Feasible Iteration of Feasible Learning Functionals*

John Case[1], Timo Kötzing[1], and Todd Paddock[2]

[1] Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716-2586,USA
case@cis.udel.edu, koetzing@cis.udel.edu
[2] Majestic Research,
1270 Avenue of the Americas, Suite 1900, New York, NY 10020
todd@majesticresearch.com

Abstract. For learning functions in the limit, an algorithmic learner obtains successively more data about a function and calculates trials each resulting in the output of a corresponding program, where, hopefully, these programs eventually converge to a correct program for the function. The authors desired to provide a feasible version of this learning in the limit — a version where each trial was conducted feasibly *and* there was some feasible limit on the number of trials allowed. Employed were *basic feasible functionals* which query an input function as to its values and which provide each trial. An additional tally argument 0^i was provided to the functionals for their execution of the i-th trial. In this way more time resource was available for each successive trial. The mechanism employed to feasibly limit the number of trials was to feasibly count them down from some feasible notation for a constructive ordinal. Since all processes were feasible, their termination was feasibly detectable, and, so, it was possible to wait for the trials to terminate and suppress all the output programs but the last. Hence, although there is still an iteration of trials, the learning was a special case of what has long been known as total **Fin**-learning, i.e., learning in the limit, where, on each function, the learner always outputs exactly one conjectured program. Our general main results provide for strict learning hierarchies where the trial count down involves all and only notations for infinite limit ordinals. For our hierarchies featuring finitely many limit ordinal jumps, we have upper and lower total run time bounds of our feasible **Fin**-learners in terms of finite stacks of exponentials. We provide, though, an example of how to regain feasibility by a suitable parameterized complexity analysis.

* Case and Paddock were supported in part by NSF grant number NSF CCR-0208616. We are also grateful to anonymous referees for many helpful suggestions. One such referee provided hints about the truth and truth and proof, respectively, of what became, then, Lemmas 6 and 7; hence, these results are joint work with that referee. This same referee suggested, for the future, team learning as an approach to studying some probabilistic variants of our learning criteria.

M. Hutter, R.A. Servedio, and E. Takimoto (Eds.): ALT 2007, LNAI 4754, pp. 34–48, 2007.

1 Introduction and Motivation

One-shot (algorithmic) learners, on input data about a function, output at most a single (hopefully correct) conjectured program [JORS99]. *Feasible* (deterministic) one-shot function learning can be modeled by the polytime multi-tape Oracle Turing machines (OTMs) as used in [IKR01] (see also [KC96, Meh76]). We call the corresponding functionals *basic feasible functionals*.

In the context of learning in the limit, i.e., learning with a succession of one-shots, where only the final shots are hoped to be correct, we are interested, then, in how one *might* define *feasible* for *limiting*-computable functionals. We next discuss the concepts we require for such a definition.

Intuitively *ordinals* [Sie65] are representations of well-orderings. 0 represents the empty ordering, 1 represents the ordering of 0 by itself, 2 the ordering $0 < 1$, 3 the ordering $0 < 1 < 2$, ... The ordinal ω represents the standard ordering of all of $\mathbb{N}$. $\omega + 1$ represents the ordering of $\mathbb{N}$ consisting of the positive integers in standard order *followed by* 0. The *successor ordinals* are those of the form $\alpha + 1$ which have a single element laid out after a copy of another ordinal α. $\omega + \omega$ can be thought of as two copies of ω laid end to end — much bigger than ω. $\omega \cdot 3$ represents three copies of ω laid end to end. By contrast, $3 \cdot \omega$ represents ω copies of 3 — which is just ω. We see, for ordinals, $+, \cdot$ are not commutative. $\omega \cdot \omega$ is ω copies of ω laid out end to end. We can iterate this and define exponentiation for ordinals. *Limit ordinals* are those, like ω, $\omega + \omega$, $\omega \cdot \omega$, and ω^ω, which are not 0 and are not successor ordinals. All of them are infinite. Importantly, the *constructive ordinals* are just those that have a program (called a *notation*) in some system which specifies how to build them (lay them out end to end, so to speak). Everyone knows how to use the natural numbers for counting, including for counting *down*. Freivalds and Smith [FS93], as well as [ACJS04], employed in learning theory *notations for constructive ordinals* as devices for algorithmic counting down. Herein we need to count down iterations of applications of feasible learning functionals. For example, for us, as we will see more formally in Section 4 below, algorithmic counting down iterations from any notation u for $\omega + 1$ is roughly equivalent to counting down one iteration and, then, deciding dynamically how many further but finite number of iterations will be allowed. Herein, though, we want the counting down process itself to be feasible. Hence, in Section 3, we introduce *feasibly related feasible systems of ordinal notations*, where, basically, the definition of a system of ordinal notations (as in [Rog67]) is restricted to those systems where all necessary operations and decision processes are *feasibly* computable. In Section 3, by Theorem 8, for each constructive ordinal α, we have such a system containing a notation for α and all its predecessors.

In Section 4, we present our proposed definition (Definition 11) for feasible iteration of feasible learning functionals. Then we present our general main results providing for strict learning hierarchies at all and only notations for (infinite) limit ordinals. First, Theorem 14 provides the learning hierarchy collapse between feasible notations for α and for $\alpha + 1$. Importantly, Theorem 17, provides a *strict* learning hierarchy between feasible notations for successive feasible *limit* ordinals.

In Section 5, our main results involve upper and lower runtime bounds for learning hierarchies featuring feasibly counting down from feasible notations for the successive initial limit ordinals $\omega \cdot n$, $n = 1, 2, 3, \ldots$. These runtime bounds are expressed in terms of exponential polynomials $\mathbf{q}$. In Theorem 20, for learning featuring feasible counting down from feasible notations for $\omega \cdot n$, the stacking of exponentials in the *upper* bound $\mathbf{q}$ is no more than n. Theorem 21 says there are classes learnable featuring feasible counting down from feasible notations for $\omega \cdot n$, where the stacking of exponentials in the *lower* bound $\mathbf{q}$ is at least n. In Section 5, we provide, though, an *example* of how to regain feasibility by a suitable parameterized complexity analysis [DF98].

Due to space constraints some portions of proofs are omitted. Complete proofs are in [CPK07].

2 Mathematical Preliminaries

$\mathbb{N}$ denotes the set of natural numbers, {0,1,2,...}. We do not distinguish between natural numbers and their *dyadic* representation.[1] $\mathrm{card}(D)$ denotes the cardinality of a set D.

The symbols $\subseteq, \subset, \supseteq, \supset$ respectively denote the subset, proper subset, superset and proper superset relation between sets.

We sometimes denote a function f of $n > 0$ arguments $x_1, \ldots, x_n$ in lambda notation (as in Lisp) as $\lambda x_1, \ldots, x_n .f(x)$. For example, with $c \in \mathbb{N}$, $\lambda x .c$ is the constantly c function of one argument. From now on, by convention, f and g with or without decoration range over functions $\mathbb{N} \to \mathbb{N}$, x, y with or without decorations range over $\mathbb{N}$, 0^i, 0^j range over $\{0\}^*$.

We use 'string' and 'finite sequence' synonymously, and, for each sequence s, we will denote the first element of that sequence by $s(0)$, (or, equivalently, with s_0,) the second with $s(1)$ (or s_1) and so on.

Similarly we will consider infinite sequences s as functions with domain $\mathbb{N}$ (or $\mathbb{N} \cup \{-1\}$, as the case may be), and denote them at position a in the domain by $s(a)$ or s_a.

For each string w, define $\mathrm{len}(w)$ to be the length of the string. As we identify each natural number x with its dyadic representation, $\mathrm{len}(x)$ denotes the length of the dyadic representation of x. For all strings w, we define $|w|$ to be $\max\{1, \mathrm{len}(w)\}$.[2]

Following [LV97], we define for all $x \in \mathbb{N}$: $\overline{x} = 1^{\mathrm{len}(x)}0x$. Using this notation we can define a function $\langle \cdot \rangle$ coding tuples of natural numbers of arbitrary size ($k \geq 0$) into $\mathbb{N}$ such that $\langle v_1, \ldots, v_k \rangle := \overline{v_1} \ldots \overline{v_k}$.

For example the tuple $(4, 7, 10)_{decimal} = (01, 000, 011)_{dyadic}$ would be coded as 11 0 01 111 0 000 111 0 011 (but without the spaces added for ease of parsing).

[1] The *dyadic* representation of a natural number $x :=$ the x-th finite string over $\{0, 1\}$ *in lexicographical order*, where the counting of strings starts with zero [RC94]. Hence, unlike with binary representation, lead zeros matter.

[2] ε denotes the empty string. This convention about $|\varepsilon| = 1$ helps with runtime considerations.

Obviously $\langle\cdot\rangle$ is 1-1. The time to encode tuples, that is, to compute $\lambda v_1, \ldots, v_k \,.\, \langle v_1, \ldots, v_k\rangle$ is $\in \mathcal{O}(\lambda v_1, \ldots, v_k \,.\, \sum_{i=1}^{k} |v_i|)$. Therefore the size of the codeword is also linear in the size of the components: $\lambda v_1, \ldots, v_k \,.\, |\langle v_1, \ldots, v_k\rangle| \in \mathcal{O}(\lambda v_1, \ldots, v_k \,.\, \sum_{i=1}^{k} |v_i|)$. Decoding is linear in the length of the codeword: For all $k, i \leq k$, we have that $\lambda\langle v_1, \ldots, v_k\rangle \,.\, v_i$ is computable in linear time, so is $\lambda\langle v_1, \ldots, v_k\rangle \,.\, k$.

A function ψ is *partial computable* iff there is a Turing machine computing ψ.

φ^{TM} is the fixed programming system from [RC94, Chapter 3] for the partial computable functions. This system is based on deterministic, multi-tape Turing machines (TMs). In this system the TM-programs are *efficiently* given numerical names or codes.[3] Φ^{TM} denotes the TM step counting complexity measure also from [RC94, Chapter 3] and associated with φ^{TM}. In the present paper, we employ a number of complexity bound results from [RC94, Chapters 3 & 4] regarding $(\varphi^{\mathrm{TM}}, \Phi^{\mathrm{TM}})$. These results will be clearly referenced as we use them. For simplicity of notation, hereafter we write (φ, Φ) for $(\varphi^{\mathrm{TM}}, \Phi^{\mathrm{TM}})$. φ_p denotes the partial computable function computed by the TM-program with code number p in the φ-system, and Φ_p denotes the partial computable *runtime* function of the TM-program with code number p in the φ-system.

Whenever we consider tuples of natural numbers as input to TMs, it is understood that the general coding function $\langle\cdot\rangle$ is used to code the tuples into appropriate TM-input. We say that a function from k-tuples of natural numbers into $\mathbb{N}$ is *feasibly computable* iff, for some p, it is computed by TM p in polytime in the lengths of its inputs.[4]

The next definitions provide the formal details re the polytime constraint on basic feasible functionals.

The *length* of $f : \mathbb{N} \to \mathbb{N}$ is the function $|f| : \mathbb{N} \to \mathbb{N}$ such that $|f| = \lambda n. \max(\{|f(x)| \mid |x| \leq n\})$.

A *second-order polynomial* over type-1 variables $g_0, \ldots, g_m$ and type-0 variables $y_0, \ldots, y_n$ (in this paper simply referred to as a polynomial) is an expression of one of the following five forms.

$$a; \quad y_i; \quad \mathbf{q}_1 + \mathbf{q}_2; \quad \mathbf{q}_1 \cdot \mathbf{q}_2; \quad g_j(\mathbf{q}_1)$$

where $a \in \mathbb{N}$, $i \leq n$, $j \leq m$, and $\mathbf{q}_1$ and $\mathbf{q}_2$ are second-order polynomials over $\overrightarrow{g}$ and $\overrightarrow{y}$.

A *subpolynomial* of $\mathbf{q}$ is, recursively, any polynomial which is used in the construction of $\mathbf{q}$, or which is a subpolynomial of a polynomial that is used in the construction of $\mathbf{q}$.

We understand each such polynomial $\mathbf{q}$ as a symbolic object and for functions $f_0, \ldots, f_m : \mathbb{N} \to \mathbb{N}$, $x_0, \ldots, x_n \in \mathbb{N}$ we write $\mathbf{q}(f_0, \ldots, f_m, x_0, \ldots, x_n)$ as the obvious evaluation of $\mathbf{q}$ to an element in $\mathbb{N}$.

[3] This numerical coding guarantees that many simple operations involving the coding run in linear time. This is by contrast with historically more typical codings featuring prime powers and corresponding at least exponential costs to do simple things.

[4] We are mostly not considering herein interesting polytime probablistic or quantum computing variants of the deterministic feasibility case.

For two functions $h_1, h_2 : \mathbb{N} \to \mathbb{N}$ we write $h_1 \leq h_2 :\Leftrightarrow \forall n \in \mathbb{N} : h_1(x) \leq h_2(x)$ and we say that h_2 majorizes h_1. It is easy to see from the definition of a second-order polynomial $\mathbf{q}$ that $\lambda f_0, \ldots, f_m, x_0, \ldots, x_n . \mathbf{q}(f_0, \ldots, f_m, x_0, \ldots, x_n)$ is non-decreasing in each argument, given that all function-arguments are non-decreasing and order on functions is as defined just above.

An *Oracle Turing Machine* (OTM) is a multi-tape Turing Machine that also has a query tape and a reply tape. To query an oracle f, an OTM writes the dyadic representation of an $x \in \mathbb{N}$ on the query tape and enters its query state. The query tape is then erased, and the dyadic representation of $f(x)$ appears on the reply tape. This model is extended to the case of multiple oracles in the obvious way. The (time) cost model is the same as for non-oracle Turing machines, *except* for the additional cost of a query to the oracle. This is handled with the length-cost model, where the cost of a query is $|f(x)|$, where $|f(x)|$ is the length of the string on the reply tape.

Suppose $k \geq 1$ and $l \geq 0$. Then $F : (\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^l \to \mathbb{N}$ is a *basic feasible functional* if and only if there is an OTM $\mathbf{M}$ and a second-order polynomial $\mathbf{q}$, such that, for each input $(f_1, \ldots, f_k, x_1, \ldots, x_l)$,

(a) $\mathbf{M}$ outputs $F(f_1, \ldots, f_k, x_1, \ldots, x_l)$, and
(b) $\mathbf{M}$ runs within $\mathbf{q}(|f_1|, \ldots, |f_k|, |x_1|, \ldots, |x_l|)$ time steps (we will then say that $\mathbf{q}$ majorizes the runtime of F).

Any unexplained computability-theoretic notions are from [Rog67].

3 Feasible Systems of Ordinal Notations

In this section we begin with some definitions regarding systems of ordinal notations. The first definition is quite technically useful in our proofs in Section 4 below.

Definition 1. For a system of ordinal notations S as, for example, in [Rog67], a pair (l_S, n_S) is a *decompose pair for* S iff l_S and n_S are functions $\mathbb{N} \to \mathbb{N}$ and for all notations $u \in S$ for an ordinals α, $l_S(u)$ denotes a notation for the biggest (limit ordinal or 0) $\lambda \leq \alpha$, and $n_S(u)$ is such that $\alpha = \lambda + n_S(u)$.

Definition 2. (Feasible System of Ordinal Notations) For an ordinal α, a *feasible system of ordinal notations* for all and only the ordinals $< \alpha$ is a tuple $(S, \nu_S, \lim_S, +_S, \cdot_S, l_S, n_S)$ where $S \subseteq \mathbb{N}$, ν_S maps $\mathbb{N}$ onto the set of all ordinals $< \alpha$, $\lim_S : \mathbb{N} \times \{0\}^* \to \mathbb{N}$, $+_S$ and $\cdot_S$ are ordinal sum and multiplication on notations respectively[5] and (l_S, n_S) is a decompose pair for S.[6] Additionally we require:
The following predicates over $u \in S$ are feasibly decidable.

(a) "u is a notation for 0",

[5] Therefore, each feasible system of ordinal notations will give notation to an additively and multiplicatively closed set of ordinals.
[6] We will sometime ambiguously refer to $(S, \nu_S, \lim_S, +_S, \cdot_S, l_S, n_S)$ as S.

(b) "u is a notation for a successor ordinal" and
(c) "u is a notation for a limit ordinal".

And:

(d) There is a feasibly computable function pred_S such that for all u notations for a successor ordinal $\alpha + 1$, $\text{pred}_S(u)$ is a notation for α.
(e) $\lim_S$ is a feasible function and for all limit-ordinals $\lambda < \alpha$ and notations l for λ we have that $(\nu_S(\lim_S(l, 0^i)))_{i<\omega}$ is a strictly increasing sequence of ordinals with limit λ.

Up to this point in this definition, we have a modification of Rogers' concept of *system of ordinal notations* [Rog67], *where*, when we require feasible computability, Rogers requires only partial computability. Additionally we require

(f) $+_S$ is feasibly computable,
(g) $\cdot_S$ is feasibly computable,
(h) from any natural number n, a notation $\underline{n}_S$ for n is feasibly computable and
(i) l_S, n_S are feasibly computable.

Definition 3. Following Rogers [Rog67], we say that a system of ordinal notations S is *univalent* iff ν_S is 1-1; we define the relation $\leq_S$ on natural numbers such that: $u \leq_S v \Leftrightarrow [u, v \in S \wedge \nu_S(u) \leq \nu_S(v)]$. Also following Rogers, we say a system of ordinal notations S is *computably related* iff $\leq_S$ is computably decidable, and *computably decidable* iff the set of notations S is computably decidable. Analogously, we define a system S to be *feasibly related* iff $\leq_S$ is feasibly decidable, and *feasibly decidable* iff the set S is feasibly decidable.

Remark 4. In Definition 2 above we have that feasible relatedness, together with (f), (h) and (i) implies (a)-(d). Every feasibly related feasible system of ordinal notations S is feasibly decidable, as we have: $u \in S \Leftrightarrow u \leq_S u$. Every feasibly related feasible system of ordinal notations is a computably related system of ordinal notations.[7] For a feasibly related or univalent feasible system of ordinal notations S, it is feasibly decidable whether two notations are notations for the same ordinal.[8]

Lemma 5. Suppose S is a system of ordinal notations in which a notation in S for the successor ordinal is feasibly computable from a given notation in S. Let $\lim_S : \mathbb{N} \times \{0\}^* \to \mathbb{N}$ be a computable function satisfying the analog of (e) where "feasible" is replaced by "partial computable". Then there is a *feasibly* computable function $\lim'_S : \mathbb{N} \times \{0\}^* \to \mathbb{N}$ satisfying (e).

[7] Therefore, all theorems for computably related systems of ordinal notations hold. For example, there cannot be a feasibly related feasible system of ordinal notations for all constructive ordinals (see [Rog67]).

[8] For univalent systems there are of course no two different notations for the same ordinal. For a feasibly related systems of ordinal notations, $u, v \in \mathbb{N}$ are notations in S for the same ordinal iff $[u \leq_S v$ and $v \leq_S u]$.

Proofsketch. Define $\lim'_S$ thus. On input $(u, 0^i)$, run $\lim_S$ on inputs $(u, 0^j)$ for all $j \leq i$, each for up to i steps. If none converges, output $\underline{i}$ — a notation in S for i. Otherwise, for some $j \leq i$, $\lim_S(u, 0^j)$ converges. In this case, for the maximal such j, compute the i-times successor of $\lim_S(u, 0^j)$ and output the result — a notation for $\nu_S(\lim_S(u, 0^j)) + i$. Importantly, thanks to [RC94, Corollary 3.7], the algorithm just provided for $\lim'_S$ *is* feasible. We omit the remaining details of the proof. □

Lemma 6. Suppose S is a computably related system of ordinal notation for all and only the ordinals $< \alpha$ for some ordinal α. Then there is a *feasibly* related system S' of ordinal notations for all and only the ordinals $< \alpha$.

Proof. Define S' thus. Let e be the numerical name for a program deciding $\leq_S$. Define $t : \mathbb{N} \to \mathbb{N}, u \mapsto \max(\{\Phi_e(i, j) \mid i, j \leq u\})$. Let S' be the system of notations where for all β given a notation u in S, we have that $\langle 0^{t(u)}, 0^u \rangle$ is a notation for β. Obviously, $\forall m, n, u, v \in \mathbb{N} : \langle 0^m, 0^u \rangle \leq_{S'} \langle 0^n, 0^v \rangle \Leftrightarrow [\varphi_e(u, v) = 1$ in $\leq \max\{n, m\}$ steps and $m = t(u)$ and $n = t(v)]$. It follows from [RC94, Lemma 3.2(f) and Corollary 3.7] that $\lambda 0^m, 0^u . t(u) = m$ is feasibly decidable. Therefore, on the resulting notations we have that order is feasibly decidable. □

Lemma 7. Suppose S is a feasibly related system of ordinal notations giving a notation to all and only the ordinals $< \alpha$ for some ordinal α. Then there is a feasibly related system of ordinal notations S' fulfilling (a)-(f) and (h)-(i) as in Definition 2, giving a notation at least to all ordinals $< \alpha$. In fact, S' gives a notation to all and only the ordinals $< \omega^\alpha$. If S is univalent, so is S'.

Proofsketch. Assume without loss of generality that 0 is the only notation for 0 in S. Let $\langle\rangle$ be a notation in S' for 0. By the Cantor Normal Form theorem, each ordinal γ, $0 < \gamma < \omega^\alpha$ has exactly one representation such that $\gamma = \sum_{i=k}^{0} \omega^{\delta_i} \times n_i$, where $\alpha > \delta_k > \ldots > \delta_0 \geq 0$ and $n_k, \ldots, n_0 \in \mathbb{N} \setminus \{0\}$ (see [Sie65, Theorem 2, Chapter XIV.19, page 323]). Define a system S' by the following assignment of notations. For each γ with $0 < \gamma < \omega^\alpha$, the representation as above and $d_k, \ldots, d_0$ notations in S for $\delta_k, \ldots, \delta_0$, respectively, let

$$\langle d_k, n_k, \ldots, d_0, n_0 \rangle \text{ be a notation in } S' \text{ for } \gamma.$$

From here we omit most remaining details. To show (e) for S': We apply Lemma 5. □

Theorem 8. Suppose S is a feasibly related system of ordinal notations giving a notation to all and only the ordinals $< \alpha$. Then there is a feasibly related *feasible* system of ordinal notations S' giving a notation at least to all ordinals $< \alpha$. In fact, S' gives a notation to all and only the ordinals $< \omega^{\omega^\alpha}$. If S is univalent, so is S'.

Proofsketch. Apply the construction of the proof of Lemma 7 *twice* to S. The resulting system will also allow for feasible multiplication. □

Corollary 9. *Let α be a constructive ordinal. Then there is a univalent, feasibly related feasible system of ordinal notations giving a notation to α.*

Proof. By [Rog67, Theorem 11.XIX], there is a univalent, computably related system of ordinal notations giving a notation to α. The result follows now from first applying Lemma 6 and then Theorem 8. □

Assumption 10. For the rest of this paper, fix an arbitrary univalent feasibly related feasible system of ordinal notations S. We furthermore make the following assumption.

$$\forall u \in S, n \in \mathbb{N} : |n| \leq |\underline{n}| \leq |u +_S \underline{n}| \,. \tag{1}$$

This reasonable assumption holds for all systems constructed in the proof of Corollary 9. (1) above also shows that for all $u \in S$ we have $|n_S(u)| \leq |l_S(u) + n_S(u)| = |u|$; therefore, we get

$$\forall u \in S : n_S(u) \leq u \,. \tag{2}$$

4 Hierarchies at Limit Ordinal Jumps

Next is our proposed definition of feasible iteration of feasible learning functionals.

Definition 11. Suppose $u \in S$. A set of functions $\mathcal{S}$ is $\mathrm{Itr}_u\mathrm{BffFin}$-*identifiable* (we write $\mathcal{S} \in \mathrm{Itr}_u\mathrm{BffFin}$) iff there exist basic feasible functionals $H : (\mathbb{N} \to \mathbb{N}) \times \{0\}^* \to \mathbb{N}$ and $F : (\mathbb{N} \to \mathbb{N}) \times \{0\}^* \to \mathbb{N}$ such that for all $f \in \mathcal{S}$ there exists $k \in \mathbb{N}$ such that

(a) $F(f, 0^t) <_S u$ for all $t < k$,[9]
(b) $F(f, 0^{t+1}) <_S F(f, 0^t)$ for all $t < k$,
(c) $F(f, 0^k) = \underline{0}$ and
(d) $\varphi_{H(f,0^k)} = f$.

Lemma 12. Without loss of generality the count down function F in Definition 11 can be chosen such that for all computable functions f there is a $k \in \mathbb{N}$ such that (a) and (b) in Definition 11 hold, as well as $\forall t \geq k : F(f, 0^t) = \underline{0}$.

Proof. Let q be a polynomial upper-bounding the runtime of F. Then there is F' such that F' on input $(f, 0^t)$ computes for all $w \leq t$ $F(f, 0^w)$ (taking time in $\mathcal{O}(\sum_{w=0}^{t} q(|f|, w)) \subseteq \mathcal{O}(t \cdot q(|f|, t))$). If we have $F(f, 0^0) <_S u$ and for all $w < t$ $F(f, 0^{w+1}) <_S F(f, 0^w)$ (t comparisons decidable in polytime), then output $F(f, 0^t)$, otherwise output $\underline{0}$. □

[9] Earlier papers using count down functions, such as for example [ACJS04], usually use instead, at this point in the definition, $\leq_S u$. This earlier way of starting count downs can be recovered in the version presented herein by using $<_S u +_S \underline{1}$. Our present version has additional expressibility for u being a notation for a limit ordinal, which is not available in a version starting with $\leq_S u$. However, it is a theorem in this paper (Theorem 14 below) that, for our way herein of starting count downs, no resultant extra learning power class exists.

Assumption 13. From now on, all witnesses for a set to be in $\mathrm{Itr}_u\mathrm{BffFin}$ to have these additional properties as stated in Lemma 12. Witnesses explicitly constructed might not have this property.

Note that for all $u \in S$, as it is also the case for many other identification criteria [JORS99], $\mathrm{Itr}_u\mathrm{BffFin}$ is closed under taking subsets.

The first theorem shows that there is no difference in learning power for an ordinal and its successor:

Theorem 14. Suppose $u \in S$. Then $\mathrm{Itr}_u\mathrm{BffFin} = \mathrm{Itr}_{(u+_S\underline{1})}\mathrm{BffFin}$.

Proof. Trivial for u a notation for 0. Otherwise, let $\mathcal{S} \in \mathrm{Itr}_{(u+_S\underline{1})}\mathrm{BffFin}$ as witnessed by (H, F). We have for all $f \in \mathcal{S}$: $F(f, 0^0) <_S u \vee F(f, 0^1) <_S u$. Let $P := \lambda f .\mu i < 2 .(F(f, 0^i) <_S u)$; $H' := \lambda f, 0^i .H(f, 0^{i+P(f)})$; $F' := \lambda f, 0^i .F(f, 0^{i+P(f)})$. So (H', F') witnesses $\mathcal{S} \in \mathrm{Itr}_u\mathrm{BffFin}$. □

Recall that in the mathematical preliminaries it has been mentioned that all polynomials as defined in this paper fulfill several monotonicity constraints. Furthermore, we can find a single polynomial upper bounding the runtime all functions of a given, finite set of BFFs (for example by adding all polynomials for each single BFF up). The next definition gives a desirable property of polynomials. The following remark will imply that we can – for all uses of polynomials in this paper – suppose without loss of generality that our polynomials have this property.

Definition 15. A polynomial $\mathbf{q}$ is called *request-bounding* iff for all polynomials $\mathbf{q}'$ such that $g(\mathbf{q}')$ is a subpolynomial of $\mathbf{q}$ we have that $\mathbf{q}$ majorizes $\mathbf{q}'$.

Remark 16. For all polynomials $\mathbf{q}$ there is a request-bounding polynomial majorizing $\mathbf{q}$.

Proof. Let $\mathbf{q}$ be a polynomial, $\mathbf{q}'$ a polynomial such that $g(\mathbf{q}')$ is a subpolynomial of $\mathbf{q}$ and $\mathbf{q}$ does not majorize $\mathbf{q}'$. We have that $\overline{\mathbf{q}} := \mathbf{q} + \mathbf{q}'$ majorizes $\mathbf{q}$ and $\mathbf{q}'$ such that $\{\mathbf{r} \mid g(\mathbf{r}) \text{ is a subpolynomial of } \overline{\mathbf{q}}\} = \{\mathbf{r} \mid g(\mathbf{r}) \text{ is a subpolynomial of } \overline{\mathbf{q}}\}$. Iterating this construction of a bigger polynomial will finally yield a polynomial with the desired properties. □

Next is the main theorem of this section. It provides a strict learning power hierarchy.

Theorem 17. Suppose $u <_S v \in S$ represent non-successor ordinals. Then $\mathrm{Itr}_u\mathrm{BffFin} \subset \mathrm{Itr}_v\mathrm{BffFin}$.

Proofsketch. The inclusion is trivial, so that the separation remains to be shown. This is done by constructing a suitable set of total computable functions which belongs to the right set, but not to the left. Let $\mathcal{S}^*$ be the set of all functions f such that: There is a sequence r of notations for non-successor ordinals, strictly decreasing (with respect to $<_S$), where $r(0) < v$. There is a strictly increasing sequence s of natural numbers of length $\mathrm{len}(r) + 1$ such that:

(a) $s(0) = 1$,
(b) for all $i < \text{len}(s) - 1 : s(i+1) \in \{2^{f(s(i))}, \ldots, 2^{f(s(i))} + |f(0)| - 1\}$,[10]
(c) for all $i < \text{len}(s) - 1 : f(s(i)) \in S$,
(d) for all $i < \text{len}(s) - 1 : l_S(f(s(i))) = r(i)$,
(e) $r(\text{len}(r) - 1) = \underline{0}$ and
(f) for all $i \in \mathbb{N}$: $i \in (\text{range}(s) \cup \{0\}) \Leftrightarrow f(i) \neq 0$.

That our $\mathcal{S}^*$ witnesses the separation of Theorem 17 just above provides the reason its use in Proposition 22 (in Section 5 below) is interesting.

Define $\mathcal{S} := \{f \in \mathcal{S}^* \mid |f(0)| = 1\}$. To save space in this proof, we will actually show instead that $\mathcal{S}$, a proper subset of $\mathcal{S}^*$, witnesses the separation. Of course, the negative part of the separation trivially applies to supersets.

Claim: $\mathcal{S} \in \text{Itr}_v\text{BffFin}$

Proof (of claim). Let e be such that for all finite functions σ (treated as strings of size $\text{len}(\sigma)$, coded onto the tape by $\langle\rangle$), and for all $x \in \mathbb{N}$,

$$\varphi_e(\sigma, x) = \begin{cases} \sigma(i) \text{ , if } i < \text{len}(\sigma) \text{ such that } x = 2^i; \\ 0 \quad \text{ , otherwise.} \end{cases}$$

Runtime in $\mathcal{O}$

F `on` $(f, 0^t)$`:`
  `query` f `for` $\sigma := \lambda i \leq t.f(2^i)$ $|f|(t) \cdot t$
  `determine biggest index` $i \leq t$ `such that` $f(2^i) > 0$ $|f|(t) \cdot t$
  `if such an index does not exist, output` $\underline{0}$
  `if two such indices exist, let` $l < i$ `be the biggest two`
  `if` $l_S(f(2^i)) = \underline{0}$ `redefine` $i := l$
  `output` $l_S(f(2^i)) +_S \underline{(f(2^i) - t)}$

The next functional makes use of a linear time instance of an s-m-n-function.[11]

H `on` $(f, 0^t)$`:`
  `query` f `for` $\sigma := \lambda i \leq t.f(2^i)$ $|f|(t) \cdot t$
  `output s-m-n`(e, σ) linear time s-m-n

(H, F) shows the claim. □ (OF CLAIM)

Claim: $\mathcal{S} \notin \text{Itr}_v\text{BffFin}$.

Proof (of claim). Suppose by way of contradiction otherwise, as witnessed by (H, F). Let $\mathbf{q}(g, x)$ be a polynomial with the following properties. $\mathbf{q}$ strictly majorizes the runtime of F and H; $\mathbf{q}$ is request-bounding; and, for technical reasons, for all $c, n \in \mathbb{N}$, $\mathbf{q}(\lambda x.c, n) \geq c$.[12]

Define now two functions $a, b : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $\forall x, y : a(x, y) = \mathbf{q}(\lambda z.|x|, y), b(x, y) = \mathbf{q}(\lambda z.|x|, a(x, y))$. For all $x, y \in \mathbb{N}$, $a(x, y)$ is, then, an

[10] This entails, as s is required to be strictly increasing, that $f(s(i)) > 0$ for all $i < \text{len}(s) - 1$.

[11] Linear time s-m-n is a function s running in linear time such that $\forall e, x, y : \varphi_{s(e,x)}(y) = \varphi_e(x, y)$; see [RC94, Theorem 4.7(b)].

[12] For each polynomial $\mathbf{q}'$, $\mathbf{q}' + g(0)$ is an example polynomial fulfilling this property.

upper runtime-bound for the computation of F on second argument 0^y, if the first argument is never requested at anything yielding something bigger then x. For all $x, y \in \mathbb{N}$, $b(x, y)$ is an upper runtime-bound for the computation of F on second argument $0^{a(x,y)}$ with the same restriction on the first argument.

Let w be a notation for ω. We define, by multiply recursive calls, for $i \geq 0$, infinite sequences s, v and a computable function f as follows.

$$\begin{aligned}
v_{-1} &= 0\\
s(0) &= 1\\
f(s(0)) = v_0 &= u +_S \underline{x},\\
&\quad \text{where } x \text{ minimal so that } v_0 > b(v_0, v_{-1}) \text{ and}\\
&\quad |v_0| > 2^{b(v_{-1},0)} + 1\\
s(i+1) &= \begin{cases} 2^{v_i}, \text{ if } v_i \neq 0\\ \max\{s(j) \mid j \leq i\} + 1, \text{ otherwise} \end{cases}\\
f(s(i+1)) = v_{i+1} &= \begin{cases} 0, \text{ if there is } j < i-1 \text{ such that } l_S(v_j) = w\\ \underline{1}, \text{ if } l_S(v_{i-1}) = w\\ \underline{((v_i+1)+x)}, \text{ if } l_S(v_i) = w\\ (F(f, 0^{a(v_i,v_{i-1})}) +_S w) +_S \underline{((v_i+1)+x)}, \text{ otherwise} \end{cases}\\
&\quad \text{where } x \text{ minimal so that } v_{i+1} > b(v_{i+1}, v_i) \text{ and}\\
&\quad |v_{i+1}| > 2^{b(v_i,v_{i-1})} + a(v_i, v_{i-1}) + 1
\end{aligned}$$

Define f on all so far undefined values as 0.
Notes on the construction:

- To show that the condition on v_0 is possible: On the one hand, by Assumption 10 in Section 3, $\lambda x\,.\,u +_S \underline{x}$ grows at least as fast as $\lambda x\,.\,x$, and $\lambda x\,.\,x$ grows *exponentially* in the *length* of its argument, and, on the other hand, $\lambda x\,.\,b(u +_S \underline{x}, 0)$ grows only *polynomially* in the *length* of its argument. Similar reasoning holds for the condition on v_{i+1}.
- As part (f) of the proof of the next subclaim we will show that, in the computation of $F(f, 0^{a(v_i,v_{i-1})})$, the value of f at $s(i+1)$ will never be requested. This avoids $f(s(i+1))$ calling itself.

Let m be maximal such that $v_m \neq 0$. Clearly, from above, for all i such that $0 \leq i \leq m$, $v_i \neq 0$. Abbreviate for all $i \leq m$: $a_i := a(v_i, v_{i-1})$ and $b_i := b(v_i, v_{i-1})$.

Subclaim: f, s, v are well defined.
Proof (of subclaim). We proof this by induction on $i < m$. We have the following induction invariants.

(a) if $0 \leq i$, then $b_i < v_i$,
(b) if $0 \leq i$, then there are $> v_i$ steps required to query f at $s(i+1)$,
(c) if $0 \leq i$, then f is circle-free defined up through $s(i+1) - 1$,
(d) if $0 \leq i$, then $\forall t \leq b_i : |f|(t) < v_i$,

(e) if $0 \leq i$, then b_i is an upper runtime bound for F on $(f, 0^{a_i})$,
(f) if $0 \leq i$, then $f(s(i+1))$ is not requestable in the computation of F on $(f, 0^{a_i})$.
(g) v_{i+1} is well defined

These invariants hold obviously for $i = -1$. Let now $i < m$ be such that $0 \leq i$ and these induction invariants hold for for all $k < i$ such that $-1 \geq k$. We show now that the invariants hold for i.

To show (a): By construction we have $v_i > b(v_i, v_{i-1}) = b_i$.

To show (b): We have $|s(i+1)| = |2^{v_i}| = v_i$. Therefore, v_i steps are required to write $s(i+1)$ on the query tape; at least an additional step is used to complete this query process.

To show (c): Follows from for all $k < i$, v_{k+1} well defined (that is, $\forall k \leq i$, v_k is well defined).

To show (d): Let $t \leq b_i$. We have:

$$|f|(t) \leq |f|(b_i) = \max_{|x| \leq b_i} |f(x)| \leq \max_{|x| \leq v_i} |f(x)| \leq \max_{x < 2^{v_i}} |f(x)| = |v_i| < v_i \ .$$

To show (e): By induction on the polynomials $\mathbf{q}'$ such that $g(\mathbf{q}')$ is a subpolynomial of $\mathbf{q}$. Using invariant (d) and $\mathbf{q}$ being request-bounding it is easy to show that all such polynomials $\mathbf{q}'$ evaluate on $(|f|, a_i)$ to something $\leq b_i$. Then we can conclude that $\mathbf{q}(|f|, a_i) \leq b_i$.

To show (f): (b), (e) and (a) show (f).

To show (g): From (f). □ (OF SUBCLAIM)

It is now easy to verify that $f \in \mathcal{S}$, and, then, a simple adversary argument (as, for example, in [DZ01]) shows that either f or f modified at $s(m)$ (so that the modified version is still in $\mathcal{S}$) is not properly identified by (H, F).

□ (OF CLAIM) □ (OF THEOREM)

5 Upper and Lower Bounds on Runtime

For this section assume S gives a notation to at least all ordinals $< \omega^2$ (the existence of such an S is guaranteed by Corollary 9). In this section we will characterize the hierarchy of finitely many limit ordinal jumps in terms of explicit total runtime bounds. The next definition introduces polynomials with exponential terms and the exponential nesting depth of such.

Definition 18 (Polynomials with Exponentials). We define recursively in parallel the set $Q[g]$ of symbolic polynomials with exponentials, as well as the exponential nesting depth of $\mathbf{q} \in Q[g]$, $rk(\mathbf{q})$, (read: rank of $\mathbf{q}$):

for all $a \in \mathbb{N}$: $a \in Q[g]$,	$rk(a) = 0$,
for all $\mathbf{q}_1, \mathbf{q}_2 \in Q[g]$: $(\mathbf{q}_1 + \mathbf{q}_2) \in Q[g]$,	$rk((\mathbf{q}_1 + \mathbf{q}_2)) = \max(rk(\mathbf{q}_1), rk(\mathbf{q}_2))$,
$\mathbf{q}_1, \mathbf{q}_2 \in Q[g]$: $\mathbf{q}_1 \cdot \mathbf{q}_2 \in Q[g]$,	$rk(\mathbf{q}_1 \cdot \mathbf{q}_2) = \max(rk(\mathbf{q}_1), rk(\mathbf{q}_2))$,
for all $\mathbf{q} \in Q[g]$: $g(\mathbf{q}) \in Q[g]$,	$rk(g(\mathbf{q})) = rk(\mathbf{q})$,
for all $\mathbf{q} \in Q[g]$: $2\hat{}(\mathbf{q}) \in Q[g]$,	$rk(2\hat{}(\mathbf{q})) = rk(\mathbf{q}) + 1$.

The following definition will enable us to study the runtime of functionals beyond the feasible.

Definition 19. Suppose $k \geq 1$ and $l \geq 0$. Then $F : (\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^l \to \mathbb{N}$ is a *computable functional* if and only if there is an OTM $\mathbf{M}$ such that, for each input $(f_1, \ldots, f_k, x_1, \ldots, x_l)$, $\mathbf{M}$ outputs $F(f_1, \ldots, f_k, x_1, \ldots, x_l)$.

The two theorems below are our main results of the present section.

Theorem 20 (Learning Time – Upper Bound). Let $n > 0$. Let $\mathcal{S} \in \mathrm{Itr}_{w \cdot \underline{n}}\mathrm{BffFin}$. Then there is a computable functional h such that

- $\forall f \in \mathcal{S} : \varphi_{h(f)} = f$
- the runtime of h is bounded above by some $\mathbf{q} \in Q[g]$ such that $rk(\mathbf{q}) \leq n$.

Proof. Let $\mathcal{S} \in \mathrm{Itr}_{w \cdot \underline{n}}\mathrm{BffFin}$ as witnessed by (H, F). Define t, h such that for all f computable functions: $t(f) = \mu x \,.\, F(f, 0^x) = \underline{0}$, $h(f) = H(f, 0^{t(f)})$. Obviously h fulfills the first requirement.
Let $\mathbf{p}$ be a polynomial upper-bounding the runtime of H and F. Let $\mathbf{q}_0 = 0$. Define recursively for all $i < n$: $\mathbf{q}_{i+1} = 2^{\mathbf{p}(g, \mathbf{q}_i)} + \mathbf{q}_i$. It is clear that $rk(\mathbf{q}_i) = i$.

We have now, for all $f \in \mathcal{S}$ and $i < n$, that the calculation of F on $(f, 0^{\mathbf{q}_i(|f|)})$ is bounded above by $\mathbf{p}(|f|, \mathbf{q}_i(|f|))$; therefore, $F(f, 0^{\mathbf{q}_i(|f|)}) < 2^{\mathbf{p}(|f|, \mathbf{q}_i(|f|))}$, and, hence, by (2) from the very end of Section 3:

$$n_S(F(f, 0^{\mathbf{q}_i(|f|)})) <_S 2^{\mathbf{p}(|f|, \mathbf{q}_i(|f|))} = \mathbf{q}_{i+1}(|f|) - \mathbf{q}_i(|f|) \ . \tag{3}$$

(3) is to be read as follows. After $\mathbf{q}_i(|f|)$ iterations of F, F will output a notation for an ordinal with natural-number part less then $\mathbf{q}_{i+1}(|f|) - \mathbf{q}_i(|f|)$ – therefore, after no more then $\mathbf{q}_{i+1}(|f|) - \mathbf{q}_i(|f|)$ additional iterations (after a total of $\mathbf{q}_{i+1}(|f|)$ iterations), there has to be a limit ordinal jump in the output of F. A simple induction shows now that we have, for all $f \in \mathcal{S}$ and $i < n$, $F(f, 0^{\mathbf{q}_i(|f|)}) <_S w \cdot \underline{n - i}$, and, therefore, $F(f, 0^{\mathbf{q}_n(|f|)}) = \underline{0}$ (this makes use of assumption 13).

Let $f \in \mathcal{S}$. The above shows that we have $t(f) \leq \mathbf{q}_n(|f|)$; therefore, an algorithm for computing t could run F on all arguments $(f, 0^i)$ in increasing order for all $i \leq \mathbf{q}_n(|f|)$, checking each output for equaling $\underline{0}$. This takes a total time $\leq \sum_{i=0}^{\mathbf{q}_n(|f|)} \mathbf{p}(|f|, i) \leq (\mathbf{q}_n(|f|)+1) \cdot \mathbf{p}(|f|, \mathbf{q}_n(|f|))$. Therefore, h can be computed in $\leq (\mathbf{q}_n(|f|)+1) \cdot \mathbf{p}(|f|, \mathbf{q}_n(|f|)) + \mathbf{p}(|f|, \mathbf{q}_n(|f|)) = (\mathbf{q}_n(|f|)+2) \cdot \mathbf{p}(|f|, \mathbf{q}_n(|f|))$ steps, where $rk((\mathbf{q}_n + 2) \cdot \mathbf{p}(g, \mathbf{q}_n)) = rk(\mathbf{q}_n) = n$. □

Theorem 21 (Learning Time – Lower Bound). Let $n > 0$. Let $\mathcal{S}$ be as in the proof of the limit ordinal jump hierarchy (Theorem 17) for the special case of $\mathrm{Itr}_{w \cdot \underline{(n-1)}}\mathrm{BffFin} \subset \mathrm{Itr}_{w \cdot \underline{n}}\mathrm{BffFin}$. Define $\mathcal{S}' := \{f \in \mathcal{S} \mid \mathrm{card}(\{x > 0 \mid f(x) > 0\}) = n + 1\}$. Let h be a computable functional such that $\forall f \in \mathcal{S}' : \varphi_{h(f)} = f$ and fix an OTM M computing h. Define $\mathbf{q}_0 := g(1)$, define for all $i < n$, $\mathbf{q}_{i+1} := g(2\hat{}(\mathbf{q}_i))$. Then $rk(\mathbf{q}_n) = n$ and we have, for all $f \in \mathcal{S}'$, $\mathbf{q}_n(|f|)$ is a *lower bound* on the runtime of M on argument f.

Proof. For $f \in \mathcal{S}$ we have by induction that $|\max(\{x > 0 \mid f(x) > 0\})| \geq \mathbf{q}_n(|f|)$, which shows by way of a simple adversary argument (as, for example, in [DZ01]) the claim. □

The following proposition illustrates *one* possibility to analyze a set in $\mathrm{Itr}_v\mathrm{BffFin}$ (for any $v \in S$ representing a limit ordinal) in terms of parametrized complexity as in [DF98].

Proposition 22. Let $v \in S$ be a notation for a limit ordinal, let $\mathcal{S}^* \in \mathrm{Itr}_v\mathrm{BffFin}$ be as in the proof of Theorem 17. Define for all $k \in \mathbb{N}$, $\mathcal{S}^*_k := \{f \in \mathcal{S}^* \mid \forall x > 0 : f(x) < k\}$. Then we have

(a) $\bigcup_{k\in\mathbb{N}} \mathcal{S}^*_k = \mathcal{S}^*$,
(b) for all $k \in \mathbb{N}$, $\mathcal{S}^*_k \in \mathrm{Itr}_0\mathrm{BffFin}$ (that is, $\mathcal{S}^*_k$ is one-shot learnable by a basic feasible functional) and
(c) there is a k_0 such that for all $k > k_0$, $\mathcal{S}^*_k$ is infinite.

Proofsketch. (a) is trivially true, as all $f \in \mathcal{S}^*$ are finite variants of the constant 0 function.

(b) Let $k \in \mathbb{N}$. Among the numbers $< k$ there are of course at most k notations for ordinals. Let $f \in \mathcal{S}^*_k$. Let $C := \{x > 0 \mid f(x) \neq 0\} = \{s(0) < s(1) < \ldots < s(m)\}$. We have now, for distinct $x, y \in C$, that $f(x) \neq f(y)$, as they have to be notations for ordinals with different limit parts. Furthermore, as $f \in \mathcal{S}^*$, for all $i < m$, we have $s(i+1) < 2^{f(s(i))} + |f(0)| < 2^k + |f(0)|$. Now we have for all $x \geq 2^k + |f(0)|$, $f(x) = 0$. Checking all other positions $< 2^k + |f(0)|$ and creating an appropriate output with linear time s-m-n (as it was also done in the positive part of the proof of Theorem 17) takes therefore $\mathcal{O}(|f|(0))$ time.

(c) Let $k_0 := \underline{3}$. We omit the remaining detailed verification. □

6 Conclusions and Future Work

In this paper we showed *one* possible approach to putting feasibilty restrictions on learning in the limit learning. However, our strict learning hierarchies are at the price of some infeasibility. Furthermore, our particular scheme of feasibly iterating basic feasible learning functionals requires the count down function to bottom out at $\underline{0}$, so one can tell when the iterations are done (and can and do suppress all the programs output but the last). We were initially surprised that, for a scheme like ours, we get a learning hierarchy result as in our Theorem 17 (in Section 4 above). We are interested in the future investigation of more ways for feasibly iterating feasible learning functionals. *We'd like variant definitions and results where one cannot suppress all the output programs but the last.* It seems this may be difficult if we retain strict determinism. In this interest, then, we would also like to see studied *probabilistic* variants of feasibly iterated feasible learners – this toward producing practical generalizations of Valiant's *PAC* learning [KV94] and Reischuk and Zeugmann's [RZ00] *stochastically finite learning*. These latter involve, probabilistic, one-shot learners. [RZ00], intriguingly for our purposes, compiles the multiple trials of a special case of *deterministic limit learning* into a feasible *probabilistic one-shot* variant.

References

[ACJS04] Ambainis, A., Case, J., Jain, S., Suraj, M.: Parsimony hierarchies for inductive inference. Journal of Symbolic Logic 69, 287–328 (2004)

[CPK07] Case, J., Paddock, T., Kötzing, T.: Feasible iteration of feasible learning functionals (expanded version). Technical report, University of Delaware (2007), At `http://www.cis.udel.edu/~case/papers/FeasibleLearningTR.pdf` and contains complete proofs

[DF98] Downey, R., Fellows, M.: Parameterized Complexity. In: Downey, R., Fellows, M. (eds.) Monographs in Computer Science. Springer, Heidelberg (1998)

[DZ01] Dor, D., Zwick, U.: Median selection requires $(2 + \epsilon)n$ comparisons. SIAM Journal on Discrete Mathematics 14(3), 312–325 (2001)

[FS93] Freivalds, R., Smith, C.: On the role of procrastination in machine learning. Information and Computation 107(2), 237–271 (1993)

[IKR01] Irwin, R., Kapron, B., Royer, J.: On characterizations of the basic feasible functional, Part I. Journal of Functional Programming 11, 117–153 (2001)

[JORS99] Jain, S., Osherson, D., Royer, J., Sharma, A.: Systems that Learn: An Introduction to Learning Theory. 2nd edn. MIT Press, Cambridge (1999)

[KC96] Kapron, B., Cook, S.: A new characterization of type 2 feasibility. SIAM Journal on Computing 25, 117–132 (1996)

[KV94] Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)

[LV97] Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, Heidelberg (1997)

[Meh76] Mehlhorn, K.: Polynomial and abstract subrecursive classes. Journal of Computer and System Sciences 12, 147–178 (1976)

[RC94] Royer, J., Case, J.: Subrecursive Programming Systems: Complexity and Succinctness. Research monograph in Progress in Theoretical Computer Science. Birkhäuser Boston (1994)

[Rog67] Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York, 1967. MIT Press (Reprinted, 1987)

[RZ00] Reischuk, R., Zeugmann, T.: An average-case optimal one-variable pattern language learner. Journal of Computer and System Sciences 60(2), 302–335 (2000), Special Issue for COLT'98

[Sie65] Sierpinski, W.: Cardinal and ordinal numbers. Second revised edn. PWN –Polish Scientific Publishers (1965)