On Low for Speed Oracles

Laurent Bienvenu
LIRMM
CNRS & Université de Montpellier, France

Rod Downey
School of Mathematics and Statistics
Victoria University of Wellington, New Zealand

October 8, 2018

Abstract

Relativizing computations of Turing machines to an oracle is a central concept in the theory of computation, both in complexity theory and in computability theory(!). Inspired by lowness notions from computability theory, Allender introduced the concept of "low for speed" oracles. An oracle A is low for speed if relativizing to A has essentially no effect on computational complexity, meaning that if a decidable language can be decided in time $f(n)$ with access to oracle A, then it can be decided in time $poly(f(n))$ without any oracle. The existence of non-computable such A's was later proven by Bayer and Slaman, who even constructed a computably enumerable one, and exhibited a number of properties of these oracles as well as interesting connections with computability theory. In this paper, we pursue this line of research, answering the questions left by Bayer and Slaman and give further evidence that the structure of the class of low for speed oracles is a very rich one.

1 Introduction

The subject of this paper is oracle computation, more specifically the effect of oracles on the speed of computation. There are many notable results about oracles in classical complexity, beginning with the Baker-Gill-Solovay result [BGS75] which asserts that there are oracles A such that $\mathsf{P}^A = \mathsf{NP}^A$, but that there are also oracles B such that $\mathsf{P}^B \neq \mathsf{NP}^B$ (thus demonstrating that methods that relativize will not suffice to solve basic questions like P vs NP). An underlying question is whether oracle results can say things about complexity questions in the unrelativized world. Eric Allender and his co-authors [ABK06, AFG13] showed that oracle access to the sets of random strings could give insight into basic complexity questions. For example, in [AFG13], Allender et. al. showed that $\cap_U \mathsf{P}^{R_{K_U}} \cap \mathsf{COMP} \subseteq \mathsf{PSPACE}$ where

R_{K_U} denotes the strings whose prefix-free Kolmogorov complexity (relative to universal machine U) is at least their length, and COMP denotes the collection of computable sets. Later the "$\cap\mathsf{COMP}$" was removed by Cai et. al. [CDE+14]. Thus we conclude that reductions to very complex sets like the random strings somehow gives insight into very simple things like computable sets.

Inspired by lowness notions in computability theory, Allender asked whether there were non-trivial sets which were "low for speed" in that, as oracles, they did not accelerate running times of computations by more than a polynomial amount. Of course, as stated this makes little sense since using any X as oracle, we can decide membership in X in linear time, while without oracle X may not even be computable at all! Thus, what we are really interested in is the set of oracles which do not speed-up the computation of *computable sets* by more than a polynomial amount. More precisely, an oracle X is *low for speed* if for any computable language L, if some Turing machine M with access to oracle X decides L in time f, then there is a Turing machine M' without oracle and polynomial p such that M' decides L in time $p \circ f$. (Here computation time of oracle computation is counted in the usual complexity-theoretic fashion: we have a "query tape" on which we can write strings, and once a string x is written on this tape, we get to ask the oracle whether x belongs to it in time $O(1)$).

There are trivial examples of such sets, namely oracles that belong to P, because any query to such an oracle can be replaced by a polynomial-time computation. Allender's precise question was therefore:

Is there an oracle $X \notin \mathsf{P}$ which is low for speed?

Such an X, if it exists, has to be non-computable, for the same reason as above (if X is computable and low for speed, then X is decidable in linear time using oracle X, thus – by lowness – decidable in polynomial time without oracle, i.e., $X \in \mathsf{P}$).

A partial answer was given by Lance Fortnow (unpublished), who observed the following.

Theorem 1.1 (Fortnow)**.** *If X is a hypersimple and computably enumerable oracle, then X is low for polynomial time, in that if $L \in \mathsf{P}^X$, then $L \in \mathsf{P}$.*

Allender's question was finally solved by Bayer and Slaman, who showed the following.

Theorem 1.2 (Bayer-Slaman [Bay12])**.** *There are non-computable, computably enumerable, sets X which are low for speed.*

Once their existence is established, it is natural to wonder what kind of sets might be low for speed. A precise characterization seems currently out of reach, but it is interesting to see how lowness for speed interacts with other computability-theoretic properties. One needs however to keep in mind that lowness for speed is *not* closed under Turing equivalence: as we saw above that in the $\mathbf{0}$ degree (computable sets) some members are low for speed and others that are not (on the other hand it is easy to see that if A is polynomial-time reducible to B and B is low for speed, then A is also low for speed).

In his PhD thesis, Bayer showed that if X is computably enumerable and of promptly simple Turing degree, then X is *not* low for speed, but also proved that this did not characterize the computable enumerable oracles that are low for speed. Bayer also studied the size of the set of low for speed oracles, where 'size' is understood in terms of Baire category. Surprisingly, whether the set of low for speed oracles is meager or co-meager depends on the answer of the famous $\mathsf{P} =?\mathsf{NP}$ question.

In this paper, we continue Bayer and Slaman's investigation on the set of low for speed oracles. In the next section, we give an easier proof of the existence of non-computable low for speed oracles which does not require the full Bayer-Slaman machinery (but the oracle we construct is not computably enumerable). In Section 3, we focus on the computably enumerable low for speed oracles, and prove that – quite surprisingly – they cannot be low in the computability-theoretic sense, but can however be low_2. Finally, we pursue Bayer and Slaman's idea to study how large the set of low for speed oracles is, in terms of measure and category. In particular, we solve a question they left open by showing that the set of low for speed oracles has measure 0 and obtain some interesting connections with algorithmic randomness. Finally, though lowness for speed is not closed under Turing equivalence it is nonetheless natural to ask which Turing degrees contain a low for speed member, which is what Section 5 is about.

Throughout this paper, we will denote by $\{0,1\}^*$ the set of finite strings. In our setting, an oracle is a *language*, i.e., a subset of $\{0,1\}^*$; however, as is typical in computability theory, it is more convenient in some of the results we present below to view oracles as infinite binary sequences (whose set we denote by $\{0,1\}^\omega$), by first identifying finite strings with integers (the $(n+1)$-th string in the length-lexicographic order being identified with n) making the oracle a subset of $\mathbb{N}$ (which is what we mean when we talk about a 'set' without further precision) and then identifying the oracle with its characteristic sequence (the $(n+1)$-th bit is 1 if n belongs to the oracle, 0 otherwise. When building oracles X with certain computability-theoretic properties, viewed as infinite binary sequences, we will often need to refer to prefixes of X, which are themselves binary strings. To avoid confusion between *members* and *prefixes* of oracles, we will use latin letters $x, y, z, \ldots$ to denote members of oracles, and greek letters $\sigma, \tau, \ldots$ for prefixes of oracles. Two strings σ and τ are *incompatible* if for some $i < \min(|\sigma|, |\tau|)$, $\sigma(i) \neq \tau(i)$. We denote this by $\sigma \perp \tau$. The join $X \oplus Y$ of two infinite binary sequences X, Y is the sequence $X(0)Y(0)X(1)Y(1)\ldots$. Finally $X \upharpoonright n$ is the prefix of X of length n.

Our paper requires some knowledge of computability theory and algorithmic randomness. One can consult the book [DH10] for the results and concepts we allude to below. Our notation is mostly standard. We denote Cantor's pairing function by $\langle .,. \rangle$. We also fix an effective list (Φ_e) of all oracle Turing functionals (or machines: Φ_e^A is the Turing machine of index e with oracle A, which for a fixed A is a partial function from $\{0,1\}^*$ to $\{0,1\}$). For a given functional Φ_e and oracle A, $time(\Phi_e^A, x)$ denotes the running time of Φ_e on input x with oracle A (counting time according to the model of computation described above) and $time(\Phi_e^A)$ is the function $x \mapsto time(\Phi_e^A, x)$. We let (R_i) be an effective enumeration of all partial computable functions from $\{0,1\}^*$ to $\{0,1\}$. We denote the set of low for speed

oracles by LFS, and the subset of LFS consisting of its non-computable elements by LFS^*.

2 Existence of non-computable low for speed sets

In this section we will present a simple proof of the existence of a non-computable low for speed oracle. Define the set S of strings by $S = \{0^{2^n} \mid n \in \mathbb{N}\}$ and let $\mathbb{S}$ be the set of 'sparse' sets of strings which only contain strings from S, that is, $\mathbb{S} = \{X \in \{0,1\}^\omega \mid X \subseteq S\}$.

By extension, we say that a string σ is in $\mathbb{S}$ if it is a prefix of some element of $\mathbb{S}$. The interest of the set $\mathbb{S}$ is that there are only $O(n)$ strings in $\mathbb{S}$ of length n. Thus, given a Turing machine Φ, it is possible to simulate in time $poly(t)$ the behaviour of Φ^X during t steps of computation on all $X \in \mathbb{S}$ (an idea which is already present in the Bayer-Slaman argument presented in the next section).

Theorem 2.1. *There exists a non-computable X which is low for speed.*

Proof. We want X to satisfy all requirements $\mathcal{R}_{(e,i)}$, where e, i range over integers, defined as follows

$\mathcal{R}_{(e,i)}$: either R_i is partial, or $\Phi_e^X \neq R_i$, or $\Phi_e^X = R_i$ but the computation of R_i via Φ_e^X can be simulated by a functional Ψ running in time polynomial in $time(\Phi_e^X)$.

We build our oracle X by finite extension. Let σ_0 be the empty string. At stage $s+1 = \langle e, i\rangle$, do the following.

(a) If there is an n and a $\tau \in \mathbb{S}$ extending σ_s such that $\Phi_e^\tau(n)$ and $R_i(n)$ both converge and have different values, then let σ_{s+1} be the first (say in length-lexicographic order) such string τ.

(b) If there is no such string τ, then take $\sigma_{s+1} = \sigma_s 0$

Finally let X be the unique infinite sequence extending all σ_s. We claim that X is as wanted. Let us first prove that X must be incomputable. Suppose $X = R_i$ for a total R_i. Let e be an index such that Φ_e is the identity functional. By construction, when choosing the prefix τ of X at stage $s+1 = \langle e, i\rangle$, we must be in case (a), and thus τ is precisely chosen to ensure $X \neq R_i$, a contradiction. Let us now prove that X is low for speed. Fix a pair (e, i) let $s+1 = \langle e, i\rangle$, and let us see how σ_{s+1} was constructed. If we were in case (a) at that stage, we have ensured $\Phi_e^{\sigma_{s+1}} \perp R_i$ and thus $\Phi_e^X \perp R_i$, thereby satisfying $\mathcal{R}_{(e,i)}$. If we were in case (b), there are three subcases:

- Either R_i is partial, then the requirement $\mathcal{R}_{(e,i)}$ is satisfied
- Or there is an n such that $\Phi_e^\tau(n) \uparrow$ for any extension τ of σ_s, in which case $\Phi_e^X(n) \uparrow$ and thus $\Phi_e^X \neq R_i$ should R_i be total.

- Or, if we are in neither of the two above cases, for every n there is an extension τ of σ_s such that $\Phi_e^\tau(n)\downarrow$, and for any such τ, we have $\Phi_e^\tau(n) = R_i(n)$. In this case, we can build a functional Ψ which computes R_i as follows. On input n, at stage t, it computes $\Phi_e^\tau(n)$ during t steps of computation for all $\tau \in \mathbb{S}$ of length t extending σ_s. If a τ is found such that $\Phi_e^\tau(n)\downarrow$, then we set $\Psi(n) = \Phi_e^\tau(n)$. As we already mentioned, there are only $O(t)$ strings of length t in $\mathbb{S}$ and it is obvious that they can be listed in polynomial time. Hence, simulating all computations $\Phi_e^\tau(n)$ during t steps can be done in time $p(t)$ for some polynomial t. This shows that for any $Y \in \mathbb{S}$ extending σ_s, if $\Phi_e^Y(n)$ returns (the value of $R_i(n)$) in time t, this is found out by the procedure Ψ at stage t, which corresponds to $\sum_{s\leq t} p(s) + O(1)$ steps of computation for Ψ, which is also polynomial in t. This being true for any $Y \in \mathbb{S}$ extending σ_s, we have in particular that $time(\Psi) = poly(time(\Phi_e^X))$.

□

One should note that the case disjunction in this proof is a Σ_1/Π_1 dichotomy, and therefore one can carry out the construction below $\mathbf{0}'$, therefore establishing the existence of a $\mathbf{0}'$-computable set that is low for speed. This is weaker than the Bayer-Slaman result presented in the next section, which asserts the existence of a *c.e.* such set. However, this proof is both simpler and, as we will see in the remainder of the paper, has further useful corollaries.

3 Computably enumerable low for speed sets

We now restrict ourselves to the computably enumerable (c.e.) sets, and study which of these can be low for speed. For the sake of completeness, we present the main ideas of the proof of Bayer and Slaman [Bay12] that there are indeed c.e. sets in LFS^*.

Theorem 3.1 (Bayer-Slaman Theorem)**.** *There exist c.e. non-computable sets that are low for speed.*

Proof sketch. The proof uses a tree-of-strategies argument. We need to satisfy

$$\mathcal{P}_e : \overline{A} \neq W_e,$$

and

$$\mathcal{L}_{e,i} : \text{If } \Phi_e^A = R_i \text{ total, then some } \Psi \text{ computes } R_i \text{ in time polynomial in } time(\Phi_e^A).$$

The $\mathcal{P}_e$-strategy is a standard Friedberg-Muchnik strategy on a tree. A node ρ devoted to this requirement picks a fresh follower x, waits for $x \in W_e[s]$ and if this happens puts x into A.

The basic strategy for $\mathcal{L}_{e,i}$ is the following. First, throughout the whole construction of A, we will promise that if we add an element x to A at stage t, then we must immediately

also add all $y \in [x,t]$ (this is often referred to as a *dump construction*). This way, at any stage s, there will only be at most s strings α of length s that can potentially be a prefix of (the final) A. And thus – just like in the previous section – at stage s, it is possible to emulate all computations $\Phi_e^\alpha(x)[s]$ for all such α's and $x \leq s$ in time $poly(s)$.

When the strategy is eligible to act at stage s, for every $x \leq s$ on which Ψ is not defined yet, it computes all $\Phi_e^\alpha(x)[s]$ for all potential prefixes α of A, and should one of them converge, defines $\Psi(x)$ to be the value of $\Phi_e^\alpha(x)$ for the α that has the fastest convergence. If no $\Phi_e^\alpha(x)[s]$ converges, $\Psi(x)$ remains undefined until the strategy is eligible to act again.

Now, if at some later stage we find a value x such that $R_i(x)\downarrow$ and $\Psi(x) \neq R_i(x)$, then we find the α such that $\Psi(x) = \Phi_e^\alpha$ and add elements into A so that α becomes a prefix of A. This ensures $\Phi_e^A \neq R_i$ and terminates the strategy. All strategies of lower priorities are then injured and must be reset. If we never find such an x, this means that either Φ_e^A is partial, or R_i is, or $\Psi = \Phi_e^A = R_i$ and by construction the running time of Ψ is polynomial in the running time of Φ_e^A.[1]

This is enough to ensure the success of the strategy in isolation. The difficulty comes from the interaction with lower-priority strategies which might want to add elements into A. The final key to the Bayer-Slaman proof is the following. Suppose that at some stage s a strategy of lower priority wants to add an interval $[y,s]$ of elements into A. The problem is that the computations on this configuration might be *slow*. Perhaps for some x of length $\leq s$ we have not as yet seen $\Phi_e^{A_s \cup [y,s]}(x)\downarrow$. Even more importantly, we don't even know that the value of this will agree with the value $\Psi(x)$ we have already defined.

The idea is the following. R_i has to confirm the computations, that is, we wait until $R_i(x)$ converges on all x where Ψ has already been defined. When (and if) this happens, we must have $\Psi(x) = R_i(x)$ for all such x otherwise we would be in the above case where we can ensure $\Phi_e^A \neq R_i$ and satisfy the requirement. If this never happens, our requirement will be satisfied because R_i would be partial. And if it does happen, then we can safely add $[y,s]$ to A because if this causes $\Phi_e^A(x)$ to change, it will yield $\Phi_e^A(x) \neq R_i(x)$ which satisfies the requirement. But there is one last problem: while waiting for this confirmation, the construction of Ψ cannot wait as we need it to be as fast as Φ_e^A. The crucial trick is, from the point of view of our strategy, to carry on *as if* $[y,s]$ had already been enumerated into A. Indeed, if the confirmation ever happens, the elements of $[y,s]$ will be truly enumerated into A which Ψ will have correctly assumed ahead of time, and if the confirmation never happens, Ψ might be wrong (i.e., $\Psi \neq \Phi_e^A$) but this will not matter because in this case R_i will be either partial or different from Φ_e^A. Of course, when the confirmation never occurs the strategy of lower priority never gets to enumerate into A the elements it wants. This is where we make use of a standard tree construction where strategies of lower priorities guess the outcome of strategies of higher priority. We refer the reader to [Bay12] for details.

□

Within the c.e. sets, would one expect a set to be low for speed to have low or high

[1] Actually, there is a subtlety here: one must ensure that the strategy for $\mathcal{L}_{e,i}$ is eligible to act often enough, i.e., allowed to act for the n-th time before stage $q(n)$ for some polynomial q, but this can easily be ensured.

information content? In particular, one would expect that a low for speed c.e. set would be one with little computational power, in the same way that sets low for 1-randomness are all (super-)low (see Nies [Nie09]). The next theorem is therefore quite surprising.

Theorem 3.2. *If A is c.e. and of low Turing degree (i.e. $A' \equiv_T \emptyset'$), then A is* not *low for speed.*

Proof. Assume that A is not computable, is c.e., and is low. Let (Φ_e, p_e) be an enumeration of pairs of one functional and one polynomial with coefficients in $\mathbb{N}$. We will build a Turing functional Ψ and a computable set R such that $\Psi^A = R$. This is our global requirement and we make the following global commitment: if a value $R(n)$ gets defined at some stage, $\Psi^X(n)$ is immediately defined to be equal to $R(n)$ for all X's on which $\Psi^X(n)$ is still undefined. We want to satisfy, for each e:

$$(\mathcal{R}_e) : \Phi_e \text{ does not compute } R \text{ in time } p_e(time(\Psi^A))$$

thus proving that A is not low for speed. The strategy for a single requirement $(\mathcal{R}_e)$ is the following. Throughout the construction, we build a 'verifier', i.e., a partial computable S such that $S(e, .)$ is the attempt by the $(\mathcal{R}_e)$-strategy to guess A. We also define an auxiliary functional Θ common to all strategies whose index we know in advance, and use the lowness of A to obtain a computable 0-1 valued function $h(., .)$ such that $\lim_t h(e, t)$ exists for all e, and equals 1 when $\Theta^A(e) \downarrow$, 0 otherwise. (Informally, $\Theta^X(e) \downarrow$ means that a prefix of X is believed to be a prefix of A at some stage of the strategy for $(\mathcal{R}_e)$, and this will cause the strategy to enter Case 3 as described below.)

At the initial stage s_1, S is empty and we pick a first fresh witness w_1 larger than any integer mentioned so far in the construction and define $\Psi^{A_{s_1} \restriction 1}(w_1) = 0$. Let t_1 be the time this computation takes. Now, check whether $\Phi_e(w_1)$ returns in $p_e(t_1)$ steps. We distinguish three cases:

Case 1: $\Phi_e(w_1)$ returns 1 in $\leq p_e(t_1)$ steps. In this case, we set $R(w_1) = 0$ and $R(n) = 0$ for all $n \leq w_1$ on which R is still undefined, and commit to having $\Psi^A(w_1) = 0$ even after potential future A-changes. This way we ensure $\Phi_e \neq R = \Psi^A$, thus immediately satisfying $(\mathcal{R}_e)$, and we stop the strategy for this requirement.

Case 2: $\Phi_e(w_1)$ returns 0 in $\leq p_e(t_1)$ steps. In this case, we do not define $R(w_1)$ just yet. Instead, we set $S(e, 1) = A_{s_1} \upharpoonright 1$. We then create a second witness w_2 at stage s_2 and proceed as above for this new witness (with $A_{s_2} \upharpoonright 2$ in place of $A_{s_1} \upharpoonright 1$). And so on: for further occurrences of this case, the procedure will extend S and create a witness w_3 at stage s_3 looking at prefixes of length $l = 3$, etc (and if Case 2 then causes a reset, we stay at the same level l when resetting). Meanwhile, we continue to monitor A. Again, there are two subcases for a given l:

(a) At some point we discover that $A_{s_l} \upharpoonright i$ is not in fact an initial segment of A, we are then free to set $R(w_l) = 1$ (which will guarantee $\Psi^A(w_l) = 1 = R(w_l) \neq \Phi_e(w_l)$ since we only committed to $\Psi^\sigma(w_l) = 0$ for σ's that are not prefix of A), and this way we have satisfied $(\mathcal{R}_e)$. We then stop the strategy.

(b) $A_{s_l} \upharpoonright l$ is a true initial segment of A, in which case nothing further will happen regarding witness w_l. What is gained is that $S(e,l)$ will be defined to be $A_{s_l} \upharpoonright l = A \upharpoonright l$, thus progress was made towards computing A.

Case 3: $\Phi_e(w_1)$ is still undefined after $p_e(t_1)$ steps. In this case, we set $\Theta^{A_{s_1}\upharpoonright 1}(e)\downarrow$ (which should be interpreted as signalling that we have entered Case 3). Observe that is $A_{s_1} \upharpoonright 1$ is a true prefix of A, this implies $\Theta^A(e)\downarrow$ and therefore we would have $\lim h(e,t) = 1$. We distinguish two subcases

(a) The current value $h(e,s)$ is 0. Then we wait for a stage $t > s$ such that either $h(e,t) = 1$ or $A_t \upharpoonright 1 \neq A_s \upharpoonright 1$ (one of the two must happen as we explained above). If the former happens first we move to subcase (b) below. If the latter happens first, we restart the procedure resetting s_1 to the current stage t and keeping the same w_1.

(b) The current value $h(e,s)$ is 1. We then set $R(w_1) = 0$, set $R(n) = 0$ for all $n \leq w_1$ on which R is still undefined and terminate the strategy *for now*. However, if at a later time $t > s$, we see that $h(e,t) = 0$ *and* $A_t \upharpoonright 1 \neq A_{s_1} \upharpoonright 1$, then we resurrect the strategy and start over at the level i where we left off.

We claim that this strategy satisfies the requirement $(\mathcal{R}_e)$. If Case 1 happens for any witness w_l, the requirement is satisfied. Case 3a can only happen finitely many times at a given level since as $A_{s_l} \upharpoonright l$ can only change finitely many times. Case 3b can only happen finitely many times across all levels as each passage through this case causes a flip of $h(e,.)$, and we know $h(e,.)$ converges. Case 2b can also happen only finitely often, because each time we go through this case and do not get to diagonalize, $S(e,.)$ computes a longer initial segment of A, but A is incomputable so $S(e,l)$ would eventually have to be wrong.

Thus we either eventually end up in Case 1 (and immediately succeed) or Case 2a (and immediately succeed) or a terminal Case 3b, i.e., the strategy enters Case 3b and stays there forever. It remains to check this last scenario. Suppose the terminal Case 3b happens for some $A_{s_l} \upharpoonright l$ which is not a prefix of A, this means that $\Psi^A(w_l)$ has not been defined yet and thus, should nothing else happen, we would have $\lim_t h(e,t) = 0$ and would see a change in $A \upharpoonright l$, thus leaving this occurrence of Case 3b, a contradiction. So $A_{s_l} \upharpoonright l$ is indeed a prefix of A and by construction $\Psi^A(w_l)$ returns $0 = R(i)$ in a number of steps t while $\Phi_e(w_l)$ does not return in less than $p_e(t)$ steps, thus the requirement is satisfied. It is now straightforward to satisfy all requirements by ordering them in order of priority, noticing that each strategy only makes finitely many changes to R before achieving its goal and R is total as every time $R(w)$ becomes defined, so do the $R(n)$ for $n \leq w$ that were previously undefined.

□

It is important to note that the above result fails to hold outside of the c.e. setting.

Theorem 3.3. *There exists a low, non-computable set X which is low for speed.*

Proof. See Section 5. □

We next ask whether Theorem 3.3 is tight in terms of the lowness/highness hierarchy. Recall that A is low_2 if $A'' \equiv_T \emptyset''$.

Theorem 3.4. *There is a low_2 c.e. set that is low for speed.*

Proof. This time we must build A to satisfy additionally to $\mathcal{P}_e$ and $\mathcal{L}_e$:

$$\mathcal{N}_e : (\Phi_e^A \text{ is total}) \to (\Theta_e \text{ is total}).$$

where (Θ_e) is a sequence of functionals we build.

The basic idea is that at stages where the length of convergence $\ell(e,s) = \max\{x \mid (\forall y \leq x)\ \Phi_e^A(y)[s]\downarrow\}$ increases we would like to preserve this computation and set $\Theta_e(x)\downarrow$. We think of this having subrequirements $\mathcal{N}_{e,x}$ devoted to preserving $\Phi_e^A(x)[s]$.

There is an apparent conflict between our requirements. Suppose we have three requirements $\mathcal{N}_i$, $\mathcal{L}_j$, $\mathcal{P}_k$, in decreasing order of priority, and $\mathcal{P}_k$ believes that $\mathcal{L}_j$ will have infinitary outcome. At some point, $\mathcal{P}_k$ wants to enumerate some element x into A. As we saw in the previous construction, $\mathcal{L}_j$ will start behaving as if x had already entered, while $\mathcal{P}_k$ waits for confirmation before actually doing so. However, if at some point in the process $\mathcal{N}_i$ imposes a restraint containing x, this will prevent x from entering A and thus destroys $\mathcal{L}_j$'s plan to add x after confirmation, and the functional Ψ built by $\mathcal{L}_j$ might just be wrong. One can check that any other interaction between requirements does not cause any conflict.

The solution to the problematic case is to look closer at the tree of strategies. If the strategies for $\mathcal{L}_j$ and $\mathcal{P}_k$ are working under the assumption that $\mathcal{N}_i$ has finitary outcome (waits forever to preserve $\Phi_e^A \upharpoonright x$ for some x), there is no problem: if it is indeed the case they will not be bothered by any restraint $\mathcal{N}_i$ might impose, and if they are wrong they will not be on the true path anyways. So the problematic case only happens if they are working under the assumption that $\mathcal{N}_i$ has infinitary outcome (arbitrarily long restraints). The idea is, under the assumption that the requirement $\mathcal{N}_i$ has infinitary outcome, to break it down to subrequirements $\mathcal{N}_{i,x}$ devoted to preserve $\Phi_i^A \upharpoonright x$) and place them below on the tree, interleaved with other strategies. So now we are down to the case where $\mathcal{L}_j$ and $\mathcal{P}_k$ are working below an $\mathcal{N}_{i,x}$, under the assumption that $\mathcal{N}_i$ has infinitary outcome. But then there is no problem at all: because of the infinitary assumption, the strategies for $\mathcal{L}_j$ and $\mathcal{P}_k$ can both wait for $\mathcal{N}_{i,x}$ to place its restraint, and then only allow bigger elements to enter A, thus respecting this restraint.

There is no other difficulty to take care of when putting the strategies on a tree, except a small detail: when an $\mathcal{N}$-strategy operate below an $\mathcal{L}$-strategy assuming infinitary outcome, when the $\mathcal{L}$ strategy behaves as if some x has already entered A, then so should the $\mathcal{N}$-strategy. The rest is routine. □

We can also combine the the same ideas (dump construction together with awaiting for certification) with the standard proof that there exists an incomplete c.e. set A of high Turing degree (i.e., $A' \equiv_T \emptyset''$) to get the following.

Theorem 3.5. *There is a high c.e. set A which is low for speed.*

Note that we do not say in the theorem that A is Turing incomplete, because this in fact follows from lowness for speed, as we will see in Proposition 5.7.

4 How big is LFS?

Bayer and Slaman showed that whether LFS is meager or not... depends on the answer to P vs NP question! More precisely, if P = NP, then LFS is co-meager (more precisely, every 2-generic is low for speed), while if P $\neq$ NP, then LFS is meager (more precisely, every recursively generic is not low for speed). They left as an open question whether LFS has measure (Lebesgue) 0 or 1 (by Kolmogorov's 0/1-law, it has to be one or the other). One might expect that, just like the meagerness of LFS depends on the P vs NP question, its measure depends on complexity-theoretic assumptions, such as the 'P vs BPP' question. This is not the case: we show that LFS is – unconditionally – a nullset.

Theorem 4.1. *The set* LFS *has measure* 0. *Indeed, no Schnorr random is low for speed.*

Proof. We first build a computable set R with the following properties: (1) the set R contains at most one string of any given length and (2) for all e, if Φ_e computes R, then $time(\Phi_e, x) > 2^{|x|}$ for almost all x. To do so, we declare at the beginning of the construction all indices e 'active', and for every n in order, do the following. We compute $\Phi_e(x)$ during $2^{|x|}$ of computation for all x of length n and all currently active $e \leq n$. If for some pair (e, x) we see that $\Phi_e(x)$ converges, we take the smallest such pair in the lexicographic order (smallest e first, then smallest x), we diagonalize against Φ_e by setting $R(x) = 1 - \Phi_e(x)$ (thus ensuring $\Phi_e \neq R$), as well as $R(y) = 0$ for all $y \neq x$ of length n, and then declares e inactive from now on. If no such pair (e, x) exists, set $R(y) = 0$ for all y of length n. This finishes the construction of R. It is clear that R is computable (it is even in EXPTIME) and that it contains at most one string of each length. To verify property (2), suppose Φ_e computes R, and observe that for any x such that $time(\Phi_e, x) \leq 2^{|x|}$, the only way Φ_e can escape being diagonlalized against by R is when some Φ_i with $i < e$ is diagonlized against on strings of length n, but this can only happen e times. Thus if Φ_e does indeed compute R, it must do so in time $2^{|x|}$ for almost all x.

Now we need to see how to speed up computations with a Schnorr random oracle. Consider the following procedure Ψ. Given oracle Z and input x, $\Psi^Z(x)$ first splits Z (viewed as a binary sequence) as $Z = \zeta_1\zeta_2\ldots$ with $|\zeta_i| = i$ and $\Psi^Z(x)$ returns 0 if $x = \zeta_{|x|}$ (thus the resulting computation is polynomial in $|x|$), and $\Psi^Z(x) = R(x)$ otherwise, using a fixed procedure to compute R. So there is a polynomial p such that for any Z, $time(\Psi^Z, x) \leq p(|x|)$ for infinitely many x's. Furthermore, we can only have $\Psi^Z(x) \neq R(x)$ if $x = \zeta_{|x|}$ and $\zeta_{|x|}$ happens to be the only string of its length in R. This has probability at most $2^{-|x|}$ ('at most' because R can also have no string of length $|x|$ at all) if Z is chosen at random. This means that, by setting $C_n = \{Z \mid (\exists x)\, |x| = n \wedge \Psi^Z(x) \neq R(x)\}$, we have $\lambda(C_n) \leq 2^{-n}$.

The C_n's are uniformly computable clopen subsets of $\{0,1\}^\omega$ because Ψ is a tt-functional. Thus, a Schnorr random A can only belong to finitely many C_n's (see for example [Bie08, Lemma 1.5.9]), meaning that $\Psi^A(x) = R(x)$ for almost all x. Thus there is a finite variation

$\hat{\Psi}$ of Ψ such that $\hat{\Psi}^A = R$, and $\hat{\Psi}^A(x)$ is computed in polynomial time for infinitely many x while $time(\Phi_i, x) > 2^{|x|}$ for any Φ_i computing R and almost all x. This shows that A is not low for speed. □

On the other hand, we will see in the next section that LFS is large in the set-theoretic sense, namely that it has the size of the continuum.

Finally, there is one last notion of size for subsets of $\{0,1\}^\omega$ that is dear to computability theorists, namely, a set is 'large' if it contains a Turing upper cone and is 'small' if it disjoint from a Turing upper cone. Martin's Turing determinacy theorem tells us that any Borel set which is closed under Turing equivalence must be either large or small on this account. The set LFS is indeed Borel (this is easy to see from the definition), but it is not closed under Turing equivalence, so Martin's theorem does not apply. In the next section, we will use a classical result from complexity theory to show that LFS is in fact disjoint from a Turing upper cone (Theorem 5.8).

5 Lowness for speed and Turing degrees

While lowness for speed is not closed under Turing equivalence, the following question is nonetheless interesting:

Which sets are Turing equivalent to some low for speed X? Which sets compute some non-computable low for speed X?

We denote by **LFS** and **LFS*** the set of Turing degrees that contain a low for speed set and a non-computable low for speed set, respectively. One of the main results of Bayer [Bay12] is that not all degrees are in **LFS**. Indeed, there exists a c.e. degree $\mathbf{a} \notin \mathbf{LFS}$. The main question left open by Bayer regarding **LFS** is whether it is downward closed under $\leq_T$ or closed under join. We give a negative answer to both questions. To show that it is not downward closed, we need the following extension of Theorem 3.2 to degrees.

Theorem 5.1. *For any low c.e. degree $\mathbf{a} > \mathbf{0}$, we have $\mathbf{a} \notin \mathbf{LFS}$.*

Proof. We now suppose that A only is of c.e. degree (as opposed to being c.e.) and is low. Let (A_s) be a Δ^0_2 approximation of A, and let μ be its modulus of convergence, i.e., $\mu(n)$ is the smallest s such that all $\{A_t \mid t \geq s\}$ have the same prefix of length n (which must thus be the prefix of A of length n). Observe that μ is a lower semi-computable function, and let μ_s be its approximation at stage n (setting $\mu_0(n) = 0$ for all n).

The fact that A has c.e. degree is equivalent to the fact that μ is A-computable (see [Soa16]), so let M be a functional such that $M^A(n) = \mu(n)$ for all n. Consider the set $\{X \in \{0,1\}^\omega \mid M^X(n) = k\}$. This is an effectively open set, uniformly in (n,k) and thus can be represented by a c.e. set of strings that are pairwise incomparable for the prefix relation (by this, we mean that an X is in $M^{-1}(n,k)$ if and only if one of these strings is a prefix of X), and this representation is effective and uniform in (n,k). We call $P(n,k)$ the set of strings which both

belong to this set *and* are a prefix of some approximation A_s of A. Observe the following easy facts:

(i) $P(n,k)$ is c.e. uniformly in (n,k)

(ii) $P(n,k)$ is finite for all (n,k).

(iii) If $k \neq k'$, then any two strings $\sigma \in P(n,k)$ and $\tau \in P(n,k')$ are pairwise incomparable for the prefix order.

(iv) A has a prefix in $P(n,\mu(n))$ for all n (and thus no prefix in other $P(n,k)$ for $k \neq \mu(n)$ by the previous fact).

Facts (i), (iii) and (iv) follow directly from the definition. Fact (ii) holds because the sequence (A_s) pointwise converges.

We now adapt the construction of R and Ψ as in the proof of Theorem 3.2. The strategy to win against a pair (Φ_e, p_e) still works by levels, with a verifier S which this time attempts to compute μ (it will eventually fail because if μ were computable, so would be A), and again an auxiliary functional Θ, with $h(.,.)$ a computable approximation of Θ^A, by lowness of A. Also, as before, as soon as $R(n)$ becomes defined, $\Psi^X(n)$ is immediately set to be equal to $R(n)$ for any X on which $\Psi^X(n)$ was still undefined. Our strategy starts with $l = 1$ and S undefined everywhere, and does the following. First, it looks at the current value $k = \mu_s(l)$ and starts enumerating $P(l,k)$. When a new string σ enters $P(l,k)$, we pick a fresh witness $w = w(l,\sigma)$ for this σ, set $\Psi^\sigma(w) = 0$, look at the number t of steps this computation takes, and compute $\Phi_e(w)$ during $p_e(t)$ steps of computation. Once again, there are three cases which will require different actions for this σ:

Case 1: $\Phi_e(w)$ returns 1 in $\leq p_e(t)$ steps. In this case, we set $R(w) = 0$ and $R(n) = 0$ for all n below the current stage on which R is still undefined, and stop the strategy forever.

Case 2: $\Phi_e(w)$ returns 0 in $\leq p_e(t)$ steps. We set $S(e,l) = k$. We then start again the procedure with $l + 1$ and

(a) If at some point we discover that $\mu(l) > k$, we are then free to set $R(w) = 1$, $R(n) = 0$ for all n below the current stage on which R is still undefined, and stop the strategy forever.

(b) $\mu_s(l) = \mu(l)$, in which case $S(e,.)$ has made progress towards computing μ.

Case 3: $\Phi_e(w)$ is still undefined after $p(t)$ steps. In this case, we set $\Theta^\sigma(e)\downarrow$. We have the same two subcases as before:

(a) The current value $h(e,s)$ is 0 (here, this should be interpreted as the fact that $\emptyset'$ currently does not believe that any of the strings that have been enumerated so far in $P(l,k)$ and reached Case 3 is a true initial segment of A). Then we wait for a stage

$t > s$ such that either $h(e,t) = 1$ or to see a new σ enter $P(l,k)$, or to see an increase of μ so that $\mu(l) > k$. If $h(e,t) = 1$ happens first, we move to subcase (b) below. If a new σ enters $P(l,k)$, we set $R(w) = 0$ and $R(n) = 0$ for all and run the strategy on the new σ. Finally, if $\mu(l)$ increases above k, then we know that we have not seen any prefix of A yet at this level, so we restart the procedure at the same level with the new value of k.

(b) The current value $h(e,s)$ is 1. We then set $R(w) = 0$, set $R(n) = 0$ for all n below the current stage on which R is still undefined and terminate the strategy *for now*. We resurrect the strategy at this level if we see either a new σ entering $P(l,k)$ or an increase in $\mu(l)$.

On top of the action for a single σ that enters $P(l,k)$, if at any point of time we see a τ that enters $P(l,k)$, picks a witness w_τ and lands in Case 1 or Case 2, then we immediately stop the action of the other σ's that we have seen so far in $P(l,k)$, and for all of those, copy the action of τ on witness w_τ. This is in fact only important in Case 2, the reason being that we do not want the action of a 'bad' σ (i.e., not a true prefix of A) to interfere with the strategy on a good σ that enters Case 2, as the bad σ would otherwise (by going through Case 3 and defining $R(n)$ for the n needed by the good σ).

The rest of the verification is pretty much the same as before, with the additional point that we can only go through Case 3 finitely often at a given level because h can only flip finitely many times *and* $P(l,k)$ is a finite set, and go through Case 2b only finitely often otherwise $S(e,.)$ would be a computation of μ.

□

Corollary 5.2. LFS *is not downward closed under* $\leq_T$*, even within c.e. degrees.*

Proof. Let $\mathbf{a} > \mathbf{0}$ be a c.e. degree in **LFS** whose existence was explained in Section 3. By Sacks's splitting theorem [Sac63], there is a low c.e. degree $\mathbf{0} < \mathbf{b} < \mathbf{a}$. By Theorem 5.1, $\mathbf{b} \notin$ **LFS**. □

The next theorem will show that while not every c.e. degree contains a low for speed member, every non-zero c.e. degree $\mathbf{a}$ *bounds* a degree $\mathbf{b} \in$ **LFS**. Recall Bayer's result that whether 2-generics are low for speed or not depends on the 'P vs NP' question. When it comes to the *degree* of generics, we have that every 1-generic is Turing-equivalent to a set that is low for speed, independently of complexity-theoretic assumptions.

Theorem 5.3. *Every 1-generic degree* $\mathbf{g}$ *belongs to* **LFS**.

Proof. We get this result by refining the proof of Theorem 2.1. In that proof, we built an X low for speed by finite extension, and ensuring that X was a subset of $S = \{0^{2^n} \mid n \in \mathbb{N}\}$. For $G \subseteq \mathbb{N}$, let $S_G = \{0^{2^n} \mid n \in G\}$. We claim that when G is 1-generic, $S_G = \{0^{2^n} \mid n \in G\}$ is low for speed (and clearly $S_G \equiv_T G$). In the proof of Theorem 2.1, if we let $\mathcal{U}_{\rceil,\rangle}$ be the effectively open sets of those Z such that for some n, $\Phi_e^{Sz}(n)$ and $R_i(n)$ both converge to different values, we know that G, being 1-generic, is either in $\mathcal{U}_{e,i}$ (hence satisfying the

requirement $\mathcal{R}_{e,i}$ as per case (a)), or in the interior of the complement of $\mathcal{U}_{e,i}$, which precisely corresponds to case (b), hence the requirement is also satisfied in this case. □

We can derive a number of useful corollaries from this theorem. First of all, we see that LFS has the size of the continuum since $G \mapsto S_G$ is one-to-one, and there are continuum many 1-generic G. We also get an immediate proof of Theorem 3.3 that asserts the existence of a set of low degree that is low for speed.

Proof of Theorem 3.3. Take a Δ^0_2 1-generic G; the corresponding set S_G is low for speed and is low as it is both Δ^0_2 and of GL_1 (Indeed a result of Jockusch [Joc80] states that every 1-generic degree G is GL_1, that is, $G' \equiv_T G \oplus \emptyset'$; when $G \leq_T \emptyset'$, this is equivalent to $G' \equiv_T \emptyset'$). □

A similar idea allows us to show that **LFS** contains a non-trivial interval in the Turing degrees.

Corollary 5.4. *There is a degree* $\mathbf{a} > \mathbf{0}$ *such that every* $\mathbf{0} \leq \mathbf{b} \leq \mathbf{a}$ *is in* **LFS**.

Proof. By a result of Haught [Hau86], if $\mathbf{a}$ is a Δ^0_2 1-generic degree, every $\mathbf{b} > \mathbf{0}$ below $\mathbf{a}$ is of 1-generic degree. Then the result follows immediately from Theorem 5.3. □

Another interesting corollary is that every non-computable c.e. set bounds a non-computable low for speed set. Likewise almost every set, in the measure-theoretic sense, *bounds* a non-computable low for speed set.

Corollary 5.5. *Every non-zero c.e. degree bounds a member* $\mathbf{LFS}^*$*, every* 2*-random degree bounds a member of* $\mathbf{LFS}^*$.

Proof. This is simply because every non-zero c.e. degree and every 2-random degree bounds a 1-generic degree [Kur81, Kau91]. □

Theorem 5.6. *There are* $\mathbf{a}, \mathbf{b} \in \mathbf{LFS}$ *such that* $\mathbf{a} \vee \mathbf{b} \notin \mathbf{LFS}$

Proof. Let G_0 be 2-generic, i.e., 1-generic relative to $\emptyset'$. Consider $G_1 = G_0 \Delta \emptyset'$ where Δ is the symmetric difference. It is easy to check that G_1 is also 2-generic. Thus S_{G_0} and S_{G_1} (defined as in the proof of Theorem 5.3) are both low for speed but $S_{G_0} \oplus S_{G_1} \geq_T G_0 \oplus G_1 \geq_T G_0 \Delta G_1 = \emptyset'$, so by the previous theorem, $deg(S_{G_0} \oplus S_{G_1}) \notin \mathbf{LFS}$. □

After generic degrees, let us move to random degrees. We have seen in Theorem 4.1 that LFS is a nullset, and in fact no Schnorr degree is low for speed. Interestingly, this result does not extend to Turing degrees: by a result of Nies et al. [NST05], every high degree has a Schnorr random member and we have seen that there is a low for speed of high degree. This leaves open the possibility that almost all X are *Turing-equivalent* to a low for speed set. This would be similar to the category situation where – under the reasonable assumption $\mathsf{P} \neq \mathsf{NP}$ – the set LFS is meager (as proven by Bayer and Slaman) but the set of A's whose *degree* is in **LFS** is co-meager (Theorem 5.3). This is not the case however: if we increase the algorithmic randomness level from Schnorr randomness to Martin-Löf randomness, then the distinction disappears.

Proposition 5.7. If **a** is, or simply bounds, a Martin-Löf random degree then **a** $\notin$ **LFS** (equivalently: if $A \in \{0,1\}^\omega$ computes a Martin-Löf random, A is not low for speed).

Note that this shows in particular that $\{A \mid deg(A) \in \mathbf{LFS}\}$ has measure 0, and also that any $A \geq_T \emptyset'$ is not low for speed, as Chaitin's Ω number is Martin-Löf random and Turing equivalent to $\emptyset'$.

Instead of proving Proposition 5.7 directly, we will prove the following stronger theorem.

Theorem 5.8. *If* **a** *is a DNC degree, then* **a** $\notin$ **LFS** *(equivalently: if $A \in \{0,1\}^\omega$ computes a DNC function, A is not low for speed).*

Proposition 5.7 follows from this theorem because DNC degrees are closed upwards, and by a result of Kučera [Kuč85], every Martin-Löf random degree is a DNC degree.

Proof. The proof of this theorem relies on the proof of a classical computational complexity theorem, namely Blum's speed-up theorem [Blu71] (see also [Koz06, Theorem 32.2]), which asserts that for every sufficiently fast growing computable function f, there exists a computable set R which admits no fastest algorithm in that for every i such that $\Phi_i = R$, there is a j such that $\Phi_j = R$ and $f(time(\Phi_j, x)) \leq time(\Phi_i, x)$ for almost every x.

Let us first discuss how to prove Blum's speed-up theorem. As it happens, we have already seen a proof with similar features, namely the proof of Theorem 4.1 (this is no coincidence, as it is the proof Blum's speed-up theorem that inspired this other proof). We build R by diagonalization against all Φ_i, where for all x in order we try to find an active $i \leq |x|$ such that $\Phi_i(x)$ converges in less than $f^{|x|-i}(|x|)$ steps (here the exponent is understood as composition: f^0 is the identity, and $f^{n+1} = f \circ f^n$) and if such an i is found, we diagonalize against Φ_i by setting $R(x) = 1 - \Phi_i(x)$ for the smallest such i, and declare i inactive from that point on. If no such i is found, set $R(x) = 0$. Obviously R is computable, and the same argument as in the proof of Theorem 4.1 shows that for any Φ_i computing R, one must have $time(\Phi_i, x) \geq f^{|x|-i}(|x|)$ for almost all x. Now, suppose Φ_e is a functional that computes R. We need to show that there is another functional which computes R much faster than Φ_e. Fix a large k and assume we are given as 'advice' the finite list σ_k of indices $i < k$ such that Φ_i eventually gets diagonalized against (and therefore i becomes inactive) in the construction of R. Now, we can compute R via the following procedure. In a first phase, simply follow the construction of R as described above, until we reach a point where all $i \in \sigma_k$ have become inactive. At this point, we know (only because we know σ_k!) that none of the $\{\Phi_i \mid i < k\}$ are relevant for the construction of R on future x. Thus, we enter a second phase where to compute each $R(x)$, we only need to simulate, for $k \leq j \leq |x|$, $\Phi_j(x)$ during $f^{|x|-j}(|x|)$ steps of computation. By dovetailing, this can be done in $poly(|x| \cdot f^{|x|-k}(|x|))$ (the polynomial being independent of k) which, if f is fast growing enough and k large enough compared to e, is $< f\big(f^{|x|-e}(|x|)\big)$, which in turn is $< f(time(\Phi_e, x))$ for almost all x (note that such a k can be computed uniformly given e), and this finishes the proof of Blum's speed-up theorem.

Looking more closely, the core of the argument is that for a given x and a given i, if we somehow knew that Φ_i is not diagonalized against *exactly* at that point x, then we can save the computation of $\Phi_i(x)[f^{|x|-i}(|x|)]$ in the computation of $R(x)$. We already see why DNC-ness naturally comes into play. Let $\theta : \mathbb{N} \to \{0,1\}^*$ be the partial computable function such

that $\theta(i)$ is the one string x on which Φ_i is diagonalized against during the construction of R, and $\theta(i)\uparrow$ if there is no such x. Having a DNC function as oracle allows us to find, for any given i, a $y \neq \theta(i)$ should $\theta(i)$ be defined, and thus one can speed up a bit the computation of $R(y)$ for this y. In fact we can do much better: by a result of Jockusch [Joc89], having access to a DNC function allows us to uniformly avoid a finite number of values of a given partial recursive function. Here this means that with our DNC oracle, we can, uniformly in k, find a $y \notin \theta(i)$ for any $i < k$ on which θ is defined, or equivalently, a y such that none of the Φ_i with $i < k$ gets diagonalized at y. Applying Blum's argument, one can indeed use this information to truly speed up the computation of $R(y)$. We are not quite done yet however: our argument shows that once we have computed y from k we can speed-up the computation of $R(y)$, but the computation of y itself might take a long time and offset the time we save on the computation of $R(y)$.

To overcome this problem, we need to refine the above idea. Let $\langle .,.,. \rangle$ be the canonical 'tripling' function: $\langle a, b, c\rangle = \langle a, \langle b, c\rangle\rangle$ and for $1 \leq i \leq 3$, let π_3^i be the i-th projection ($\pi_3^i(\langle x_1, x_2, x_3\rangle) = x_i$, note that these function are polynomial-time computable). Since A has DNC degree, again using Jockusch's result, A computes, via a fixed functional Ξ, a function F such that $F(k) \neq \pi_3^2(\theta(i))$ for any $i < k$ on which θ is defined (here and in the rest of the proof we identify strings and integers). Said otherwise, for *any* pair (a, c) of integers, none of the functionals $\{\Phi_i \mid i < k\}$ gets diagonalized against at $y = \langle a, F(k), c\rangle$.

Now let Ψ the functional which computes R as follows using oracle A. On input x, it first computes the projections of x, i.e., finds k, l, m such that $x = \langle k, l, m\rangle$. Then, it tries to compute $F(k)$ via Ξ and using oracle A during m steps of computation. If it fails to do so, it then computes $R(x)$ using a fixed procedure (with no access to the oracle) and returns this value. If it does succeed to compute $F(k)$, it then checks whether $l = F(k)$. If not, $\Psi^A(x)$ again returns $R(x)$ using a fixed procedure to perform the computation, without access to the oracle. Finally, and this is the interesting case, if $F(k)$ is computed in $\leq m$ steps of computation, and $l = F(k)$, we know by definition of F that none of the $\{\Phi_i \mid i < k\}$ gets diagonalized against at x. Thus Ψ^A can compute $R(x)$ by only simulating, like in the proof of Blum's theorem $\Phi_j(x)$ during $f^{|x|-j}(|x|)$ steps of computation for $k \leq j \leq |x|$. In that case, the total computation time is still $poly(|x| \cdot f^{|x|-k}(|x|))$ because the m steps of computation needed to get $F(k)$ are simply $poly(|x|)$. For a given k, taking m sufficiently large will give enough time for the computation of $F(k)$ by A, and make $x = \langle k, F(k), m\rangle$ large enough to ensure that the computation time of $\Psi^A(x)$ is $< f(time(\Phi_e, x))$, as long as f is sufficiently fast growing. This shows that A is not low for speed. □

As an immediate corollary, we get the result mentioned in the pervious section that **LFS** is disjoint from a cone in the Turing degrees, again because DNC degrees are closed upwards. Another interesting consequence is that the analogue of the low basis theorem for Π_1^0 class does not extend to lowness for speed: it is not true that every non-empty Π_1^0 class contains a member of low for speed degree. For example, one can take a Π_1^0 class of Martin-Löf randoms: all its elements have DNC degree by Kučera's theorem, and thus do not have low for speed degree.

At this point, we know that the set of X's which *compute* a member of LFS^* is very

large: it has measure 1 and is co-meager, it contains every c.e. set, etc. We might even start thinking that *every* non-computable X computes a member of LFS^*. This is not the case however, as shown by the following theorem (which contrasts Corollary 5.4).

Theorem 5.9. *There is a degree* $\mathbf{a} > \mathbf{0}$ *such that no* $\mathbf{0} < \mathbf{b} \leq \mathbf{a}$ *is in* $\mathbf{LFS}^*$. *Indeed,* $\mathbf{a}$ *can be chosen to be a minimal Turing degree.*

Note that another way to obtain a degree $\mathbf{a} > \mathbf{0}$ such that no $\mathbf{0} < \mathbf{b} \leq \mathbf{a}$ is in $\mathbf{LFS}^*$ is by taking $\mathbf{a}$ to be a hyperimmune-free degree containing a Martin-Löf random member, which ensures that hat every $\mathbf{0} < \mathbf{b} \leq \mathbf{a}$ also computes a 1-random (see [DH10, Corollary 8.6.2]), and apply Proposition 5.7. However, the existence of a minimal low for speed degree is interesting in its own right.

The classical construction of a minimal degree is done by forcing over total computable function trees (here we follow the terminology of [Soa16]). A function tree is a partial function $T : \{0,1\}^* \to \{0,1\}^*$ such that if either $T(\sigma 0)$ or $T(\sigma 1)$ is defined, then all of $T(\sigma)$, $T(\sigma 0)$ and $T(\sigma 1)$ are, and $T(\sigma 0)$ and $T(\sigma 1)$ are strict extensions of $T(\sigma)$ such that $T(\sigma 0) \perp T(\sigma 1)$ (we say that $T(\sigma 0)$ and $T(\sigma 1)$ split $T(\sigma)$). We say that σ is a node of T if $\sigma \in rng(T)$. A tree S is a sub-f-tree of T, which we denote by $S \preccurlyeq T$ when every node of S is a node of T. An infinite binary sequence Z is a path on an f-tree T if infinitely many prefixes of Z are nodes of T. The set of paths of T is denoted by $[T]$. An f-tree naturally extends to a functional from $\{0,1\}^\omega$ to $\{0,1\}^* \cup \{0,1\}^\omega$ by setting $T(X) = \bigcup_{\sigma \preccurlyeq X} T(\sigma)$ When an f-tree T is total, this extension is an homeomorphism from $\{0,1\}^\omega$ to $\{0,1\}^\omega$, and if T is furthermore computable, its inverse T^{-1} is also computable. Given a functional Φ_e, we say that T is e-consistent if for any two nodes σ and τ on T and any n, if $\Phi_e^\sigma(n)$ and $\Phi_e^\tau(n)$ are both defined, then they are equal; in this case, for any path X of T, Φ_e^X is either partial or computable. We say that T is e-splitting if T is total and for any σ, $\Phi_e^{T(\sigma 0)}$ and $\Phi_e^{T(\sigma 1)}$ are incomparable; in this case, the restriction of Φ_e to $[T]$ is total, one-to-one, and its inverse is computable.

The key lemma in the construction of a minimal degree states that for any total computable f-tree T and every e, there is a total computable sub-f-tree S of T which is either e-consistent or e-splitting. Then an $X \in \{0,1\}^\omega$ of minimal degree is obtained by taking a sufficiently generic filter G over the set of total computable f-trees ordered by $\preccurlyeq$, and take the intersection of their sets of paths (the non-computability of X can be further ensured by remarking that for any σ, the set of computable f-trees whose nodes are all incomparable with σ is a dense set for the order $\preccurlyeq$, thus one can choose X to avoid any fixed subset of $\{0,1\}^\omega$, such as its computable elements).

We are going to prove Theorem 5.9 by showing that taking a sufficiently generic filter for the order $\preccurlyeq$ ensures lowness for speed as well. For this, we will make use of the following lemma.

Lemma 5.10. *Let* T *be a total computable f-tree. There exists a total computable f-subtree* $S \preccurlyeq T$ *none of whose paths is low for speed.*

Proof. The functional $T^{-1} : [T] \to \{0,1\}^\omega$ is total on its domain, which is a Π^0_1 class, and thus is a tt-reduction by effective compactness. Let f be a computable time bound for

the running time of T^{-1}, that is, for any $Y \in [T]$, whether $x \in T^{-1}(Y)$ can be decided in time $f(|x|)$ with access to oracle Y. Now, let L be a computable set that cannot be computed in time $2^{f(n+1)}$. Let S be the sub-f-tree of T defined by $S(\sigma) = T(\sigma \oplus L)$ (where $\sigma \oplus L = \sigma(0)L(0)\dots\sigma(k-1)L(k-1)$ when $k = |\sigma|$). The paths of S are exactly the sets of the form $T(X \oplus L)$ for some X. Each of them computes L in time $f(n+1)$ by definition of f, which is exponentially faster than any procedure computing L without oracle by our assumption on L. □

Proof of Theorem 5.9. Let T be a total computable f-tree and Φ_e a functional. As we explained earlier, the usual construction of a minimal degree shows that there is $S \preccurlyeq T$ which is either e-consistent or e-splitting. In the case S is e-consistent, we are satisfied (this guarantees Φ_e^A to be either partial or computable). If it is e-splitting, we further refine S as follows. Since S is e-splitting, we consider the total computable f-tree S' corresponding to the image of S by Φ_e: $S'(\sigma) = \Phi_e^{S(\sigma)}$ (this is indeed an f-tree precisely because S is e-splitting). By the previous lemma, there is a total computable $S'' \preccurlyeq S'$ none of whose paths is low for speed. Now the pullback $T' = \Phi_e^{-1}(S'')$ is a total computable f-subtree of T, which forces Φ_e^A to not be low for speed.

Thus we can force for all e that Φ_e^A is partial or not low for speed, and force A to be of minimal degree and be non-computable as usual. □

We do not know whether or not *all* sets of minimal degree are in fact non-low for speed.

Remark 5.11. Another way to prove Theorem 5.9 is to use Kumabe ans Lewis's theorem that there exists a minimal degree of DNC degree [KL09], and apply Theorem 5.8, but our proof is more informative, as it also shows that lowness for speed is a generic notion for forcing with total with computable f-trees.

Acknowledgements. This paper grew from interactions between complexity theory and classical computability theory originating from the 2012 Dagstuhl Seminar "Computability, Complexity and Randomness" (Seminar 12012). Bienvenu acknowledges support of ANR-15-CE40-0016-01 RaCAF grant, Downey thanks the Marsden Fund of New Zealand, and the LIRMM (University of Montpellier) where this research was undertaken.

References

[ABK06] Eric Allender, Harry Buhrman, and Michal Koucký. What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic*, 138:2–19, 2006. 1

[AFG13] Eric Allender, Luke Friedman, and William I. Gasarch. Limits on the computational power of random strings. *Information and Computation*, 222:80–92, 2013. 1

[Bay12] Robertson Bayer. *Lowness For Computational Speed*. PhD thesis, University of California Berkeley, 2012. 2, 5, 6, 11

[BGS75] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, 1975. 1

[Bie08] Laurent Bienvenu. *Game-theoretic characterizations of randomness: unpredictability and stochasticity*. PhD thesis, Université de Provence, 2008. https://tel.archives-ouvertes.fr/tel-00332425v2. 10

[Blu71] Manuel Blum. On effective procedures for speeding up algorithms. *Journal of the ACM*, 18(290-305), 1971. 15

[CDE+14] Mingzhong Cai, Rodney Downey, Rachel Epstein, Steffen Lempp, and Joseph Miller. Random strings and tt-degrees of Turing complete c.e. sets. *Logical Methods in Computer Science*, 10(3), 2014. 2

[DH10] Rodney Downey and Denis Hirschfeldt. *Algorithmic randomness and complexity*. Theory and Applications of Computability. Springer, 2010. 3, 17

[Hau86] Christine A. Haught. The degrees below a 1-generic degree $< 0'$. *Journal of Symbolic Logic*, 51, 1986. 14

[Joc80] Carl Jockusch. Degrees of generic sets. In Frank Drake and Stanley S. Wainer, editors, *Recursion theory: its generalizations and applications*, number 45 in London Mathematical Society Lecture Note Series, pages 110–139. Cambridge Unversity Press, 1980. 14

[Joc89] Carl Jockusch. *Degrees of functions with no fixed points*, pages 191–201. North-Holland, Amsterdam, 1989. 16

[Kau91] Steven M. Kautz. *Degrees of random sets*. PhD thesis, Cornell University, 1991. 14

[KL09] Masahiro Kumabe and Andrew E. M. Lewis. A fixed-point-free minimal degree. *Journal of the London Mathematical Society*, 80(3):785–797, 2009. 18

[Koz06] Dexter Kozen. *Theory of Computation*. Springer, New York, 2006. 15

[Kuč85] Antonin Kučera. Measure, Π^0_1 classes, and complete extensions of PA. *Lecture Notes in Mathematics*, 1141:245–259, 1985. 15

[Kur81] Stuart Kurtz. *Randomness and genericity in the degrees of unsolvability*. PhD dissertation, University of Illinois at Urbana, 1981. 14

[Nie09] André Nies. *Computability and randomness*. Oxford Logic Guides. Oxford University Press, 2009. 7

[NST05] André Nies, Frank Stephan, and Sebastiaan Terwijn. Randomness, relativization and Turing degrees. *Journal of Symbolic Logic*, 70:515–535, 2005. 14

[Sac63] Gerald Sacks. On the degrees less than 0'. *Annals of Mathematics*, 77:211–231, 1963. 13

[Soa16] Robert Soare. *Turing Computability: Theory and Applications*. Theory and Applications of Computability. Springer, 2016. 11, 17