FOUNDATIONS OF ONLINE STRUCTURE THEORY II: THE OPERATOR APPROACH

ROD DOWNEY, ALEXANDER MELNIKOV, AND KENG MENG NG

ABSTRACT. We introduce a framework for online structure theory. Our approach generalises notions arising independently in several areas of computability theory and complexity theory. We suggest a unifying approach using operators where we allow the input to be a countable object of an arbitrary complexity.

CONTENTS

1. INTRODUCTION

1.1. **Our Goal.** Imagine you are tasked with putting objects of differing sizes into bins of a fixed size. Your goal is to minimize the number of bins you need. This is the famous BIN PACKING problem which we know is NP complete (see Karp [30]). But imagine that we change the rules and you are only given the objects one at a time and you must choose which bin to put the object into before being given the

2010 *Mathematics Subject Classification.* Primary 03D45, 03C57. Secondary 03D75, 03D80.
The first two authors were partially supported by Marsden Fund of New Zealand. The third author is partially supported by the grant MOE2015-T2-2-055 and RG131/17.

next object. You are in an *online* situation and this is the ONLINE BIN PACKING problem. The "first fit" method is well-known to give a 2-approximation algorithm for this problem (definitions given in detail below). Alternatively imagine you are a scheduler, and your goal is to schedule requests within a computer for memory allocation amongst users. Again you are in an online situation, but here you might want to change the order of allocation depending on priorities of the requests.

A brief thought on this will reveal that there are potentially hundreds of situations where we are dealing with combinatorial algorithms for tasks where we only have partial evolving information about the the input data, or perhaps the data is so large that we cannot see it in total. This is the reason that there are so many algorithms for online tasks. On the other hand, there seems no general *theory* which we can use as a conceptual basis for the theory of online algorithms, and online structures. Books in this area, such as Albers [3], all tend to be taxonomies of algorithms. Our goal is to give a theoretical basis for the theory of online algorithms and structures.

1.2. **The Punctual Model.** In [5] a project was started aiming at providing a model-theoretical foundation to this theory. In that paper we focussed upon the intuition that online decisions in practice have *lack of delay.* That is, we need to pack the object into some bin immediately, before the next one is presented to us (in the BIN PACKING example). This led to a theory of online structures and algorithms we generally referred to as *punctual* structure theory.

Here we would imagine that our structures are given in a primitive recursive way and our goal is to build some desired function or object accordingly. This resulted in a surprisingly rich theory (e.g. [5, 15, 28, 29]). This theory applies in many situations stemming from computable structure theory particularly those arising from natural decision procedures, since almost all proofs of decidability of "natural" theories give primitive recursive decision procedures.

1.3. **The Uniform Model.** Imagine we need to build a colouring of a graph G which is given online. Thus, in the very simplest case, we would be given the graph $G = \lim_s G_s$, where G_s has s vertices. When the vertex s is introduced, we are also given at the same time precisely which vertices amongst $\{1, \ldots, s-1\}$ has an edge with s (and this cannot change later). Our task is to colour s so that no two vertices which are connected have the same colour, before the opponent presents us with G_{s+1}.

Now if we imagine this as an infinite process and we need to colour the whole of an infinite graph G. We can think of each possible version of G as being a path through an infinite tree of possibilities. Each node σ of length s of the tree will represent some graph G_σ with s vertices, and if $\sigma \prec \sigma'$ then G_σ is the subgraph of $G_{\sigma'}$ induced by vertices $\{1, \ldots, s\}$. (Note that there are only primitively recursively many non-isomorphic graphs with s vertices.)

The critical observation is the following. Although G can be viewed a path on an infinite primitive recursive tree of possibilities, there is no *a priori* reason that we should only consider a primitive recursive graph G. There are continuum many such paths and the online graph colouring problem can be considered for an infinite countable graph of any complexity.

Comparing this situation with general online algorithms, the reader will quickly realise that the key point about online algorithms is one of *continuity* or *uniformity.*

If we have a colouring of G_σ and we add $s+1$, the next G will be one of the possible $(s+1)$-extensions of G_σ. If this is, say, G_τ, with $|\tau| = s+1$, then $\chi_{G_\tau}(s+1)$ (the colour of vertex $s+1$ in G_τ) must extend, or in other words, be compatible with the colouring χ_{G_σ} on G_σ. The reader should then realise that we are in fact dealing with certain special kinds of *uniform computable operators* acting on trees of possibilities. We will also demand that the action works primitive recursively, or perhaps even running in polynomial time. These will all soon be made precise.

What is very interesting is that once we have made all these observations, we realise that this material also have a connection with *computable* and *feasible analysis*, and also with the complexity theory for operators in analysis along the lines of Kawamura and Cook [31], Melhorn [41], Ko and Friedman [20], and others. It turns out also to have connections with reverse mathematics, computational learning theory, and even algorithmic randomness. We will also see that, in this setting, the finiteness of the objects being given is not an essential restriction. In the online case, finite objects are only revealed one bit at a time, and for all intents and purposes, we may as well treat all inputs as arbitrarily large finite structures. We will prove that under the uniform operator framework, working with arbitrarily large finite structures and infinite structures are indeed the same. This allows for example, for a formal approach in which one can study finite combinatorics in reverse mathematics.

2. The main definition

2.1. **Representation spaces.** A class $\mathcal{C}$ of relational structures is called *inductive* if $A \in \mathcal{C}$ implies A has a *filtration* $A = \cup_s A_s$ where each A_n is finite, has universe $\{1, \ldots, n\}$, and for all $n' > n$ the substructure induced by $\{1, \ldots, n\}$ in $A_{n'}$ is A_n. More generally, for a fixed computable function g, we say that $\mathcal{C}$ is *g-inductive* if it has a g-filtration meaning that each A_n has universe $\{1, \ldots, g(n)\}$.

Our language will typically be finite and relational.

Remark 2.1. We need some care if the language has function symbols. For example, we can follow [27] and assume that the indices of the f-images of $\{x_1, \ldots, x_n\}$ are within $\{x_1, \ldots, x_{g(n)}\}$, where g is a primitive recursive function. Here g can be taken equal to the number of distinct terms of $\{x_1, \ldots, x_n\}$ in the (free) term algebra of this language.

The intuition is that, although f may be played by the Universe, this is we who give notations to the elements. It seems unreasonable to set $f(x_1, x_3, x_5)$ equal to $x_{10^{10}}$ if we can name $f(x_1, x_3, x_5)$ with x_7.

Here also we refer the reader to Section 5, where we study situations where we are in a very large space and can only have local knowledge; which is a natural online situation. There we would expect the values of functions to be relatively local in the online situation.

We refer to the substructure of A based on $\{1, \ldots, n\}$ the substructure of *height* $h(n) = n$. In the example discussed above, the height n structures are the graphs with n vertices. Another example is considered by Khoussainov [32] with a height function in his work on random infinite structures. By abusing notation, we will let $\mathcal{C}^{<\omega}$ denote the class of finite substructures of $\mathcal{C}$. There is also the natural induced topology. For example, in the graph case this would be compact and have the totally disconnected topology with basic open sets being the extensions of graphs of height n.

2.2. **Representations.** To have a computability theory, we need a notion of effective presentation. We borrow ideas from computable analysis. When a computability theorist is considering, for example, effective procedures in polynomial rings, we would consider questions like "Find a root of $p(x) = a_0 + a_1x + \cdots + a_nx^{n-1}$". Now, *implicitly* what we are really talking about is a coded version of this question where we are really considering $p(x)$ coded as a string in, for example, $2^{<\omega}$, or, classically, using Gödel numbering. As with our upcoming definition of online computation, sometimes we need some care as to how we represent the data. For example in modelling actual computations, or in computational complexity theory, reprentations in say, unary or binary, can have dramatic effects.

There is a tradition in classical computable analysis to make codings explicit. For example a partial function f taking a polynomial $p(X)$ to it's least rational root q, if any, would be considered as follows. A *representation* of the set $\mathbb{Q}[X]$ of rational polynomials is a partial (in this case computable) $\delta_0 : 2^{<\omega} \to \mathbb{Q}[X]$, and similarly we'd have $\delta_1 : 2^{<\omega} \to \mathbb{Q}$, and, for instance, we'd say that a string $\sigma \in 2^{<\omega}$ is a δ_0-*name* for a polynomial $p(X)$ if $\delta_0(\sigma) = p(X)$. Then the the function f would be considers as being *realized*, i.e. represented, by any function F such that if $f(x) = q$, then for all δ_0-names σ of $p(X)$ $F(\sigma) = \tau$, and τ is a δ_1-name of q.

Generally in "Type I" computability where the objects of study are effective functions on natural numbers, this is a bit of an overkill; hence codings tend to be supressed. Our online work concerns functions whose inputs are filtrations and hence infinite. This is the essence of "Type II" computability. The main idea will be to represent the paths as sequences of codes and have the algorithm act on the codes.

A *representation* of an inductive class $\mathcal{C}$ of structures is a surjective function[1] $\delta : \omega^{<\omega} \to \mathcal{C}^{<\omega}$, which acts computably in the sense that $\delta(\sigma) = C_n$ for $|\sigma| = n$ and $h(C_n) = n$, and if $\sigma \preceq \tau$ then $\delta(\sigma)$ is an induced substructure of $F(\tau)$. We can also extend this in the natural way to g-filtrations. Thus such an δ induces a map $\overline{\delta}$ from $\omega^\omega \to \mathcal{C}$, namely $\lim\{\delta\sigma \mid \sigma \prec x\}$. We will call $x \in \omega^\omega$ a *name* for $C \in \mathcal{C}$ if $\overline{\delta}(x) = C$. Note that it is possible that each structure C to have a number of different names[2].

For the time being, we will regard $\overline{\delta}$ as being injective. When it is possible, we will replace $\omega^{<\omega}$ with $2^{<\omega}$ or some homeomorphic image. For example, if we are dealing with the space of (isomorphism types) of graphs, then the space will be a compact space with graphs of n vertices at height n, and the representation will be a $2^{O(n^2)}$-branching primitive recursive subtree $\mathcal{T}$ of $\omega^{<\omega}$ at each level n corresponding to graphs of height n. We will be dealing with functions on such represented spaces. Thus we will consider functions $f : \mathcal{C}_1 \to \mathcal{C}_2$ and these are represented by by functions F acting on repfresentations Q_i of $\mathcal{C}_i$, so taking finite strings to finite strings. In particular, if $A \in \mathcal{C}_1$ with then A has a filtration $A = \cup_n A_n$. It follows A represented by a path $\alpha = \lim_s \alpha \upharpoonright s \in [\mathcal{T}]$ with $\delta(\alpha \upharpoonright s) = A_s$, $f(A)$ is represented by $\lim_s F(\alpha \upharpoonright s)$.

[1]Later when we consider partial online algorithms, this might be a partial map.

[2]The point of such naming systems is to allow computational comparison of problems in differing domains. For example, think about the classical situation where names are classical Cauchy sequences in computable analysis. One domain might be the totally disconnected Cantor Space, and the other might be $\mathbb{R} \cap [0, 1]$. We might want to compare a theorem on the reals with one in Cantor space, and for purely topological reasons there can be no homeomorphism between them both ways.

We emphasise that the function F is acting on *strings* which are finite objects. These represent, e.g., graphs. It is the continuity of the action will induce a map $\overline{F}$ which is the completion of the finite maps. If x is a member of that completion; such as an infinite graph represented by a path through the relevant representing tree, then $x = \lim_n \delta^{-1}(G_n)$. We could have caused extra confusion by having a space consisting of completions *and* all initial segments, so in the case of graphs a misture of finite and infinite graphs. But as we soon see this is of no consequence since in the online case finite and infinite are indistinguishable.

Remark 2.2. The reader will note that in our definitions below, the actual representation *does* affect what we will regard as online. For example, using the current representation, graphs of height n are represented (and hence regarded) as strings of length n, although their size is $O(n^2)$. If we chose to represent graphs as binary, then we would be considering $O(n^2)$-inductive systems. As with classical complexity theory, there is usually a *natural* representation for a system we are interested in.

It is also possible to base a general theory of online algorithms upon standard spaces line 2^ω and ω^ω, but then we would have a two (primitive recursive) height functions h_1, h_2, and algorithms taking e.g. strings of height $h_1(n)$ to ones of height $h_2(n)$ at step n, in place of Definition 2.3, below. It is our intuition that the current approach is the easiest to visualise.

2.3. **Online problems.** Although our objects of study are not strings, we will implicitly identify them with their representations, in accordance with the previous subsection. In particular, if the representation space is compact then our objects can be identified with strings over a finite alphabet. More generally, in the case of $\omega^{<\omega}$ we will work over an infinite alphabet, but this case requires more care especially if we want to measure the time and space complexity of algorithms.

Definition 2.3. A online problem is a triple (I, S, s), where I is the space of inputs viewed as finite strings in a finite or infinite computable alphabet, S is the space of outputs viewed as finite strings in (perhaps, some other) alphabet, and $s : I \rightrightarrows S^{<\omega}$ is a (multi-)function which maps $\sigma \in I$ to the set of admissible solutions of σ in S.

Remark 2.4. *Some further refinements of this definition will be discussed in Subsection 2.4.2.*

For instance, for a colouring problem I will be codes for finite graphs and S for finite coloured graphs. Then $s(\sigma)$ will correspond to the collection of all admissible colourings; e.g., such that adjacent vertices are distinctly coloured. These colourings will form the space of admissible solutions.

Convention 2.5. Unless specified otherwise, the space I of inputs will be a primitively recursively branching tree of finite strings.

Because of the convention above, there usually will be a natural and primitive recursive way to transform I into a copy of $2^{<\omega}$ and back. Thus, we can think of I as the collection of finite strings in the binary alphabet. Usually the same will be true for S.

Remark 2.6. In various examples throughout the paper the formal definition of an online problem in terms of Definition 2.3 is usually left as an exercise. There will be no ambiguity in the sense that different versions will be the same up to a (primitive recursive) change of notation.

Note that the multi-valued function s does not have to be computable in general. Intuitively, to solve a problem (I, S, s) we need to find an "online" computable function f which, on input i, chooses an admissible solution from the finite set $s(i)$. The sense in which it has to be online will be clarified below.

2.4. **The definition.**

2.4.1. *An informal version of the main definition.* Recall that both representation spaces I and S are assumed to be compact with a primitive recursive modulus of compactness; i.e., it is primitively recursively branching when viewed as a tree of strings. The only informal clause of the definition below is $(O3)$; it will be formally clarified later (§2.4.4) after we develop some rudiments of the theory of primitive recursive functionals.

Definition 2.7. A solution to (a representation of) an online problem (I, S, s) is a function $f : I \to S$ with the properties:

(O1) $f(\sigma) \in s(\sigma)$ for every $\sigma \in I$;
(O2) If $\sigma \prec \tau$ then $f(\sigma) \prec f(\tau)$.
(O3) $f(\sigma)$ will be computed without delay.

- Condition (O1) says that the output of f is an admissible solution.
- Condition (O2): What we ask for is that each increment of the input yields an increment in the output. In the simplest case we ask that the height of $f(\sigma)$ is the same as the height of σ, and using the appropriate representations, we can think of this as $|\sigma| = |f(\sigma)|$.
- Regarding (O3). In the literature it is often simply assumed that $f(\sigma)$ halts before an extension of σ is received. That is, there is no a priori bound on the delay needed to define $f(\sigma)$. This is fine when counterexamples are found, defeating any such "unbounded" online algorithm. For example, when it is shown that colouring trees online needs at least $\Omega(\log n)$ many colours at level n (see Gasarch [22]), this proof does not entail any complexity considerations for the online procedure; it only has to be total (i.e., eventually halt on every input). However, for any practical online models, we would expect that the f would be in some low level complexity class like polynomial time. Using the arguments given in [5], we will *idealize* this by asking that f is *primitive recursive.* There are two natural ways to formalise what it means for such an f to be primitive recursive.

  In the simplest case we require that f is a *primitive recursive function* which maps finite strings to finite strings. In Fact 2.13 we will see that in this case our definition of online could be considered as a primitive recursive analog of ibT-reduction, but acting on compact spaces with primitive recursive branchings instead of 2^ω. Classically, ibT refers to a wtt oracle procedure $\Gamma^B = A$ with $\gamma(x) = x$ for all x, and here we are identifying sets with their characteristic functions as usual. Although ibT functionals and the induced reduction have been studied quite intensively [12, 4, 14, 16, 49] and even used in (classical) differential geometry [12, 43], as far as we know the natural primitive recursive version of it is new.

Remark 2.8. To obtain the more general definition of online from [22], replace primitive recursive ibT functionals with total *but not necessarily computable* ibT functionals. That is, allow an arbitrarily complex oracle on a separate tape to help your computation. It should be clear that this definition from [22] is significantly less constructive, let alone practical, than the seemingly very general primitive recursive approach taken by us.

In the most general case, to compute $f(\sigma)$ we may ask for a τ extending σ and make our decision based on τ. Then f should be viewed as a *primitive recursive Turing functional* (to be clarified in §2.4.3) which maps approximations to elements in $[I]$ to approximations of elements elements in $[S]$, and so that the use on σ along $\xi \in [I]$ can be some extension of σ along ξ. There are two natural ways to interpret what it means for a Turing functional to be primitive recursive. In §2.4.3 we will prove that in our setup both definitions are equivalent, and therefore the notion is robust.

2.4.2. *Taking the completion of an online problem.* A solution f to an online problem (I, S, s) induces a solution for the completion of the initial problem (I, S, s), in the sense that f can be uniquely extended to a functional $\bar{f} : [I] \to [S]$. Here $[I]$ consists of infinite strings ξ such that for every i, $\xi \upharpoonright i \in I$, and similarly for $[S]$.

In general, in Definition 2.7 we may also require $\bar{f}$ to satisfy some global property which cannot be always captured by s from Definition 2.3. For example, in Section 3 a solution must be an isomorphism between two presentations of the same infinite graph. In general, even if at every stage $f(\sigma)$ may be extendable to some isomorphism, the map associated with $\bar{f}$ may fail to be surjective in the limit. Also, in another example in Section 3 we will require our solution to work only if the input is a presentation of some fixed infinite graph, which is also a property of $\bar{f}$ rather than of any finite approximation to it. In particular, in this case admissibility of $\bar{f}$ cannot be captured by s in Definition 2.3; at least not in general.

Convention 2.9. We will refer to such properties of $\bar{f}$ as *global* and will not incorporate them into Definition 2.7.

2.4.3. *Primitive recursive functionals.* As we mentioned above, there are two natural ways of interpreting what it means for a Turing functional acting on finite strings to be primitive recursive.

In the first definition, we require that it is a Turing functional that possesses a primitive recursive time-function t which, on every input σ outputs the number of steps which f takes to compute $f(\sigma)$. In particular, $t(\sigma)$ bounds the use of the operator, that is, the length of τ extending σ which may be used in the computation of $f(\sigma)$. The length of the output $f(\sigma)$ will also be bounded by $t(\sigma)$.

The other, seemingly more general definition of a primitive recursive functional says that, for each infinite path x through the space of inputs, f is primitive recursive relative to $x = \lim_s\{\sigma \mid \sigma \prec x\}$. The latter can be formally defined by adding the characteristic function for x to the primitive recursive schema, and hence would potentially entail that $f(\sigma)$ could be arbitrarily long for various extensions of σ. Luckily these two notions are equivalent in our framework. (Recall Convention 2.5.)

Lemma 2.10. *For a primitively recursively branching I, a Turing functional $f : I \to S$ possesses a primitive recursive time-function iff f is a primitive recursive functional.*

Proof sketch. A rigorous formal proof is done by induction on the complexity of the recursive definition of f. In a slightly different terminology it will appear in [27]. A similar formal argument can be found in the appendix of [5]. We give an extended sketch.

Suppose the Turing functional f possesses a primitive recursive time function t. Using t as a universal bound on all the searches which may occur in a computation with any oracle x extending σ, we can transform the general recursive scheme (augmented with the characteristic function χ_x for x) into a primitive recursive scheme augmented with χ_x. This implication holds in general, i.e., without any extra assumption on I.

Now, assuming I is primitively recursively branching, suppose f is a primitive recursive functional in the most general relativised sense. On input σ, use primitive recursive branching of I to compute a primitive recursive bound for *all* halting computations, as follows. Consider the formal description of the operator; it is a finite description which uses elementary functions, together with a characteristic function χ for the oracle, and several applications of primitive recursion and composition. Fix some input σ. The first ever application of $\chi(j)$, where j is primitively recursively defined from σ, can have only a finite number of possible values; in fact, this number is bounded by the primitive recursive branching of I at level j. We list all possible variants of $\chi(j)$. Each such value will lead to a new version of the primitive recursive schema; we keep all of them.

For each specific choice of the value $\chi(j)$, we search for the next instance of χ in the schema and split it further into finitely many versions according to the branching of I at the respective level, etc.

The resulting process builds a primitively recursively branching tree of possible primitive recursive computations on input σ; the hight of the tree depends only on the complexity of the primitive recursive description of f. This can be turned into a primitive recursive procedure which runs all these computations one after another, thus leading to a primitive recursive calculation of the maximal possible number of steps in the computation of f on input σ. □

2.4.4. *The two formal versions of the main definition.* As before, let (I, S, s) be (a presentation of) an online problem; see Definition 2.3. Assume furthermore that the space I of inputs is primitively recursively branching. (This is Convention 2.5.)

Depending on how we interpret $(O3)$ in the informal Definition 2.7, we arrive at the following two general versions of the main definition.

The most general notion is below.

Definition 2.11. A *solution* to an online problem (I, S, s) is a function $f : I \to S$ with the properties:

- (O1) $f(\sigma) \in s(\sigma)$ for every $\sigma \in I$;
- (O2) If $\sigma \prec \tau$ then $f(\sigma) \prec f(\tau)$.
- (O3) f is induced by a primitive recursive functional. That is, there is a primitive recursive Turing functional $\Phi : [I] \to [S]$ such that, for every finite $\sigma \in I$ and every $\xi \in [I]$ extending σ, $f(\sigma) = \Phi^{\xi}(\sigma)$.

Lemma 2.10 ensures that the definition is robust. In particular, we may assume that there is a primitive recursive time function t such that $t(\sigma)$ bounds the number of steps in the computation of $\Phi^{\xi}(\sigma)$ for *any* oracle $\xi \in [I]$. We can refine this definition by assuming that the time function can be taken from some natural complexity classes, e.g., polynomial time.

The other, stronger version of the definition above uses primitive recursive functions.

Definition 2.12. A *strong solution* to an online problem (I, S, s) is a function $f : I \to S$ which satisfies $(O1)$ and $(O2)$ of Definition 2.11, and with $(O3)$ replaced by:

(O3s) f is a primitive recursion function mapping (indices of) finite strings to (indices of) finite strings.

The simple fact below shows that in $(O3s)$ we may equally require that the functional Φ in $(O3)$ of Definition 2.11 is a primitive recursive ibT functional. (By Lemma 2.10, there is no ambiguity in the notion of a primitive recursive ibT functional.)

Fact 2.13. *Suppose $\mathcal{P} = (I, S, s)$ is an online problem. Then the following are equivalent:*

1. *$\mathcal{P}$ has a strong solution witnessed by a primitive recursive function f.*
2. *The completion of $\mathcal{P}$ has a solution witnessed by a primitive recursive ibT operator $\overline{f}$.*

Proof. A description of a primitive recursive ibT operator can be rewritten into a primitive recursive scheme of a function, which will be a (strong) solution to the finitistic version of the problem. On the other hand, a primitive recursive function can be transformed into a description of a primitive recursive ibT operator. □

The fact above can be stated in terms of more narrow complexity classes, e.g., polynomial time. As before, restricting $(O3s)$ to a complexity class will give a refinement of Definition 2.12.

Henceforth we will make our primitive recursive analog of ibT explicit by calling it obT, to refer to "online bounded Turing".

Definition 2.14. An obT operator (functional) is an ibT operator (functional) which is furthermore primitive recursive.

Remark 2.15. If the space I is not primitively recursively branching or not even compact, then Lemma 2.10 no longer holds. Thus, in this case the more general Definition 2.11 becomes ambiguous. In contrast, Fact 2.13 does not rely on compactness of I, let alone its primitive recursive branching, and therefore the stronger Definition 2.12 still makes sense even for non-compact I. Of course, the classical ibT reduction is usually viewed as working on 2^ω. If our space does not have primitive recursive branching then we no longer can transform it effectively into a copy of 2^ω. But we see this aspect as a feature of the model, and not a flaw. One should expect online-ness to be generally representation dependent, at least to some extent.

Remark 2.16. Notice that in actual practice, we might also need a further generalisation of the above. Sometimes we might compute a (bounded) collection of solutions at least one of which is correct at any stage and at height n. This occurs in, for example, using automata to compute minimization problems for graphs of bounded pathwidth (or k-interval graphs, see section 4.2) given the path decomposition. We will be computing a table of $f(k)$ many solutions at each level n. For example, for finding maximal clique you would have a collection of 2^k many possible solutions. However, it appears that a suitable choice of the space of outputs S can cover this seemingly more general case too.

3. Oracle computation and uniformity

3.1. **Graph oracles do not help.** The main goal of this section is to show that a graph-oracle cannot significantly help in computing a function online. For that, we consider online functionals and online oracle computations.

Definition 3.1. We say that $f : 2^\omega \to 2^\omega$ is *online computable* if f has a representation $F : 2^{<\omega} \to 2^{<\omega}$, which is online computable in the sense above, so that for all $\alpha \in 2^\omega$, $F(\alpha \upharpoonright u(n)) = F(\alpha) \upharpoonright n$, where $F(\alpha) = \lim\{F(\sigma) \mid \sigma \prec \alpha\}$ and u is primitive recursive.

The space 2^ω can be replaced with a primitively recursively branching totally disconnected space. Identifying f with its representation F, we can unambiguously write this as $f(\alpha \upharpoonright u(n)) = f(\alpha) \upharpoonright n$, and (in view of Lemma 2.10) this should cause no problems in the case of primitively recursively branching spaces of strings. We may also allow more than one input in f.

Notation 3.2. It is natural to write $f^{\alpha \upharpoonright u(i)}(i)$ instead of $f(\alpha \upharpoonright u(i))$ and view α as an oracle. The output of $f^{\alpha \upharpoonright u(i)}(i)$ can also be interpreted as a natural number, when necessary.

Remark 3.3. There are obvious refinements of this. For example, it is natural to restrict ourselves to functionals f whose running time is a polynomial in the length of α. Also, having in mind some particularly nice primitive recursive function u, f is *u-online computable* if $f(\alpha \upharpoonright u(n)) = f(\alpha) \upharpoonright n$. An obvious case is when $u(n) = n + k$, which would be *online with delay k*. When $u(n) = n$ then the notions can be restated in terms of obT-functionals, while online with delay k corresponds to Lipschitz reducibility. Computable Lipschitz reducibility comes from algorithmic randomness ([14], Chapter 9) where it is shown that if f is online computable Lipschitz acting on 2^ω, then it preserves the Kolmogorov complexity of all sequences in the sense that for all n, $K(\alpha \upharpoonright n) \geq^+ K(f(\alpha) \upharpoonright n)$; that is $K(\alpha \upharpoonright n) \geq K(f(\alpha) \upharpoonright n) \pm O(1)$.

We will consider online functionals acting on algebraic or combinatorial structures, e.g., α could be viewed as a description of a finite segment of an infinite structure of some fixed finite relational signature, e.g., a graph. The extensions of $\alpha \upharpoonright n$ are the finitely many possible relational structures on $n+1$ elements extending the structure described by $\alpha \upharpoonright n$. The intuition is that $f^{\alpha \upharpoonright u(i)}(i)$ is expected to compute correctly only if α is an initial segment of a graph G. This is a global property; see Convention 2.9.

Definition 3.4. A function $h : \mathbb{N} \to \mathbb{N}$ is online computable from the isomorphism type of a structure G if there is an online f such that, whenever α is a description of G, $h(i) = f^\alpha(i)$.

In other words, h is allowed to use any presentation of some fixed G as its (online) oracle.

Example 3.5. To see how much extra computational power algebraic oracles can give, consider the following example. Let X be an arbitrary subset of $\mathbb{N}$, and define $A(X)$ to be an algebraic structure in the language of one unary function s, one unary predicate p, and one constant o, and which has the following isomorphism type. When restricted to s and o, it is just $\mathbb{N}$ with $s(x) = x + 1$ and o interpreted as 0. Now define $p(x) \iff x \in X$. Given any presentation α of $A(X)$, we can decide X. So, in particular, computation from an isomorphism type is potentially as powerful as just the usual oracle computation.

In view of the example above, the reader will likely find the theorem below unexpected. Its proof is however not difficult; it can be viewed as a variation of an argument in Kalimullin, Melnikov, and Montalban [27].

Theorem 3.6. *A function h is online computable from the isomorphism type of an infinite graph G if, and only if, h is primitive recursive.*

Remark 3.7. It will be clear from the proof below that the result has a natural polynomial time version. The exact definition of a polynomial time functional is a bit lengthy; see [20, 31][3]. We leave the polynomial time case to the reader.

Proof. By Ramsey's theorem, G either has an infinite clique or an infinite anti-clique; without loss of generality, suppose it is a clique. Since $g(i) = f^{\alpha \upharpoonright u(i)}(i)$, where α is *any* representation of G, we can assume that the first $u(i)$ bits of α describe a clique. Since the space of all presentations of G is primitively recursively branching, the use u is primitive recursive (see Lemma 2.10). Thus, the oracle can be completely suppressed and the trivial description of an infinite clique can be incorporated into a new procedure f_0 which does not use any oracle. On input i the procedure produces a string of length $u(i)$ which describes a finite clique, and then refers to this finite string (viewed as a partial function) whenever it needs to use the characteristic function of the oracle. This procedure is easily seen to be primitive recursive (as a function). $\square$

Informally, the result says that, from the perspective of online computation, graphs cannot code any non-trivial information into their isomorphism type; i.e., up to a change of their presentation. Both the theorem above and the main result in [15] imply that graphs are *not universal for punctual computability* – a notion which we will not formally define here (see [5]). See Kalimullin, Melnikov, and Montalban [27] for a generalisation of Theorem 3.6 to structures in an arbitrary finite relational language.

3.2. **Interactions with punctual structure theory.** In [5] we described the foundations of online structure theory. The main objects in this theory are infinite algebraic structures in which operations and relations are primitive recursive. As we argued in [5], there are natural strong connections of this new theory and the theory of polynomial-time algebraic structures (see also Alaev [1] and Alaev and Selivanov [2]) with applications to automatic structures [6]. In this paper structures

[3]The point is that care is needed with which representations are allowed. Polynomial time functionals for $(0, 1)$ typically use the so-called signed digit representation, but even for $\mathbb{R}$ there is some problem with the notion of the size of the input as discussed in, for instance, [31]. However, for any reasonable representation of graphs of size n this becomes relatively straightforward using, e.g, the standard matrix representation as in [21].

themselves do not have to be primitive recursive. However, the frameworks are closely related via, e.g., Theorem 3.9 below.

A *presentation* of a countably infinite algebraic structure in a finite language is an isomorphic copy of the structure upon the domain $\mathbb{N}$. For simplicity, we may assume that the structures in this section are all relational. In this case it becomes consistent with our framework; in particular, the space of all presentations I of a fixed structure in a finite relational language is primitively recursively branching.

Each such presentation $\alpha \in [I]$ can be viewed as an isomorphic copy of the structure upon the domain of $\mathbb{N}$. Some of these presentations will be *computable* in the sense that the relations on α will be computable predicates over $\mathbb{N}$. It is well-known that a structure may have non-computably isomorphic computable presentations. When we restrict ourselves to primitive recursive presentations and primitive recursive isomorphisms the situation becomes even more complex because the inverse of a primitive recursive function does not have to be primitive recursive. See [5] for a detailed exposition of the theory of punctually categorical structures.

The following notion is not restricted to primitive recursive presentations. A more general version of the definition below was first discussed briefly in [29] and then also mentioned in [28]. An even more general model-theoretic version of the definition can be found in [27].

Definition 3.8. *A structure G is strongly online categorical if there is an online obT operator f which, on input α and β arbitrary representations of G outputs an isomorphism from α onto β.*

In other words, there exists a primitive recursive functional $f^{\alpha;\beta}$ with both uses being the identity function, such that the associated function $h(i) = f^{\alpha\restriction i;\beta\restriction i}$ (whose output is interpreted as a natural number) induces an isomorphism from α onto β; recall the latter two are isomorphic copies of G upon the domain $\mathbb{N}$. Equivalently, we could replace the functional by a primitive recursive function of three inputs σ, τ, i where $|\sigma| = |\tau| = i$ and finite strings are identified with their indices (under some fixed natural enumeration).

The theorem below can be viewed as a variation of another result of Kalimullin, Melnikov, and Montalban [27] on punctual categoricity, but in our strongly online case the proof will be significantly simpler. Recall that a structure G is homogeneous if for any tuple $\bar{x}$ in G and any pair of elements $y, z \in G$, we have that y is automorphic to z over $\bar{x}$.

Theorem 3.9. *A structure in a finite relational language is strongly online categorical if, and only if, it is homogeneous.*

Proof. Each homogeneous structure is trivially strongly online categorical. Now suppose G is strongly online categorical. Suppose the structure is not homogeneous, and let $\bar{x}$ be shortest (of length n) such that for some z, y we have that z is not in the same automorphism orbit as y over $\bar{x}$. Construct α and β as follows. First, copy $\bar{x}$ into both and calculate the online isomorphism f from $\alpha \restriction n$ to $\beta \restriction n$. If we identify $\alpha \restriction n$ and $\beta \restriction n$ with $\bar{x}$, then f induces a permutation of $\beta \restriction n$; by the choice of n any permutation of $\bar{x}$ can be extended to an automorphism of the whole structure. Adjoin z to α and find a y' which plays the role of y over $\beta \restriction n$ under any automorphism extending the permutation $\beta \restriction n \leftrightarrow f(\alpha \restriction n)$. Then necessarily $f(z) = f(y')$, because f has already shown its computation on the first n bits.

However, by the choice of z and y', f cannot be extended to an isomorphism no matter how we extend the presentations further. □

Note that we used only totality of the obT functional in the proof. In the case when the language has functional symbols the theorem no longer holds. Of course, the notion of strongly online and of a presentation will have to be adjusted. But regardless, strong homogeneity will no longer capture the property (whatever it may be exactly).

Example 3.10. Consider the structure in the language of only one unary functional symbol s, and which consists entirely of disjoint 2-cycles. Here a 2-cycle is of course a component of the form $\{x, s(x)\}$ where $s(s(x)) = x$ and $x \neq s(x)$. According to any reasonable definition of (strong) online categoricity for functional structures, this structure has to be (strongly) online categorical. However, it is not homogeneous.

We leave open:

Question 1. Is it possible to find a reasonable algebraic description of (strongly) online categorical algebraic structures in an arbitrary finite language?

We suspect that such a description exists, and that the solution will likely boil down to setting the definitions right. If we replace obT with primitive recursive operators in Definition 3.8 we will obtain the more general notion of (uniform) online categoricity. With quite a bit of effort Theorem 3.9 can be extended [27] to this more general notion, and even beyond.

4. Weihrauch reduction and online algorithms

Weihrauch reduction is one of the central notions in computable analysis. It was coined by Brattka and Gherardi [9]. It can be viewed as a natural generalisation of computable Wadge reducibility [50]. Henceforth will use $f \leq_W g$. We have some problem we wish to solve by computing some function f. To do this we produce another problem and solve for g, and then convert g back to an instance of f. In more detail, we for functions f and g function on ω^ω-represented spaces X and Y, $f \leq_W g$, is defined to mean that there are computable A and B on ω^ω, such that for *any* p_x, and *any* representation G of g,

$$A(p_x, G(B(p_x))$$

realizes f (i.e. is a name for $f(x)$). (This is defined here for single-valued functions, but does have a mult-valued version we won't need.) This should be thought of as follows for the archetypal case of a computable metric space. We computable metric space, we take a Cauchy sequence converging to x, use B to convert this into a one converging to $B(x)$, and hence one converging to $g(B(X))$, and finally using the one converging to x and this one, to one converging to $A(x, g(B(x)))$.

The definition has a number of natural variations; some of these will be discussed below.

4.1. Weihrauch reduction and incremental computation.

In this subsection we establish a formal connection between computable analysis and computer science. More specifically, we show that a version of Weihrauch reduction borrowed from computable analysis [51] is equivalent to incremental reduction between online problems suggested in Miltersen et al. [42].

We first state Weihrauch reductions in the online setting. Suppose $\mathcal{P}$, $\mathcal{Q}$ are online problems.

Definition 4.1. *We say that $\mathcal{P}$ is strongly Weihrauch reducible to $\mathcal{Q}$, written $\mathcal{P} \leq_{sW} \mathcal{Q}$, if there exist Turing functionals Φ and Ψ such that, whenever $\sigma \in I_{\mathcal{P}}$ is an instance of $\mathcal{P}$, $\Phi^\sigma = \tau \in I_{\mathcal{Q}}$ is an instance of $\mathcal{Q}$, and whenever $\rho \in s(\Phi^\sigma)$ is a solution to Φ^σ then $\theta = \Phi^\rho \in s(\sigma)$ is a solution to σ.*

Here the reduction is strong in the sense that there is a provably more general definition of (plain) Weihrauch reduction which will be given in due course. Note that, according to the definition above, all functionals involved are obT, but this condition can be relaxed giving a less tight reduction.

Notation 4.2. We write $\mathcal{P} \leq_{sW}^{C} \mathcal{Q}$ if both obT functionals (in our sense) Φ and Ψ in the definitions above belong to a complexity class C having sufficiently strong closure properties (e.g., polynomial-time, polylogspace, primitive recursive, etc.).

Remark 4.3. The reader might wonder why we will restrict ourselves to obT-type reductions, or slight variations, for the online setting. The reason is the following. Suppose that we have two (represented) online problems I_1 and I_2. In an online way we want to use I_2 to solve I_1. Now suppose that we have some online algorithm for I_2. We could take a σ of length n representing an instance G_n of height n of I_1, and convert it into an instance σ' of I_2, and use it to produce a solution $s(\sigma')$ of I_2, which could be converted back into a solution $s(\sigma) = A(s(\sigma'))$ of I_1. The key issue we will investigate is how tight the relationships of sizes of the representations are. Ideally $|\sigma'| = |\sigma|$.

A problem $P = (I, O, s)$ is a *decision problem* if $O = \{0, 1\}$ and s is merely a predicate on I. This is the same as to say that any solution simply decides whether a predicate holds on a string or not. We say that $\sigma \in I$ is a *positive instance* of I if $s(\sigma) = 1$. Milterson et.al. [42] analysed complexity classes for online algorithms, and in a slighly more general situation than our *monotone* one where, for example, the objects only get bigger. Miltersen et. al. [42] investigate online algorithms in which input data may change with time. For example, in a graph a vertex or an edge can disappear. Their reduction takes into account the potential changes of the input.

Definition 4.4. *Let C be a complexity class. A decision problem $\mathcal{P}$ is C-incrementally reducible to another decision problem $\mathcal{R}$, denoted $\mathcal{P} \leq_{incr}^{C} \mathcal{R}$, if the following two conditions hold:*

1. There is a transformation $T : I_{\mathcal{P}} \to I_{\mathcal{R}}$ in C which maps instances of $\mathcal{P}$ to instances of $\mathcal{R}$ such that $s_{\mathcal{P}}(\sigma) = s_{\mathcal{R}}(T(\sigma))$ (i.e, σ is a positive instance iff its image is a positive instance).

2. There is a transformation Q in C which, given $\sigma \in I_{\mathcal{P}}$ and the incremental change δ to σ, where δ changes σ to σ' of the same length[4], constructs the incremental change δ' to $T(\sigma)$ (where δ' changes $T(\sigma)$ to $T(\sigma')$).

Remark 4.5. We will here only consider C to be the class of polynomial time computable functions, and hence use $\leq_{incr}^{P}$ accordingly. Milterson et. al. [42] also considered e.g C to be LOGSPACE. In [42] the authors specify the exact time bounds for all computations involved. This is the reason why they need the seemingly

[4]That is, δ is the difference between σ and σ'.

redundant part 2 of the definition above. Also, they look at auxiliary data structure generated for each instance and at the changes induced to the structure. However, from the perspective of general (e.g.) polynomial time computation this extra information is not necessary since these auxiliary bounds are evidently polynomial time.

The proposition below shows that $P \leq_{incr}^{P} Q$ is a variation of Weihrauch reduction from computable analysis which was independently rediscovered by computer scientists. Recall that strong Weihrauch reduction is witnessed by a pair of functionals Φ and Ψ.

Fact 4.6. *Suppose $\mathcal{P}$ and $\mathcal{Q}$ are online decision problems. Then $\mathcal{P} \leq_{incr}^{P} \mathcal{Q}$ iff $\mathcal{P} \leq_{sW}^{P} \mathcal{Q}$ with $\Psi = Id_{\{0,1\}}$.*

Proof. Suppose $P \leq_{incr}^{P} Q$. Then the transformation T from the definition of incremental reduction can be used as Φ in the definition of $\leq_{sW}^{P}$. Since σ is a positive instance iff $T(\sigma)$ is, $\Psi = Id_{\{0,1\}}$.

Conversely, suppose $P \leq_{sW}^{P} Q$ via $(\Psi, Id_{\{0,1\}})$, where Ψ is a polynomial functional from the space of inputs $I_{\mathcal{P}}$ of $\mathcal{P}$ to the space of inputs $I_{\mathcal{Q}}$ of $\mathcal{Q}$. Then the first part of the definition of incremental reduction follows from the assumption that Ψ is a functional in C. By the continuity of Ψ and the fact that we used Id as the second functional, it suffices to deduce a polynomial time bound on the changes in the inputs of $\Psi(\sigma)$ based on the changes in σ. But this bound is just a big-O of the bound given by Ψ. □

Following [42], we can impose specific bounds on the number of steps required for example, calculating δ' based on δ. The expectation is that it should be easier to make the change than to simply recompute $T(\sigma')$ "from scratch". All these specialised bounds can also be expressed in terms of strong Weihrauch reduction; we omit details. As an application of Theorem 4.6 and various results in [42], we can obtain a number of polynomial time and polylogtime Weihrauch reductions in the study of online algorithms.

4.2. **Weihrauch reduction and online graph colouring.** Before we discuss the role of Weihrauch reduction in online colouring problem we give a brief overview of the latter.

4.2.1. *Online graph colouring.* Many problems can be re-cast as colouring problems, for example Bin Packing. Indeed, colouring can be thought of as *avoiding configurations*. In basic graph colouring, we are simply avoiding an edge connecting vertices of the same colour, but we could instead avoid, for example, triangles or any finite set of configurations in some kind of constraint satisfaction problem. However, as this is an introductory paper we will stick to basic graph colouring. There is a large literature on this area such as Kierstead [34]. Graph colouring is quite a flexible tool, and many algorithmic meta-theorems such as for monadic second order logic (like Courcelle's Theorem (see [13, 24])) can be viewed as colouring with constraints. We believe that this material has great online potential.

We will mention some of this in this subsection. As an illustrative example, we will online colour finite or infinite *trees* (or more accurately forests). So the objects of interest are trees being presented one vertex at a time som at height n we would have a forest of n vertices. Then we can use a compact representation $T \subset \omega^{\omega}$ of the collection of forests of of height n. The result below is an easy (restated in our notation) result from the folklore essentially following from Bean [7].

Proposition 4.7. *For every online algorithm A there is a σ in T of length 2^{t-1} such that the tree A acting on $T_{F(\sigma)}$ needs at least t colours.*

We will write $\chi_A(G_\sigma)$ for the number of colours used to colour G when processed by online algorithm A. The above is nearly optimal, in that we have the following:

Theorem 4.8 (Lovasz, Saks and Trotter [40])**.** *There exists an online algorithm A such that for every 2-colourable graph G, if G has n vertices then $\chi_A(G) \leq 1 + 2\log n$.*

This brings us to the notion of a *performance ratio.* Most combinatorial algorithms we would teach in a standard combinatorics class are *offline.* This means that for a finite structure H, say, the algorithm has H as part of the input and calculates using the global structure of H.

Consider the situation of an inductive problem in a class $\mathcal{C}$, and for simplicity we will stick with colourings. The *offline* chromatic number of G_σ will be denoted by $\chi_{\text{off}}(G_\sigma)$ and for forests, we would have

$$\chi_{\text{off}}(T_\sigma) = 2.$$

Definition 4.9 (Sleator and Tarjan [48])**.** The *performance ratio* is defined as be

$$r(\sigma) = \frac{\chi_A(T_\sigma)}{\chi_{\text{off}}(T_\sigma)}.$$

Here we are stating the definition for colouring but the definition applies to any online optimization minimization problem. In the case of colouring forests, we see that the approximation ratio is $O(\log(|\sigma|))$. In the infinite case, the relevant approximation ratio is the growth rate of $r(\sigma)$ for all paths in the tree T representing the problem.

For example, a graph is called *d-inductive* (or *d-degenerate*) if the vertices of G can be ordered as $\{v_1, \ldots, v_n\}$ so that for every $i \leq n$, $|\{j > i \mid v_i v_j \in E\}| \leq d$. For example, by Euler's formula, all planar graphs are 5-inductive. For those who know some graph theory, d-inductive graphs also include all graphs of treewidth d, and extremely important class in algorithmic graph theory (see Downey and Fellows [13], for example). Again note that d-inductive graphs have a compact representation (space).

Theorem 4.10 (Irani [25, 26])**.** *Let σ represent a d-inductive graph of height n. Then* first fit *will use at most $O(d \log n)$ many colours to colour G_σ. Moreover, for any online algorithm A, there is a d-inductive G_σ such that $\chi_A(G_\sigma)$ is $\Omega(d \log n)$.*

Sometimes, this growth rate reaches a limit, as in problems with *constant* approximation ratios.

The classical example is Bin Packing. We can think of bins as colours, and the objects having sizes and the constraint being that we cannot have more objects of a specific colour than the bin constraint. That is, Bin Packing takes as input sizes $a_i \in \mathbb{N}$ and a parameter V for simplicity, and colours a_i with colour $c(a_i)$ subject to $\sum_{c(a_i)=c} a_i \leq V$, and seeks to minimize the number of colours.

Theorem 4.11 (see [21])**.** *First fit gives a performance ratio of* 2 *for online* Bin Packing.

Notice that Bin Packing is another example of colouring with constraints.

4.2.2. *Online reduction.* In this subsection we define a new version of Weihrauch reduction, and we also give a non-trivial example of such a reduction between two distinct online problems.

For convenience we use 2^ω as the ambient totally disconnected space, but otherwise use appropriate names. Let X and Y be spaces represented by 2^ω. Again we think of f and g being solutions for minimization X and Y-problems respectively. Thus, for example, we are thinking of X and Y as inductive structures with filtrations $\{X_n \mid n \in \mathbb{N}\}$ and $\{Y_n \mid n \in \mathbb{N}\}$ respectively. Then the strings of length n represent the structures of height n, and $f(\sigma)$ will represent a solution to the problem represented by σ. Thus they will have an associated *cost* which in the case of colouring is the number of colours, denoted $c(\cdot)$. We will denote f_{off} and g_{off} as offline solutions. That is $f_{\text{off}}(\sigma)$ would be the solution to the minimization problem X_n of height n with $\delta(\sigma) = X_n$, and similarly g_{off}.

We state the below for single valued functions, but again there is an analogous multi-valued version, where the solution produced for g should be within the correct ratio. The idea of the following is that on input $\alpha \upharpoonright n$, we want to compute (a representation of) $f(\alpha \upharpoonright n)$[5] To to this we will apply (a representation of) B to generate an input to (a representation of) an input for g, and then use the algorithm A to translate this back to give $f(\alpha \upharpoonright n)$. Again we emphasis that this is all working with representations, and should be read this way.

Definition 4.12. Let f, g be functions on 2^ω. Then f is called *ratio preserving online reducible to* g, $f \leq^r_O g$, if there are (type II) online computable functions A and B with and a constant d, such that for all n,

$$f(\alpha \upharpoonright n) = A(\alpha \upharpoonright n, g(B(\alpha \upharpoonright n)),$$

and the ratio of $c(f(\alpha \upharpoonright n))$ to $c(f_{\text{off}}(\alpha \upharpoonright n))$ is at most d times the ration of $c(g(B(\alpha \upharpoonright n)))$ to $c(g_{\text{off}}(B(\alpha \upharpoonright n)))$.

The fact below isolates the most important feature of the reduction.

Fact 4.13. *If* $f \leq^r_O g$ *then , for some* $d > 0$, $\dfrac{c(f \upharpoonright n)}{c(f_{\mathit{off}} \upharpoonright n)} \leq d \ \dfrac{c(g \upharpoonright n)}{c(g_{\mathit{off}} \upharpoonright n)}$.

To give a non-trivial example of an online reduction we need several definitions.

In classical colouring, Kierstead investigated online colouring of Interval Graphs. A graph $G = (V, E)$ is called a k-interval graph if each vertex v of G can be represented by a closed subinterval of $[0, 1]$ such that if I_v represents v and I_w represents w, then if $vw \in E$, $I_v \cap I_w \neq \emptyset$, such that the largest number of intersecting intervals (the *cutwidth*) is $\leq k$. These are exactly the graphs which have *Pathwidth* $\leq k$, a graph metric coming from the Robertson-Seymour minors project (see [46, 13]).

Definition 4.14. Let $ColInt_k$ denote the online problem of colouring a k-interval graph. (We leave the precise representation of the problem to the reader.)

The other online problem is on covering of an interval partial ordering by chains. A partial ordering $(P, \leq)$ is called *an interval ordering* if P is isomorphic to $(I, \leq)$ where I is a set of intervals of the real line and $x \leq y$ iff the right point of x is left of the left point of y. Interval orderings can be characterised by the following theorem.

[5] We want to avoid explicit representations, but of course we should have F representing f with F acting on 2^ω, and for any $\alpha \in 2^\omega$, $\lim_n F(\alpha \upharpoonright n)$ realizes (represents) $f(\alpha)$.

Theorem 4.15 (Fishburn [18]). *Let P be a poset. Then the following are equivalent.*

(a) *P is an interval ordering.*
(b) *P has no subordering isomorphic to $\mathbf{2}+\mathbf{2}$ which is the ordering of four elements with $\{a, b, c, d\}$ with $a < b$, $c < d$ and no other relationships holding.*

The width of an interval ordering $(P, \leq)$ is defined naturally to be the minimum over all presentations of the maximum number of intervals covering some point of $[0, 1]$. Given an interval ordering $(P, \leq)$ of width k, our goal is to cover it with as few chains as possible; the chains do not have to be disjoint.

Definition 4.16. Let $ChInt_k$ denote the online problem of covering an interval ordering $(P, \leq)$ of width k by not necessarily disjoint chains. (We leave the precise representation of the problem to the reader.)

The theorem below gives a non-trivial example of an online ratio-preserving reduction between online problems. The proof of the theorem below is essentially an analysis of the clever argument given in Kierstead and Trotter [35].

Theorem 4.17. *For any positive $k \in \mathbb{N}$ there is an online solution g to $ChInt_k$ with a constant performance ratio which can be transformed into an online solution f to $ColInt_k$ with the property $f \leq^r_O g$ via a constant $d = 1$.*

Corollary 4.18 (Kierstead and Trotter [35]). *There is an online algorithm to colour k interval graphs with a constant competitive ratio.*

Proof of Corollary. Kierstead and Trotter [35] showed that every $(P, \leq)$ (online) interval ordering of width k can be online covered by $3k - 2$ many chains. Here recall that a *chain* in a partial ordering is a $\leq$-linearly ordered subset. A collection of chains $\{C_1, \ldots, C_q\}$ *covers* $P, \leq)$ if each element of P lies in one of the chains. An *antichain* is a collection of pairwise $\leq$-incomparible elements. We will see that $ColInt_k \leq^r_O ChInt_k$ is witnessed via a reduction with constant $d = 1$. It remains to apply Fact 4.13. □

Proof of Theorem 4.17. The basic idea is quite simple. Take our online k interval graph, turn it into an online interval ordering of width k, and then consider that chain covering as a colouring. However, to see that this idea works, we need to argue that there is an online solution g to the interval chain covering problem which uses only the information about comparability of various elements, and not their ordering.

We first prove the following. Suppose that $(P, \leq)$ is a online interval ordering of width k. Then P can be online covered by $3k - 2$ many chains. We need the following lemma. For a poset P, and subsets S, T, we can define $S \leq T$ iff for each $x \in S$ there is some $y \in T$ with $x \leq y$. (Similarly $S|T$ etc.)

Lemma 4.19. *If P is an interval order and $S, T \subset P$ are maximal antichains the either $S \leq T$ or $T \leq S$.*

Proof. The argument is interesting and instructive. It uses induction on k. If $k = 1$ then P is a chain, and there is nothing to prove. Suppose the result for k, and consider $k = 1$. We define B inductively by

$$B = \{p \in P : \mathrm{width}(B^P \cup \{p\}) \leq k\}.$$

Here B^p denotes the amount of B constructed by step p of the online algorithm. Then B is a maximal subordering of P or width k. By the inductive hypothesis the algorithm will have covered B by $3k - 2$ chains. Let $A = P - B$. Now it will suffice to show that A can be covered by 3 chains.

To see this it is enough to show that every elements of A is incomparable with at most two other elements of A. Then the relevant algorithm is the greedy algorithm, which will cover A, as we see elements *not* in B.

Lemma 4.20. *The width of A is at most 2.*

Proof. To see this, consider 3 elements $q, r, s \in A$. Then there are antichains Q, R, S in P of width k with $q|Q$, $r|R$ and $s|S$. Moreover these can be taken as maximal antichains. Applying Lemma 4.19, we might as well suppose $Q \leq R \leq S$. Suppose that $r|q$ and $r|s$. Then we prove that $q < s$. Since $q|r$ and $\text{width}(P) \leq k + 1$, there is some $r' \in R$ with q and r' comparable. Since $q|Q$, $r' \notin Q$. Since the width of B is $\leq k$, there is some $q' \in Q$ q' and r' comparable. Since $Q \leq R$, there is some $r_0 \in R$ with $q' \leq r_0$. Since eR is an antichain, $q' \leq r'$. Since $q|q'$, $q \leq r'$. Similarly, there exists $r'' \in R$ with $r'' \leq s$. Since P does not have any ordering isomorphic to $\mathbf{2} + \mathbf{2}$, we can choose $r' = r''$, and hence $q < s$. ☐

Now we suppose that r, q, s, t are distinct elements of A with $q|\{r, s, t\}$. Then without loss of generality $r < s < t$ since the width of A is at most 2. Since $s \in A$ there is an antichain $S \subset B$ of length k with $s|S$. Since $s|q$, and $\text{width}(P) \leq k+1$, q is comparable with some element $s' \in S$. If $s' < q$, then $s'|r$ and hence the suborder $\{s', q, r, s\}$ is isomorphic to $\mathbf{2} + \mathbf{2}$. Similarly, $q < s'$ implies $s'|t$ and then the subordering $\{q, s', s, t\}$ is isomorphic to $\mathbf{2} + \mathbf{2}$. Thus there cannot be 4 elements r, q, s, t of A with $q|\{r, s, t\}$. Hence A can be covered by 3 chains. ☐

It is easily see that the procedure above uses only comparability of intervals. Thus, the theorem follows. ☐

Problem 4.21. *Investigate online reduction between online algorithms in the literature.*

We also expect that the online reduction may lead to new online algorithms based on the already existing ones.

Also graphs with constrained decompositions such as those of bounded treewidth, pathwidth, clique-width, etc have been extensively studied in the literature, and particularly combine well with algorithmic meta-theorems (see e.g. Downey-Fellows [13], Flum and Grohe [19], Grohe [24] for a sample).

One example is given by k-interval graphs met above which are those of pathwidth $\leq k$.. A G of pathwidth k has a *path decomposition* which is a collection of sets of vertices $V_1, \dots, V_n$ all of size $\leq k + 1$ such that for all vertices $v \in V(G)$, there is at least one i with $v \in V_i$, if $xy \in E(G)$, then for some i, $\{x, y\} \subseteq V_i$ and finally if $x \in V_i$ and $x \in V_j$ (with $i < j$) the for all $q \in [i, j]$, $x \in V_q$. The last property is called the *interpolation property*, and says that pathwidth is kind of a measure of how far you are from being either a grid or a clique.

Now *given* such a path decomposition, and some optimization property we want to solve (such as for the largest clique), if the property is definable in monadic second order logic (even with counting), then we can solve the problem by dynamic programming (actually using special automata) beginning at V_1 and finishing at V_n by the methods of Courcelle [13, 19, 24].

Problem 4.22. *Investigate the extent to which this dynamic programming is online. Presumably, it will be online for properties defined by monadic second order counting logic with counting modulo some kind of delay.*

Moreover, as we have seen above for the special case of colouring above, we get a constant ratio approximation algorithm, for a graph of pathwidth k, no matter how we are given the online presentation. The difference is that if we are a given a path decomposition as the presentation, then $k+1$ colours will suffice. But perhaps the methods for colouring are more general. The point is that graphs of bounded pathwidth have very constrained structure.

Problem 4.23. *Investigate the approximability of monadic second order definable properties on graphs of bounded pathwidth, but given as arbitrary online presentations.*

The same can be asked for graphs of bounded treewidth which has the same definition as pathwidth, but the structure of the decomposition is a tree and not a path. These also have dynamic programming algorithms, but are always *leaf to root*, whereas even given a tree decomposition as an online root to leaf structure, presumably some kind of algorithm will work, but it will no longer be automatic. This seems a great area to pursue.

5. Δ^0_2 processes, finite reverse mathematics, and Weihrauch reduction

Imagine we are in a situation where the data we are dealing with is so large that we cannot see it all. At each stage s our goal is to build a solution f to some problem. But there might be no hope of giving a fixed solution at each stage n, and we would update our solution as more information becomes available. So for each $n \leq s$ we would be computing $f(n, s)$ from the finite information σ with $|\sigma| = n$. For simplicity we state the next definition for combinatorial problems with totally disconnected representations, and take 2^ω as the representing example.

Definition 5.1. *A* limiting online algorithm *on* 2^ω *is a computable function* A *such that for each* s*,* $A(\alpha \restriction s)$ *computes a string* $\{f_A(n, s) \mid n \leq s\}$ *such that* $\lim_s f_A(n, s)$ *exists for each* n*.*

As usual we would have $A(\alpha \restriction g(s))$ *for the* g*-online version.*

We can then compare combinatorial problems by how fast their limits converge.

Definition 5.2. *We say that algorithm* $A \leq_{O,lim} B$ *if there is an online Weihrauch reduction of* A *to* B *such that* $f_B(n, s) = f_B(n, t)$ *for all* $t \geq s$ *implies* $f_A(n, s) = f_A(n, t)$ *for all* $t \geq s$*.*

This gives a fine grained measure of the complexity of combinatorial problems. For example, consider the “theorem” that every finite binary tree of height n has a path of length n. Then we can consider the existence of a uniform function A which takes a given binary tree of height n to a path. This is an online limit problem where the underlying space X is that with nodes generated by the collection of binary trees of height n at level n. The completion of this will represent paths through infinite binary trees.

Remark 5.3. We could argue that the Reverse Mathematics principle WKL_0 which states that every infinite binary tree has a path, is equivalent to the statement that there is a limiting online algorithm for finding paths which works on X. We call this *limiting online paths.*

A binary tree T of height n is called *separating* if for each $j \leq n-1$, for any node σ on T of height j, and $i \in \{0,1\}$, if $\sigma * i$ does not have an extension in T of height n, then for all τ of length j, neither does $\tau * i$. Let X_S be the totally disconnected space representing the collection of all separating finite trees. The following is a online interpretation and refinement of the classical fact that Weak König's Lemma is equivalent to Weak König's Lemma for separating classes.

Proposition 5.4. *There is a $(2^{n+1}-2)$-limiting online reduction which finds limiting online paths in X from those in X_S.*

Proof. We remind the reader of how this proof works. Suppose we have a tree T_s of height s. In an online fashion, we will generate a tree H of height 2^{s+1}. This is done inductively. At step 1, we can think of the nodes labeled 0 and 1 in T as being represented by 0 and 1 in H. At step 2, in T it is possible for us to have $00, 01, 10, 11$ and these are represented by 4 levels in H, with height 2 representing 00, level 3 01, level 4 10, and level 5 11. Now we continue inductively. This makes level n of T correspond to trees of height $2+4+\cdots+2^n = 2^{n+1}-2$. As the construction proceeds, if some σ fails to have an extension at length s, in T_s, there will be some shortest $\sigma' \preceq \sigma$ which fails to have a length s extension in T_s. Then in H_s, we don't extend to length s (from length $s-1$) all paths corresponding to $\nu * j$ with j representing σ' in H_{s-1}.

Consider any limiting online algorithm for finding a path for path α corresponding to H, in X_0, This naturally and in a online way allows us from level 2^{s+1} to generate an online path in T_s, and is clearly a limiting online reduction. □

Problem 5.5. *Figure out the smallest g in place of 2^{s+1} in the reduction above, which would give a* precise measure *of how tight the reverse mathematics relationship is.*

There seems a whole research programme available here. For example, we could be given an online bipartite graph B_σ for $\sigma \prec \alpha$. We either have to build a complete matching or demonstrate that Hall's condition fails. One representation of this problem will involve a compact space where then nodes are bipartite graphs of height $2n$, say, and where the paths all represent graphs which obey Hall's condition. The online operator will act on this compact tree of representations for graphs B_σ. Now as the process goes along, we might have to update the solution at hand. That is, the online process has $B_\sigma \mapsto M_\sigma$,

One intriguing example is that of finding a basis in a vector space. In the case that the vector space is over the rationals, then presumably this will correlate to some principle like ACA_0. But consider a finite field such as GF(2). We know that RCA_0 proves that we can find a basis for a vector space for this field. But it is not hard to construct an online vector space over GF(2) for which there is no online algorithm to do this, unless we have a computable delay. Comparing the online complexity of such problems with such computable delay would see to give significant insight into the fine structure of reverse mathematics. In this particular case, we also note that a polynomial time algorithm for finding a basis of a polynomial time vector space was proven to be equivalent to $P = NP$ suggesting intriguing connections with complexity theory.

We remark that there are many processes that have been investigated and fall under the model we have introduced. One such example is algorithmic learning theory, such as EX-learning (Gold [23]). Here one is presented with $a_0, a_0, \dots$ values for a function $f(0), f(1), \dots$, and we need to eventually print out an index for $\varphi_e = f$ from some point onwards. This is clearly an example of an online algorithm, and fits into this section as a limiting algorithm. Perhaps there are connections with this and reverse mathematics, and this remains to be explored. Another area which could be incorporated would be asynchronous computing. Here we have a series of agents $A_1, \dots, A_k$ communicating through asynchronous channels, and attempting to compute a set of functions $f_1, \dots, f_k$, where there might be e.g. some kind of crash failure meaning that one of the agents dies and stops sending signals. For example, the CONSENSUS problem asks for all the f_i's which have not crashed to give the same value. A *run* could be represented in a space of possible communications and failures. There are a number of reductions which have been produced in this area, showing that Consensus is a certain kind of minimal failure, and other problems can be solved if Consensus can (Chandra and Toueg [11]). It would be interesgting to see if these results can be placed in the hierarchy of online limiting reductions, since they appear to look like online limiting reductions.

Finally, one exciting possibility would be to include randomization in this setting. Randomized online algorithms are quite common in practice (see e.g. Albers [3]). For us we could use the theory of algorithmic randomness (see [14, 39, 44]) easily. For example, an online algorithm with randomized advice (i.e. representing a coin toss at each stage) could be done via (using 2^ω as a representative space) by considering online algorithms from $2^\omega \times 2^\omega \to S$, with S some solution space, with the first copy of 2^ω representing the problem, the second representing "advice" strings and S the solution space. The online algorithm could take $(\sigma, \tau) \to s_n$, and would run on extensions of τ provided that $[\tau]$ avoids some algorithmic randomness test, such as a Martin-Löf test. Using oracles we could also tie this to the theory of algorithmic randomness using the "*fireworks*" method of Shen (see Bienvenu and Patey [8]). These ideas remain to be explored.

6. Real functions.

So far all objects of study have been discrete. However, there is a perfectly reasonable extension of these ideas to continuous objects such as the space of continuous functions on the unit interval. There has been a lot of work on complexity theory of real functions; see, e.g., book [37]. The main goal of this section is to demonstrate the role of primitive recursion as a useful abstraction. The content of this section is not technically hard, but one can easily imagine a much deeper general framework that could emerge from these basic ideas.

Recall that a Cauchy sequence $(r_i)_{i\in\mathbb{N}}$ of rationals is fast if $|r_i - r_{i+1}| < 2^{-i}$, for every i. These are the *names* which represent the space. A function $f : [0, 1] \to \mathbb{R}$ is *computable* if there is a Turing functional Φ such that, for each $x \in [0, 1]$ and for every fast Cauchy sequence χ converging to x, the functional Φ enumerates a fast Cauchy sequence for $f(x)$ using χ as an oracle. In particular, using the terminology, we would be generating a representation of the function via names of Cauchy sequences in such a way that it is representation independent. That is, $(\Phi^\chi(n))_{n\in\mathbb{N}}$ is a fast Cauchy sequence for $f(x)$. This in particular means that, on input $(r_i)_{i\in\mathbb{N}}$, the use of $\Phi^{(r_i)_{i\in\mathbb{N}}}(j)$ corresponds to δ when $\epsilon = 2^{-j+1}$ in the standard ϵ-δ definition of a continuous function.

It is well-known that Weierstrass approximation theorem is effectivizable in the sense of Turing computability [45]. This means that a function $f : [0,1] \to \mathbb{R}$ is computable iff there is a computable sequence of polynomials $(p_i)_{i\in\mathbb{N}}$ with rational coefficients with the property

$$\sup_{x\in[0,1]}|f(x) - p_i(x)| < 2^{-i},$$

for every i.

We have seen that the most general definition of being online for combinatorial structures involves being g-online for some primitive recursive function g. That is, there is a translation between using $g(n)$ many bits of α to compute n bits of $f(\alpha)$. We have also seen that for most natural online situations, we can translate this to a wider tree where $\alpha' \upharpoonright n$ represents $\alpha \upharpoonright g(n)$, so we can use *obT* procedures. It is not completely clear if this is natural in the setting of analysis, since we might wish to stick to *standard* representations of the spaces, like 2^ω and ω^ω, as above.

We first consider the most general setting where we allow g-online for a primitive recursive g, so using $g(n)$ bits to decide the output for length n. We will call this *punctually* computable. In this case, there are two natural definitions of what it would mean for such an f to be "online" computable in the most general sense of primitive recursion. The first notion is the most straightforward sub-recursive version of the standard definition.

Definition 6.1. A function $f : [0,1] \to \mathbb{R}$ is *punctually computable* if there is a primitive recursive functional Φ such that, for each $x \in [0,1]$ and for every fast Cauchy sequence χ converging to x, the functional Φ enumerates a fast Cauchy sequence for $f(x)$ using χ as an oracle.

By restricting ourselves to dyadic rationals, we can assume that fast Cauchy sequences come from a compact totally disconnected space of the names of dyadic rationals in $[0,1]$. Thus, Lemma 2.10 can be applied to ensure that there is no ambiguity in the notion of a primitive recursive functional in this case. In particular, the definition has a natural polynomial-time version which we omit (see [37]); the same applies to any natural complexity class which may be of interest.

The second version filters through the theorem of Weierstrass. It views f as a primitive recursive point in the metric space $(C[0,1], \sup)$ rather than as a functional.

Definition 6.2. A function $f : [0,1] \to \mathbb{R}$ is *uniformly punctually computable* if there is a primitive recursive function which on input i outputs (the index of) a polynomial p_i with rational coefficients such that $\sup_{x\in[0,1]}|f(x) - p_i(x)| < 2^{-i}$.

Clearly, there is a natural polynomial-time modification of the definition above which we omit.

Every uniformly punctually computable f is punctually computable. Are these two definitions equivalent? It is not completely evident why Weierstrass approximation theorem should hold primitively recursively. Indeed, in the standard Turing computable proof we would wait for a cover of $[0,1]$ by δ_i-balls B_i such that $f(B_i)$ has diameter $< \epsilon$, for every i. It seems that even when f is punctual this search could be unbounded.

Nonetheless, the theorem below shows that these definitions are equivalent. This result is not really new. With some effort its proof can be extracted from book [37], but the book is mainly focused on polynomial time and exponential versions of the definitions above. There is much combinatorics specific to complexity theory which significantly obscures the idea behind the proof. Primitive recursion strips away complex counting combinatorics thus clarifying the idea.

Theorem 6.3. *Every punctually computable $f : [0,1] \to \mathbb{R}$ is uniformly punctually computable.*

Proof sketch. The idea here is similar to that in the proof of Lemma 2.10. Fix n and consider the functional $\Psi^x_n = \Phi^x(n)$ which uniformly primitively recursively outputs the fest few bits of $f(x)$ up to error 2^{-n}, for any input x. Since Ψ_n is given a primitive recursive scheme (with parameter n), we can work by induction on the complexity of the scheme and emulate all its possible computations at once, as in Lemma 2.10. Since the space of dyadic presentations of rationals is primitively recursively compact, this will lead to a primitively recursively branching tree of possible computations whose height is determined by the syntactical complexity of the primitive recursive scheme. By the choice of Ψ_n, one of these computations must work for an arbitrary $x \in [0,1]$. Thus, we have primitively recursively calculated an open cover $[0,1]$ by basic open intervals $J_1, \dots, J_k$, such that whenever $x, y \in J_i$ we have $|f(x) - f(y)| < 2^{-n+1}$. If z_i is the center of J_i, then define (the graph of a) piecewise linear function h_n by connecting points $(z_i, \Psi^{z_i}_n)$ and $(z_{i+1}, \Phi^{z_{i+1}}_n)$, $i = 1, \dots, n$-1. Note that the values of the $\Phi^{z_i}_n$ have already been calculated. Since the intervals are overlapping, this piecewise linear function h_n approximates f with precision 2^{-n+2}. We can primitively recursively smoothen h_n by replacing it with a polynomial p_n such that $\sup_{x\in[0,1]}|p_n(x) - f(x)| < 2^{-n+3}$. □

See Chapter 8 of [37] for a detailed analysis of the polynomial-time versions of Weierstrass approximation theorem. Recall that in the proof sketch above we generated the tree of possible computations. For a polynomial-time operator this tree may be exponentially large at worst. This difficulty cannot be circumvented and the polynomial-time analogy of the theorem above *fails* as explained in great detail in [37].

We see that punctual analysis fits somewhere in-between computable analysis and polynomial-time analysis, and there is likely much depth in the subject. Such a theory could provide us with a stronger technical link between computable and feasible analysis. Nonetheless, is seems there has been no dedicated study of primitive recursive continuous functions and punctual presentations of analytic spaces.

Problem 6.4. *Develop primitive recursive ("punctual") analysis.*

Now in the case that we want to look at the strictly online model, we are stuck with using, for instance, the bit representation of a real x, and would be working, for example, with 2^ω. Then to compute $f(x)$ with precision 2^{-n} we would need $x \upharpoonright n$. We might ask for delay k so might use $2^{-(n+k)}$. Now in this case, we see that, for example addition is online (on $2^\omega \times 2^\omega$) with delay 2, and if f is a given online computable function which is bounded then $\int_0^x f(x)dx$ would also be online computable with delay 2. We remark that this model would seem to be one emulating classical numerical analysis. This fine-grained analysis of what is actually carefully online seems completely open.

References

[1] Alaev, P., Structures computable in polynomial time, *Algebra and Logic*, Vol. 55, Issue 6, (2016), 647–669.

[2] Alaev, P., and Selivanov, V., Polynomial-Time Presentations of Algebraic Number Fields. Computability in Europe 2018, (2018) 20-29.

[3] Albers, S., Online Algorithms: A Survey, in *Mathematical Programming*, Vol. 97, (2003) 3-26.

[4] Barmpalias, G. and Lewis A., A c.e. real that cannot be sw-computed by any Ω-number, *Notre Dame Journal of Formal Logic,* 47 (2006), 197–209.

[5] Bazhenov, N., Downey, R., Melnikov A., and Kalimullin, I. Foundations of online structure theory, *Bulletin of Symbolic Logic,* Vol. 25 (2019), 141-181.

[6] Bazhenov, N., Harrison-Trainor, M., Kalimullin, I., Melnikov A., and Ng, K.M. Automatic and polynomial-time algebraic structures, *J. Symbolic Logic*. In press.

[7] Bean, D., Effective Coloration, *J. Symbolic Logic* Volume 41, Issue 2 (1976), 469-480.

[8] Bienvenu, L., and Patey, L., Diagonally non-computable functions and fireworks, submitted. (arXiv:1411.6846)

[9] Brattka, V. and Gherardi, G., Weihrauch Degrees, Omniscience Principles and Weak Computability, in: Bauer, Andrej and Hertling, Peter and Ko, Ker-I (eds.), *CCA 2009, Proceedings of the Sixth International Conference on Computability and Complexity in Analysis,* Leibniz-Zentrum für Informatik, Schloss Dagstuhl, Germany, 2009, pages 83-94.

[10] Braverman, M., On the Complexity of Real Functions *Foundations of Computer Science, 2005, FOCS'05.*

[11] Chandra T. D. and Toueg, S. Unreliable failure detectors for reliable distributed systems. JACM 43(2):225–267, 1996.

[12] Csima, B., *Applications of Computability Theory to Prime Models and Differential Geometry*, Ph.D. dissertation, The University of Chicago, 2003.

[13] Downey, R. and Fellows, M. *Fundamentals of Parameterized Complexity*, Springer-Verlag, 2013.

[14] Downey, R., and Hirschfeldt, D., *Algorithmic Randomness and Complexity*, Springer-Verlag, 2010.

[15] Downey, R., Harrison-Trainor, M., Kalimullin, I., Melnikov, A., and Turetsky, D., Graphs are not universal for online computability, submitted.

[16] Downey, R., LaForte, G., and Hirschfeldt, D., Randomness and reducibility, extended abstract appeared in *Mathematical Foundations of Computer Science, 2001* J. Sgall, A. Pultr, and P. Kolman (eds.), Mathematical Foundations of Computer Science 2001, Lecture Notes in Computer Science 2136 (Springer, 2001), 316–327). final version in *Journal of Computing and System Sciences*. Vol. 68 (2004), 96-114.

[17] Fiat, A., and Woeginger, G., *Online Algorithms*, Springer LNCS, vol 1442.

[18] Fishburn, P., Interval graphs and interval orders, *Discrete Mathematics*, Volume 55, Issue 2, July 1985, Pages 135-149.

[19] Flum, J., and Grohe, M., *Parameterized Complexity Theory*, Springer-Verlag, 2006.

[20] Friedman, H. and Ko, K, Computational complexity of real functions. *Theoret. Comput. Sci.,* 20(3):323–352, 1982.

[21] Garey, R. and Johnson, D., *Computers and Intractability*, W. H. Freeman, 1979.

[22] Gasarch, W., A survey of recursive combinatorics, in *Handbook of Recursive Mathematics*, Vol. 2., Edited by Ershov, Goncharov, Marek, Nerode, and Remmel. 1998. Pages 1041–1176. Published by Elsevier ISBN: 0444544249.

[23] Gold, M., Language Identification in the Limit, *Information and Control,* Vol. 10 (1967), 447–474.

[24] Grohe, M., *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory,* Lecture Notes in Logic, Volume 47. Cambridge University Press, 2017

[25] Irani, S., Coloring Inductive Graphs On-Line, in *Proceedings for for the 31st Symposium on the Foundations of Computer Science , 1990,* pp. 470–479.

[26] Irani, S., Coloring Inductive Graphs On-Line, *Algorithmica*, vol.11, no.1, Jan. 1994, pp.53-72.

[27] Kalimullin, I., Melnikov, A., and Montalban, A., Punctuality on a cone. In preparation.

[28] Kalimullin, I., Melnikov, A., and Ng, KM., The Diversity of Categoricity Without Delay. *Algebra and Logic*. 56(2) (2017), 171-177.

[29] Kalimullin, I., Melnikov, A., and Ng, KM., Algebraic structures computable without delay. *Theoretical Computer Science.* 674 (2017), 73-98.
[30] Karp, R., Reducibility Among Combinatorial Problems (PDF). In R. E. Miller; J. W. Thatcher (eds.). *Complexity of Computer Computations.* New York: Plenum. (1972), 85–103.
[31] Kawamura, A., and Cook S., Complexity Theory for Operators in Analysis, *ACM Transactions on Computation Theory*, 4(2), Article 5, 2012.
[32] Khoussainov, B., A quest for algorithmically random infinite structures. *Proceedings of LICS-CSL 2014 conference.* Vienna, Austria.
[33] Kierstead, H., An Effective Version of Dilworth's Theorem, *Trans. Amer. Math. Soc*, 268(1)· November 1981.
[34] Kierstead, H., Recursive and On-Line Graph Coloring, In *Handbook of Recursive Mathematics,* Volume 2, pp 1233-1269, Elsevier, 1998.
[35] Kierstead, H. and Trotter, W., An Extremal Problem in Recursive Combinatorics. *Congressus Numeratium*, 33, pp 143-153, 1981.
[36] Kierstead, H, and Qin, J., Coloring interval graphs with First-Fit. in *Combinatorics of Ordered Sets,* papers from the 4th Oberwolfach Conf., 1991), M. Aigner and R. Wille (eds.), Discrete Math. 144, pp 47-57, 1995.
[37] Ko, K., Complexity theory of real functions. Progress in Theoretical Computer Science, Birkhäuser Boston, Inc., Boston, MA, 1991.
[38] Kreitz, C., and Weihrauuch, K., Theory of representations, *Theoretical Computer Science*, Volume 38, 1985, Pages 35-53.
[39] Li, M., and Vitanyi, P., *An Introduction to Kolmogorov Complexity and Its Applications.* Texts and Monographs in Computer Science. Springer-Verlag, 1993.
[40] Lovasz, L., Saks, M., and and Trotter, W., An on-line graph coloring algorithm with sublinear performance ratio, *Discrete Mathematics*, Volume 75, Issues 1–3, May 1989, Pages 319-325.
[41] Melhorn, K., Polynomial and abstract subrecursive classes. *J. Comput. Syst. Sci.,* 12(2):147–178, 1976.
[42] Bro Miltersen, P., Subramanian, S., Scott Vitter J., Tamassia R., Complexity models for incremental computation, *Theoretical Computer Science*, Vol. 130 (1994), 203-236.
[43] Nabutovsky A., and Weinberger S., The fractal nature of Riem/Diff. I. *Geometrica Dedicata,* 101 (2003), 1–54.
[44] Nies, A., *Computability and Randomness,* volume 51 of Oxford Logic Guides. Oxford University Press, Oxford, 2009.
[45] Marian B. Pour-El and J. Ian Richards. *Computability in analysis and physics.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1989.
[46] Robertson N., and Seymour, P., Graph minors II. Algorithmic aspects of tree-width. *Journal of Algorithms*, Vol. 7, pp 309-322, 1986.
[47] Schröder, M., Extended Admissibility. *Theoretical Computer Science* Vol. 284, 519-538 (2002).
[48] Sleator, D. and Tarjan, R., Amortized efficiency of list update and paging rules, *Communications of the ACM,* 28 (2) (1985), 202–208.
[49] Soare, R., Computability theory and differential geometry, *The Bulletin of Symbolic Logic,* 10 (2004), 457–486.
[50] Wadge, W., "Reducibility and determinateness on the Baire space". PhD thesis. Univ. of California, Berkeley, 1983.
[51] Weihrauch, K., *Computable Analysis*, Springer-Verlag, 2000.

Victoria University of Wellington
Email address: Rod.Downey@msor.vuw.ac.nz

Massey University
Email address: alexander.g.melnikov@gmail.com

Nanyang Technological University
Email address: kmng@ntu.edu.sg