-small' for $i \leq l$, then we first choose a function G much larger than $g_1 + ... + g_l$ and evaluate the smallness of S in terms of this new function. In case S is then assumed to be small it is just added to the list of current assumptions. In case it is correctly assumed to be big, since G is much larger than the other g_i's, the probability that we hit one of the S_i during the temporary restriction of the tree will be, as in the previous proofs, close to 0.

While we hope that the reader is already convinced at this point, we provide the formal details for completeness.

Theorem 7 Let $X \in 2^\omega$ and h be a sufficiently fast-growing computable function. For every rational $\varepsilon > 0$, one can effectively design a probabilistic algorithm which, with probability at least $1 - \epsilon$, produces an $A \in \omega^\omega$ such that (1) A is DNC^X and (2) A computes no complex real.

The algorithm with parameter $m = m(\epsilon)$ is the following.

Stage 0: Initialization. First, for each requirement $\mathcal{R}_i$, pick a number $n(i,1)$ at random between 1 and $2^{\langle i,1,0\rangle + m}$. The number $n(i,1)$ is meant to be a cap for the number of wrong C2 assumptions for requirement $\mathcal{R}_i$. As well, for each integer b between 0 and $n(i,1)$, pick a number $n(i,2,b)$ at random between 1 and $2^{\langle i,2,b\rangle + m}$. Each time C2 makes a wrong assumption, C1 starts a new series of assumptions with $n(i,2,b)$ as a new cap, where b is the

number of wrong C2 assumptions. Create two counters c_i, c'_i, initialized at 0 (c_i counts the number of wrong C2 assumptions for requirement $\mathcal{R}_i$ and c'_i counts the number of wrong C1 assumption during the current run of such assumptions for requirement $\mathcal{R}_i$). Let $\mathcal{L}$ be a list of assumptions (coded as integers, with at most one assumption per requirement), initially empty. Finally, initialize σ to be the empty string.

Loop (to be repeated indefinitely). Suppose that the values $\sigma(0), ..., \sigma(k-1)$ of σ are already defined and some requirement $\mathcal{R}_i = \mathcal{R}_{\Gamma,\varphi,d}$ receives attention. Let r be an upper bound of the Kolmogorov complexity of the current tuple $(\sigma, \Gamma, \varphi, d, \mathcal{L})$. Let $g_0, ..., g_l$ be the computable functions such that an assumption 'S is g_i-small' is currently in $\mathcal{L}$. We (locally) define a function G by $G(u) = 2^{u+m}(g_0(u) + \ldots + g_l(u))$.

(a) If this requirement receives attention for the first time, we make for this requirement the assumption that $\varphi(r+d)$ is undefined, and add this assumption to $\mathcal{L}$.

(b) If it does not receive attention for the first time, we check whether the current assumption made for this requirement still appears to be true at stage k.

  (1) If it does, we maintain this assumption and simply pick the value of $\sigma(k)$ at random between 0 and $G(k)$.

  (2) If the assumption is discovered to be false, and it was a C1 assumption (φ undefined on some value), we remove this assumption from $\mathcal{L}$ and increase c'_i by 1.

    (i) If the new value of c'_i remains less than $n(i, 2, c_i)$, we make a new assumption: we now assume that $\varphi(r+d)$ is undefined and add this assumption to $\mathcal{L}$.

    (ii) If the new value of c'_i reaches $n(i, 2, c_i)$, we wait for $\varphi(r+d)$ to become defined. We then make the assumption that the set $S = \{\tau \in 2^{<\omega} \mid |\Gamma^\tau| \geq \varphi(r+d)\}$ is $(2^{\varphi(r+d)} \cdot G)$-small above σ and add this assumption to $\mathcal{L}$.

  (3) If the assumption is discovered to be false, and it was a C2 assumption (smallness of some set S), we remove this assumption from $\mathcal{L}$, and increase c_i by 1.

    (i) If the new value of c_i remains less than $n(i, 1)$, we make a new assumption: we now assume that $\varphi(r+d)$ is undefined, add this assumption to $\mathcal{L}$, and reset c'_i to 0.

    (ii) If the new value of c_i is equal to $n(i, 1)$, we then wait until we find, for some ρ of length $\varphi(r+d)$, a set $S_\rho = \{\tau \in S \mid \Gamma^\tau \succeq \rho\}$ which is G-big above σ. When this happens, i.e., when we find a finite tree T of stem σ which is G-bushy above σ, we choose the next values of σ by a downward random walk restricted to T until we reach a leaf of T. When a leaf is reached, we mark the i-th requirement as satisfied.

The sequence A returned by the algorithm is the minimal element of $\omega^{\leq\omega}$ extending all the values taken by σ throughout the algorithm.

The verification is similar to the previous proofs.

Claim 4. The probability that the algorithm gets stuck at some substage of type (b.2.ii) or (b.3.ii) is at most 2^{-m+1}.

Proof. Indeed, by the standard verification of fireworks arguments, the probability that the algorithm gets stuck at the end of the b-th run of C1 assumptions for requirement i (making a bad assumption that φ is defined on some value) is, by construction, $2^{-\langle i,2,b\rangle-m}$. Likewise, the probability that it gets stuck because of a bad C3 assumption for requirement i is $2^{-\langle i,1,0\rangle-m}$. Since the sum of the terms $2^{-\langle i,a,b\rangle-m}$ is 2^{-m+1}, we have the desired result. □

Claim 5. Conditionally to algorithm returning an infinite sequence, the probability that all requirements are met during the construction is at least $1-2^{-m+2}$.

Proof. Fix a requirement $\mathcal{R}_i = \mathcal{R}_{\Gamma,\varphi,d}$. This requirement receives attention infinitely often until satisfied. This means that if the algorithm does not get stuck, one of the following happens

- At some point it makes for $\mathcal{R}_i$ a correct assumption that φ is undefined on some value during substep (a) or (b.3.i).
- At some point it makes for $\mathcal{R}_i$ a correct assumption that $S = \{\tau \in 2^{<\omega} \mid |\Gamma^\tau| \geq \varphi(r+d)\}$ is $(2^{\varphi(r+d)} \cdot g)$-small above σ for some computable g.
- the i-th requirement causes the algorithm to enter some substep (b.3.ii) but a tree T is found thereafter.

In the first case, the requirement is satisfied vacuously. In the second case, we have a set $S = \{\tau \in 2^{<\omega} \mid |\Gamma^\tau| \geq \varphi(r+d)\}$ which is correctly assumed to be $(2^{\varphi(r+d)} \cdot g)$-small above σ for some computable g. For any $i \geq k$, the value of $\sigma(i)$ is chosen at later stages of the algorithm at random among at least $G(i)$-many values, where G is chosen to be greater than $2^{i+m} \cdot h$ for any function h appearing in the list $\mathcal{L}$, which in particular includes g (since the assumption featuring g is correct, it is never removed from the list). By Lemma 4, it follows that the probability of hitting a node of S at some later stage is at most

$$1 - \prod_{i\geq k}\left(1 - \frac{g(i)}{2^{i+m}g(i)}\right) = 1 - \prod_{i\geq k}\left(1 - \frac{1}{2^{i+m}}\right) \leq \sum_{i\geq k} 2^{-i-m} \leq 2^{-k-m+1}$$

In third case, we find a string ρ^* of length $\varphi(r+d)$ and G-bushy finite tree T whose leaves are contained in $S_{\rho^*} = \{\tau \in S \mid \Gamma^\tau \succeq \rho^*\}$. The string ρ^* can be effectively found knowing Γ, φ, d, the current value of σ and G. But G can be computed knowing the list $\mathcal{L}$, and the integer r is precisely defined to bound the complexity of the tuple $(\sigma, \Gamma, \varphi, d, \mathcal{L})$, thus

$$K(\rho^*) \leq r$$

But $|\rho^*| = \varphi(r+d)$, so the requirement $\mathcal{R}_{\Gamma,\varphi,d}$ is satisfied for $n = r+d$. Note that to be completely rigorous, when we say that 'r bounds the Kolmogorov complexity of $(\sigma, \Gamma, \varphi, d, \mathcal{L})$', we also need r to be large enough to overcome the additive constants that will arise in the calculation of the bound of $K(\rho^*)$. This is done, as usual, by picking r 'large enough' (which can be achieved computably), or using a fixed-point argument.

Now, given a requirement $\mathcal{R}_{\Gamma,\varphi,d}$, we say that *stage k is good for $\mathcal{R}_{\Gamma,d}$* when after having built $\sigma{\restriction}k$, the next iteration of the loop, $\mathcal{R}_{\Gamma,\varphi,d}$ receives attention and a true assumption is

made or stage (b.3.ii) is reached and a tree T is found. To every requirement corresponds exactly one good stage (and no two requirements have a good stage in common). As before, if k is the good stage for a requirement, the probability that the requirement is satisfied, conditional to the algorithm returning an infinite sequence, is at least $1 - 2^{-k-m+1}$ and over all requirements, this gives a probability of at least $1 - \sum_k 2^{-k-m+1} = 1 - 2^{-m+2}$. □

Claim 6. The probability that we hit a node of B_{DNC} is at most 2^{-m+1}.

Proof. For every n the value of $\sigma(n)$ is chosen at random among at least 2^{n+m} values, so the probability of picking $\varphi_n^X(n)$, if defined, is at most 2^{-n-m}, thus a total bound of $\sum_n 2^{-n-m} = 2^{-m+1}$. □

Thus the probability of success of the algorithm is at least $1 - 2^{-m+3}$. To get Theorem 6, it remains to evaluate the level of algorithmic randomness needed, but the situation is essentially the same as in previous proofs. Checking whether the algorithm gets stuck at a given stage can be tested effectively in $\emptyset'$ (with $\emptyset'$ we can check whether partial functions are defined or not, and we can check bigness/smalllness of c.e. sets of strings) and whether it hits B_{DNC} can be tested using X as discussed before.

4 Conclusion

The above results provide a clear picture of the hierarchy of DNC notions in V'Yugin's algebra. Namely, for every computable function h_0 which grows fast enough, and computable h_1 which grows fast enough compared to h_0, we have the strict inclusions:

$$\mathbf{mlr} \sqsubset \mathbf{dnc}_{h_0} \sqsubset \mathbf{dnc}_{h_1} \sqsubset \mathbf{complex} \sqsubset \mathbf{dnc}$$

Moreover, every 2-random real computes a witness for each of the four separations. While this is an interesting result in and of itself, the techniques we employed to prove these results are without doubt the main contribution of this paper. They illustrate the power and flexibility of fireworks arguments and show that they interact well with complex forcing notions. What is also interesting is that 'true randomness' seems to be needed for our fireworks arguments. Other known fireworks arguments, such as the proof that every 2-random real computes a hyperimmune set or that every 2-random computes a 1-generic real require only a very weak form of randomness. Indeed, for both of these constructions, Barmpalias et al. [2] showed that it suffices to have a non-computable real which is *Turing-below* a 2-random, despite the fact that a real which is simply below a 2-random may have very little randomness content. Having a non-computable real which is below a 2-random real is not sufficient in our case: Indeed no 1-generic real computes a DNC function (see for example [8]), and as we just said, there are 1-generic reals which are below a 2-random real.

We believe that fireworks arguments will yield more applications in the future. In the constructions of this paper, the strategies for different requirements do interact, but there is

no injury per se. It would be very interesting to find examples of situations where fireworks arguments can be mixed with finite/infinite injury constructions.

Acknowledgements. We are thankful to Peter Gács and Alexander Shen for enlightening us on fireworks arguments, to Mushfeq Khan for having made early versions of his survey available to us, and to Chris Porter and Rupert Hölzl for having introduced us to the Levin-V'Yugin algebra and asked the question which ultimately led to this paper. We would also to thank two anonymous referees for their many useful comments and suggestions.

References

[1] Klaus Ambos-Spies, Bjørn Kjos-Hanssen, Steffen Lempp, and Theodore A. Slaman. Comparing DNR and WWKL. *Journal of Symbolic Logic*, 69(4):1089–1104, 2004.

[2] George Barmpalias, Adam Day, and Andy Lewis-Pye. The typical Turing degree. *Proceedings of the London Mathematical Society*, 109(1):1–39, 2013.

[3] Laurent Bienvenu and Christopher P. Porter. Deep Π^0_1 classes. arXiv preprint (arXiv:1403.0450), 2014.

[4] Mingzhong Cai. *Elements of Classical Recursion Theory: Degree-Theoretic Properties and Combinatorial Properties*. PhD thesis, Cornell University, 2011.

[5] Karel de Leeuw, Edward F. Moore, Claude Shannon, and Norman Shapiro. Computability by probabilistic machines. In *Automata Studies*. Princeton University Press, 1956.

[6] Oswald Demuth. Remarks on the structure of tt-degrees based on constructive measure theory. *Commentationes Mathematicae Universitatis Carolinae*, 29(2):233–247, 1988.

[7] François G Dorais, Jeffry L Hirst, and Paul Shafer. Comparing the strength of diagonally non-recursive functions in the absence of Σ^0_2 induction. *Journal of Symbolic Logic*, 80:1211–1235, 2015.

[8] Rodney Downey and Denis Hirschfeldt. *Algorithmic randomness and complexity*. Theory and Applications of Computability. Springer, 2010.

[9] Noam Greenberg and Joseph S. Miller. Diagonally non-recursive functions and effective Hausdorff dimension. *Bulletin of the London Mathematical Society*, 43(4):636–654, 2011.

[10] Carl Jockusch and Robert Soare. Π^0_1 classes and degrees of theories. *Transactions of the American Mathematical Society*, 173:33–56, 1972.

[11] Steven M. Kautz. *Degrees of random sequences*. PhD thesis, Cornell University, 1991.

[12] Mushfeq Khan. Shift-complex sequences. *Bulletin of Symbolic Logic*, 19(2):199–215, 2013.

[13] Mushfeq Khan and Joseph S. Miller. Forcing with bushy trees. arXiv preprint (arXiv:1503.08870), 2015.

[14] Bjørn Kjos-Hanssen, Wolfgang Merkle, and Frank Stephan. Kolmogorov complexity and the recursion theorem. *Transactions of the American Mathematical Society*, 363(10):5465–5480, 2011.

[15] Antonin Kučera. Measure, Π^0_1 classes, and complete extensions of PA. *Lecture Notes in Mathematics*, 1141:245–259, 1985.

[16] Masahiro Kumabe and Andrew E. M. Lewis. A fixed-point-free minimal degree. *Journal of the London Mathematical Society*, 80(3):785–797, 2009.

[17] Leonid Levin and Vladimir V'yugin. Invariant properties of informational bulks. *Lecture Notes in Computer Science*, 53:359–364, 1977.

[18] Jeffrey Paris. Measure and minimal degrees. *Annals of Mathematical Logic*, 11:203–216, 1977.

[19] Andrei Rumyantsev and Alexander Shen. Probabilistic constructions of computable objects and a computable version of Lovász local lemma. arXiv preprint (arXiv:1305.1535), 2013.

[20] Andrey Yu. Rumyantsev. Everywhere complex sequences and the probabilistic method. In *STACS*, volume 9 of *LIPIcs*, pages 464–471, 2011.

[21] Stephen G. Simpson. Mass problems associated with effectively closed sets. *Tohoku Mathematical Journal*, 63:489–517, 2011.

[22] Vladimir V'yugin. On Turing invariant sets. *Soviet Mathematics Doklady*, 17:1090–1094, 1976.

[23] Vladimir V. V'yugin. Algebra of invariant properties of binary sequences. *Problemy Peredachi Informatsii*, 18(2):83–100, 1982.