MINIMAL PAIRS IN THE C.E. TRUTH-TABLE DEGREES.

ROD DOWNEY AND KENG MENG NG

1. Introduction

Strong reducibilities such as the m-reducibility have been around implicitly, if not explicitly, since the dawn of computability theory. The explicit recognition of the existence of differing kinds of oracle access mechanisms began with the seminal work of Post [12]. Of interest to us from Post's paper are the so-called tabular reducibilities $\leq_{tt}$, truth table reducibility, and $\leq_{wtt}$, *weak* truth table reducibility. Following the work of Nerode [11], we know that $\leq_{tt}$ can be characterized by saying $A \leq_{tt} B$ iff there is a Turing procedure Φ with $\Phi^B = A$ *and* Φ^X *is total for all oracles* X. Weak truth table reducibility is defined via $X \leq_{wtt} Y$, iff there is a procedure Ψ with the use $\psi(x)$ is a computable function.

Quite aside from their intrinsic interest, reducibilities stronger than Turing reducibility crop up everywhere in effective mathematics. For example $\leq_{wtt}$ can be used for classifying the degrees of bases of vector spaces (Downey and Remmel [5]), it can be used to study the presentations of reals (Downey and Terwijn [6]), and are also applied to concepts used for the classifications of approximations to effective processes. For example, X is of hyperimmune-free Turing degree iff for all $Y \leq_T X$, $Y \leq_{tt} X$.

One reason for recent interest in truth table reducibility is due to its intrinsic relationship with algorithmic randomness. This relationship seems to occur because the totality of truth table procedures allows measure to be transformed from one space to another. For example, long ago, Demuth [4] showed that if $X \leq_{tt} Y$ is noncomputable and Y is random, then there is a random $Z \equiv_T X$. The reason being that the uniform measure for the space Y can be effectively transformed into another measure relative to which X is random, and this, in turn, can be re-translated into a uniform measure for Z. Other examples include Franklin and Stephan's use of tt-reducibility for classifying concepts related to Schnorr randomness (Franklin and Stephan [7]) and similar investigations of Greenberg, Franklin, Stephan and Wu [8].

The present paper was motivated by the consideration of the reals tt-below the collection of nonrandom strings. For example, let $N_C = \{x \mid C(x) < |x|\}$, where C is plain Kolmovorov complexity. We know that N_C is wtt-complete amongst c.e. sets. Kummer [10] showed that in fact N_C is also tt-complete. The argument was particularly interesting in that the process of constructing the tt-reduction was *nonuniform.*

On the other hand, Kummer showed that whether N_K, the analogous set for prefix-free complexity, is tt-complete depends on the choice of universal machine. That is, there are universal machines U_1, U_2 such that $N_{K_{U_1}}$ is tt-complete, and such that $N_{K_{U_2}}$ is not. The proof of the latter fact was rather novel and has seen applications as far as complexity, such as Allender, Buhrman, Friedman and Loff [1] where it is used for results relating randomness to PSPACE.

Date: May 28, 2014.

The first author is partially supported by the Marsden Fund of New Zealand, while the second author is partially supported by the MOE grants MOE2011-T2-1-071 and MOE-RG26/13.

In [2], Cai, Downey, Epstein, Lempp and Miller looked at the question of the *tt*-computational power of sets of random strings for various universal machines computing the prefix-free complexity K. This group showed that a set A is *tt*- below all such sets of random strings iff A is computable. On the other hand, these authors also showed that for any two universal machines U_1, U_2, there is a noncomputable set $X \leq_{tt} N_{K_{U_1}}, N_{K_{U_2}}$.

Since all such sets of non-random strings are necessarily *wtt*-complete, the question arises whether there exist *wtt*-complete c.e. sets A, B which form a *tt*-minimal pair. If this is false then the second result of Cai et. al would be a simple corollary. In [2], Cai et. al. showed that there are c. e. Turing complete sets forming a *tt*-minimal pair, but left the *wtt*-question open.

This question seems hard. Earlier, Jockusch and Mohrherr [9] showed that the diamond lattice can be embedded into the c.e. *tt*-degrees preserving the greatest and the least element. That is, there are c.e. sets A and B such that $A \oplus B \equiv_{tt} \emptyset'$ with $X \leq_{tt} A, B$ implies X computable. Also Degtev [3] had shown the existence of c.e. Turing complete sets forming a minimal pair in the *tt*-degrees of c.e. sets. It is unknown whether $A \equiv_{wtt} B \equiv_{wtt} \emptyset'$ is possible.

In the present paper, we will make modest progress towards the resolution of the question of a *wtt*-complete *tt*-minimal pair by proving that there can be $A \equiv_{wtt} \emptyset'$ and $B \equiv_T \emptyset'$ forming a *tt*-minimal pair.

The reader might guess that this result can be proved by a straightforward modification of the proof in the Cai et. al. paper. This is unfortunately far from the case; the method used in this paper is quite different. Our argument uses the determinacy of finite games and their computable winning strategies in a way hitherto unseen. We hope that this might lead to progress on the full questions. Also, in itself, the technique is very pretty and might well have other applications.

We prove the following Theorem:

Theorem 1.1. *There is a pair of c.e. sets $A \equiv_{wtt} \emptyset'$ and $B \equiv_T \emptyset'$ whose tt-degrees form a minimal pair, i.e. $X \leq_{tt} A, B \Rightarrow X$ is computable.*

2. Proof of Theorem 1.1: Informal description

We will use the well-known fact that every finite game is computably determined. Moreover there is a uniform procedure to pass from a finite game to a winning strategy for one of the two players in the finite game. We shall build c.e. sets A and B. To ensure that $B \geq_T \emptyset'$ we specify computable approximations to the markers $\{\gamma(n)\}$, satisfying the usual marker rules. For each n:

- $\gamma(n)$ is moved monotonically.
- $\gamma(n)$ is moved at a stage s only if $B \restriction \gamma(n)[s]$ has changed.
- $\gamma(n)$ is only moved finitely often.
- If n enters $\emptyset'$ at a stage s then $\gamma(n)$ is moved at or after stage s.

It is straightforward to check that these rules ensure that $B \geq_T \emptyset'$. To ensure $A \geq_{wtt} \emptyset'$, we will reserve intervals $\{I_n\}$ where each I_n is used for coding $\emptyset'(n)$. We ensure the following hold:

- The intervals $\{I_n\}$ and their lengths $|I_n|$ are chosen and effectively determined in advance.
- If n enters $\emptyset'$ at stage s then A changes on interval I_n at or after stage s.

If these two points are ensured during the construction then clearly $A \geq_{wtt} \emptyset'$. As we will see, during the construction we will be enumerating various numbers (for the sake of *tt*-minimal pair requirements) into $A \restriction I_n$, even when n has not yet entered $\emptyset'$. For the coding $A \geq_{wtt} \emptyset'$

to work, we must ensure that the interval I_n in A is not filled up before n enters $\emptyset'$. Otherwise if n enters $\emptyset'$ later on then there is no way to record this in A.

The requirements to meet are

$$R_e : \left(\Phi_e^A = \Phi_e^B = X\right) \Rightarrow X \text{ is recursive},$$

where Φ_e is the e^{th} potential *tt*-functional. For each n such that we observe $\Phi_e^A = \Phi_e^B = r$, we will begin a procedure to *certify* the computations. This certification procedure will be concluded when A or B changes below a small number, or when we obtain the desired certification. In the former case we will wait for both sides to agree again before beginning another certification procedure on the new computations. In the latter case we will *believe in the common value* r, and define the value of $X(n) = r$.

2.1. **The strategy to satisfy R while keeping $B \geq_T \emptyset'$.** We briefly describe the strategy to make C and B a *tt*-minimal pair while keeping $C \equiv_T B \equiv_T \emptyset'$; for more details we refer the reader to Cai et. al. [2]. The strategy for processing a pair (e, n) is as follows. We let $c_{e,n}$ be the least number on the C-side below the $\varphi_e(n)$-use such that the entry of any combination of $\gamma(i) > \gamma(c_{e,n})$ will not change the value of $\Phi_e^C(n)$. If $c_{e,n} \leq e$ then we say that $c_{e,n} \uparrow$; otherwise $c_{e,n}$ is always picked to be larger than e. We can similarly define the number $b_{e,n} > e$ for $\Phi_e^B(n)$. We then pick the larger of the two, say $b_{e,n}$, and enumerate $\gamma(b_{e,n})$ into C and B (and lifting all larger γ-markers). If $b_{e,n} = c_{e,n}$ we can then enumerate $\gamma(b_{e,n})$ into B and $\gamma(c_{e,n} + 1)$ into C.

We then get a disagreement $\Phi_e^C(n) = 0 \neq 1 = \Phi_e^B(n)$ which will be preserved unless $\gamma(x)$ is enumerated into $C \cup B$ for some $x \leq b_{e,n}$. This can be used to argue show that each marker $\gamma(x)$ is moved only finitely often, and so $B, C \geq_T \emptyset'$. We will only believe in the pair (e, n) when we find that one of $b_{e,n} \uparrow$ or $c_{e,n} \uparrow$. In this case at least one of the two sides of a certified agreement can be forever preserved at the original value (unless, of course $\gamma(e)$ is moved, where e is a small number relative to the e^{th} requirement).

Note that herein lies the fundamental difference between making the atoms Turing complete versus *wtt*-complete. In the case of *wtt*-complete this naive plan cannot work, because the uses for $\gamma(i)$, $i > b_{e,n}$ are not lifted by this action and so this disagreement *can be removed* even by a large $\gamma(i)$ change. When considering $A \geq_{wtt} \emptyset'$ we need to consider the possible effects of enumerating any combination of markers below the use. To help us organize this process we will need to consider games.

2.2. **The game $G(e, n, s)$.** We will define the finite game $G(e, n, s)$. The idea is that we will only believe a pair (e, n) when we find that no appropriate value of $b_{e,n}$ on the B-side is found. In this case we can ensure that the computation on the B-side can be preserved at the original value.

Definition 2.1. For $e, n, s \in \omega$, the game $G(e, n, s)$ is defined as follows. We assume the game $G(e, n, s)$ starts at stage s of the construction with both sides $\Phi_e^A(n)[s] = \Phi_e^B(n)[s]$ agreeing. For convenience we always assume that the starting common value is $r = 0$. The game is a finite game of perfect information played between two players. The first player, called the `SPOILER`, always starts first and the game alternates between a `SPOILER` move and a `PRESERVER` move. The `PRESERVER` wants to keep the starting value $\Phi_e^A(n) = 0$ (by enumerating elements in A). The `SPOILER` wants to flip the value of $\Phi_e^A(n)$ to 1 (also via enumeration into A). By artificially increasing the use by a recursive amount we may assume that $I_0, \cdots, I_N \subseteq \varphi_e(n)$ for some N.

At any point in the game, the `SPOILER`'s next legal move consists of a finite set $\{j_{n_0}, j_{n_1}, \cdots\}$ where $j_{n_k} \in I_{n_k}$ and n_ks are distinct numbers $\leq N$, such that the `SPOILER` has not previously played in any of the intervals $I_{n_0}, \cdots$, and such that $\Phi_e^A(n) = 1$ after playing these elements into A. In other words, in the entire game, the SPOILER can play at most one element in each interval I_n, $n \leq N$, and once he has played in an interval, he cannot later play another element in the same interval. A single move of the SPOILER can contain finitely many elements in different intervals, as long as the $\Phi_e^A(n)$-computation takes value 1 after playing these elements into A.

A legal move of the PRESERVER is similar. It consists of a finite set $\{j_{n_0}, j_{n_1}, \cdots\}$ where $j_{n_k} \in I_{n_k}$ and n_ks are distinct numbers $\leq N$, such that the PRESERVER has not previously played in any of the intervals $I_{n_0}, \cdots$, and such that $\Phi_e^A(n) = 0$ after playing these elements into A. Furthermore we require that each j_{n_k} is larger than some element (not necessarily in the same interval) already played by the SPOILER. So the PRESERVER plays like the SPOILER, with the extra restriction that it must play elements which are larger than something previously played by the SPOILER.

Note that "passing" is not allowed by either player since on a SPOILER turn the value of $\Phi_e^A(n)$ must be 0. The game ends when one of the two players has no more moves; in that case the player who made the last move wins.

In the game $G(e, n, s)$ either the PRESERVER or the SPOILER has a winning strategy.

2.3. **Making $A \geq_{wtt} \emptyset'$ and $B \geq_T \emptyset'$.** Let's consider a single pair (e, n) in isolation. How do we certify the pair (e, n)? If $b_{e,n}$ does not exist, then we have a certification of the pair (e, n); since in this case the computation on the B-side can be preserved forever. Otherwise we will begin a diagonalization procedure by considering the following cases:

2.3.1. *Case 1: The PRESERVER has a winning strategy on the A-side.* In this case we try and diagonalize by enumerating $\gamma(b_{e,n})$ into B and we will obtain $\Phi_e^A(n) = 0 \neq 1 = \Phi_e^B(n)$. We play the PRESERVER strategy on the A-side. What this means is that we enumerate the smallest element of $\omega - A$ in I_x whenever we find that x enters $\emptyset'$. If ever we discover that $\Phi_e^A(n)$ flips to 1 due to a (sequence of) coding actions, we will consider this to be a SPOILER move in the game $G(e, n, s)$. During the construction we immediately respond with a PRESERVER-move according to PRESERVER's winning strategy in $G(e, n, s)$; this immediately restores the value of $\Phi_e^A(n)$ back to 0.

Since the SPOILER moves first in the game, and every SPOILER-move is due to coding, also every PRESERVER-move is to respond to a previous SPOILER-move, and the PRESERVER only plays larger numbers, every enumeration into each interval can be accounted for against the coding of $\emptyset'$. At the end of the game when the SPOILER has no more legal moves, we end up in a situation where the truth table forever oputputs 0, regardless of any future coding, since the SPOILER has no more moves in the game. So in this situation, when the PRESERVER has a winning strategy, we can ensure that the diagonalization succeed by keeping the A-side output 0. The only time a diagonalization set up like this can be unsuccessful is if a number $x \leq b_{e,n}$ enters $\emptyset'$. We stop the process if this happens.

There are two outcomes to this diagonalization procedure:

- *The procedure is halted due to coding of some number $x \leq b_{e,n}$.* In this case all the moves on the A-side can be accounted for against coding, while the lifing of $\gamma(b_{e,n})$ on the B-side is accounted for against the coding of x.
- *The procdure is never halted.* We successfully preserve the disagreement.

Note that since the PRESERVER has a winning strategy on the A-side, *what we could have done*, was to have immediately believed in the computations and promise to always play the PRESERVER strategy to keep $\Phi_e^A(n) = 0$. This is fine when considering a single pair (e, n), but not desirable in full construction, because the interactions between different pairs get too complicated. So we want to keep attempting to diagonalize (if we can), until the B-side can change no more, i.e. $b_{e,n} \uparrow$, and only then do we believe in the computations. Note that this "permanent" computation is, of course permanent, assuming that no marker $\gamma(k), k \leq e$ enters B. Believing in such a permanent computation is fine because $\gamma(k)$ can change only finitely often for $k \leq e$, and the entire strategy for e can restart every time this happens.

Now we assume that the SPOILER has a winning strategy on the A-side. For a number $M \leq N$ we say that SPOILER has a M-winning strategy if there is a winning strategy for SPOILER such that in every possible run of the game the SPOILER has a response that plays only numbers from intervals with index $\geq M$. Since the SPOILER has a winning strategy in the game $G(e, n, s)$, there is a largest number $M_A \leq N$ such that SPOILER has an M_A-winning strategy in $G(e, n, s)$. (At worst we can take $M_A = 0$ which certainly works; any winning strategy is a 0-winning strategy). There are now two further cases.

2.3.2. *Case 2: The SPOILER has a winning strategy on the A-side and $M_A \geq b_{e,n}$.* We play the M_A-winning strategy for the SPOILER on the A-side to keep $\Phi_e^A(n) = 1$. We halt the procedure if a number $\leq b_{e,n}$ enters $\emptyset'$. While the diagonalization procedure is running, no number $\leq b_{e,n}$ enters $\emptyset'$, and so we know that $\Phi_e^B(n) = 0 \neq 1 = \Phi_e^A(n)$.

On the A-side we use the PRESERVER to code $\emptyset'$ into A. We explain what this means. We make the first move of the M_A-winning strategy for the SPOILER and flip the A-side to 1. We then wait for coding to flip the truth table back to 0. When this happens (if ever) we consider this as the next move of the PRESERVER. We then respond with a SPOILER move to flip it to 1, and so on. The only time we cannot run this strategy is when some small number, say $x \leq N$, enters $\emptyset'$ (smaller than any of the SPOILER's moves so far). Then we may have no more legal next move for PRESERVER and so the SPOILER's M_A-winning strategy does not know how to respond. In this case we halt the procedure, and wait for the next recovery $\Phi_e^A(n) = \Phi_e^B(n)$, and then do this certification over again.

There are three possible outcomes of this diagonalization procedure:

- *A number $x \leq b_{e,n}$ enters $\emptyset'$.* In this case we must stop the diagonalization procedure because the B-side may no longer have output 0. In this case the attempted diagonalization is destroyed, but it is okay because we have not yet believed in the computation, and we have one more number in $\emptyset' \upharpoonright N$. In A we have enumerated a bunch of numbers for the SPOILER (in accordance to the SPOILER's winning strategy), which are now wasted, but these numbers are all larger than $M_A \geq b_{e,n} \geq x$, so we can blame all the moves of the SPOILER on the coding of x.
- *A number x which is smaller than all the SPOILER moves so far has entered $\emptyset'$, so that the PRESERVER has no more legal moves in the game.* The procedure is also halted in this case. In this case all of the SPOILER moves are wasted, but are all involving numbers larger than x, so again we can account these against the coding of x.
- *The procedure is never halted.* We then have a permanent disagreement $\Phi_e^A(n) \neq \Phi_e^B(n)$. In this case on the A-side we have enumerated a bunch of numbers (for the sake of the M_A-winning SPOILER strategy). These numbers may be small and cannot be accounted for against coding (but each enumerated number must be at least as large as M_A). However this is okay because we now have a permanent disagreement, and so we can tolerate the wastages in I_n made by the SPOILER.

2.3.3. *Case 3: The SPOILER has a winning strategy on the A-side and $M_A < b_{e,n}$.* Since there are no $(M_A + 1)$-winning strategies for the SPOILER on the A-side, by determinacy the PRESERVER has a winning strategy which allows her to win, provided that SPOILER only plays numbers $\geq M_A + 1$. We then enumerate $b_{e,n}$ into B to make $\Phi_e^B(n) = 1$. (Note that $b_{e,n} \downarrow$ if and only if there is some combination of markers $\gamma(i)$, i in some finite set $F \subseteq \{e + 1, \cdots\}$ so that the entry of these markers will change the computation. In the case F exists we can choose $\min F = b_{e,n}$. So we can in this case enumerate $\gamma(b_{e,n})$, as well as all larger $\gamma(i), i \in F$ into B, to change the computation. For convenience we shall simply say that we enumerate $b_{e,n}$ into B).

We stop the procedure if a number $x < b_{e,n}$ enters $\emptyset'$. On the A-side we play the $(M_A + 1)$-winning strategy for the `PRESERVER` (as in Case 1 above), and we use the `SPOILER` moves to code. There are again two outcomes to this procedure.

- *The procedure is halted when some number $x < b_{e,n}$ enters $\emptyset'$.* In this case the lifting of $\gamma(b_{e,n})$ on the B-side is accounted for against the coding of x. On the A-side all `SPOILER`-moves (used for coding) as well as all `PRESERVER` response involves numbers $\geq b_{e,n} > x$. Hence all wastage in A can be blamed on the coding of x.
- *The procedure is never halted.* In this case no number less than $b_{e,n}$ enters $\emptyset'$. Hence $\Phi_e^B(n) = 1$ is held permanently. Since $M_A + 1 \leq b_{e,n}$, this means that all `SPOILER` moves involves numbers larger than M_A, and so the `PRESERVER` always has a response. Hence we are able to ensure $\Phi_e^A(n) = 0$.

2.4. **Some global considerations.** Notice that if the pair (e, n) has been certified, then $\Phi_e^B(n)$ *must remain at the believed value* 0 *no matter what happens in future.* Therefore, once we believe in the pair (e, n) there is no need for us to actively pursue the `PRESERVER`-strategy on the A-side. In fact, once the pair (e, n) is certified, we no longer need to look at the game $G(e, n)$ on the A-side, since $\Phi_e^B(n)$ must be 0. The sole purpose of looking at games and winning strategies on the A-side is to allow us to actively preserve a disagreement before certification, and is never used *after* we have obtained certification. Thus, there are *no real interactions* between two certified pairs (e, n) and (e', n'). We may treat this as a finite injury between pairs (e, n). The only interaction are between two pairs each trying to diagonalize, and because we have not yet believed in either (e, n) or (e', n'), it is easy to sort out any conflict between the two pairs; We can simply initialize the lower priority one (the priority is explained in the formal construction). If we need to make both A and B *wtt*-complete, then the winning strategies for different pairs (each of which are already certified) must be made to somehow cohere, and this introduces severe difficulties.

3. The formal construction

At each stage s of the construction, there are three lists:

- The *active list.*
- The *inactive list.*
- The *certified list.*

The certified list contains all pairs (e, n) which have been certified, i.e. $b_{e,n} \uparrow$. As mentioned previously if (e, n) has been certified then it may be removed from all future consideration. This list is ordered by magnitude of $\langle e, n\rangle$. The inactive list contains all pairs (e, n) which is waiting to be placed in the active list, and which the strategy for (e, n) has not begun. This list is also ordered by the magnitude of $\langle e, n\rangle$. Finally the active list contains all pairs (e, n) which we have begun the strategy. Each active pair (e, n) may be in *waiting phase*, or be in *diagonalization phase.* This list is a queue and is ordered by the time where each pair (e, n) is placed in the queue. The priority ordering is static in the sense that elements in the list do not have their priority reversed while they remain in the queue. Of course it can happen that an element leaves the list and later re-joins the end of the queue. In this case the priority of the element is lowered relative to the other elements of the list, but this only happens when it gets kicked out of the list.

At the beginning $s = 0$, place all requirements (e, n) in the inactive list, and do nothing. Suppose we are at stage $s > 0$. Suppose $k \in \emptyset'_s - \emptyset'_{s-1}$. We enumerate $\gamma(k)$ into B and change

I_k on the A-side. For every requirement on the active list which needs to respond according to some winning strategy, we do so (in fact, there can be at most one requirement which needs to respond). Initialize all strategies on the active list which are halted by this coding action (i.e. place all these strategies in the inactive list).

Now we find the smallest pair $\langle e, n\rangle$ such that e is not represented in the active list. Place (e, n) from the inactive list into the active list (this new pair joins the active list/queue at the end). This new pair starts off in the waiting phase. Now we check to see if there is a pair (e, n) in the active list currently in waiting phase, which can be moved to the diagonalization phase, we do so for the highest priority pair (this is described below). Otherwise if there is no waiting pair which can begin diagonalizing, end the construction.

A waiting pair (e, n) can begin diagonalization, if $\Phi_e^A(n) = \Phi_e^B(n)$ again (for convenience, we denote the common value 0). If $b_{e,n} \uparrow$ we move this waiting pair into the certified list, and end the stage immediately without initializing anybody. Otherwise assume that $b_{e,n} \downarrow$. Suppose that $\ell = \ell_{e,n}$ is the largest such that I_ℓ is below some $\varphi_i^A(j)$ for some active pair (i, j) of higher priority than (e, n). (Again, remember that the priority ordering amongst active pairs is according to the "queue number", i.e., the time where a pair enters the active list). Intuitively, any winning strategy which (e, n) follows henceforth should be restrained from modifying I_ℓ and below; so (e, n) does not interfere with any pair of higher priority. During the construction there will only be the following positive actions:

(i) $\gamma(k)$ enters B due to coding.
(ii) $\gamma(b_{e,n})$ enters B due to Case 1 or 3 below.
(iii) I_k is modified due to coding.
(iv) I_k is modified in response to a `PRESERVER`-strategy (in which case there is a smaller number blamed on coding).
(v) I_k is modified in response to a `SPOILER`-strategy.
(vi) I_k is modified by an initial `SPOILER`-move.

(i) and (iii) are called *coding actions*. (ii) and (vi) are called *initial actions*. We also assume that ℓ is larger than the last stage s where the pair (e, n) is initialized not due to any of the above reason (this is the case, if for example, some (e', n') of higher active priority starts diagonalizing but does not put a small number in A or B). Also if (e, n) is initialized due to reason (ii) or (vi) (an initial move) we also increase ℓ. We also assume that $\ell_{e,n}$ is larger than $\ell_{e,i}$ for every $i < n$.

As in Subsection 2.3, there are three possibilities. This time we make minor changes (to take I_ℓ into consideration).

(I) *Case 1: The `PRESERVER` has a winning strategy on the A-side.* In this case we play the `PRESERVER` strategy on the A-side and enumerate $\gamma(b_{e,n})$ into B. We obtain $\Phi_e^A(n) = 0 \neq 1 = \Phi_e^B(n)$. We stop the procedure if a number $< b_{e,n}$ or $\leq \ell$ enters B. (Note that ℓ may be much larger than $b_{e,n}$).
(II) *Case 2: The `SPOILER` has a winning strategy on the A-side and $M_A \geq \max\{b_{e,n}, \ell\}$.* We play the M_A-winning strategy for the `SPOILER` on the A-side. DO NOT LIFT $\gamma(b_{e,n})$. On the A-side we use the `PRESERVER` to code, and on the B-side we code with $\gamma(i)$. We halt the procedure if a number $\leq \max\{b_{e,n}, \ell\}$ enters B, or if the `PRESERVER` has no further move in this game.
(III) *Case 3: The `SPOILER` has a winning strategy on the A-side and $M_A < \max\{b_{e,n}, \ell\}$.* On the A-side we play the $(M_A - 1)$-winning strategy for the `PRESERVER`, and use the `SPOILER` to code. We then enumerate $\gamma(b_{e,n})$ into B to make $\Phi_e^B(n) = 1$. We stop the procedure if a number $\leq \max\{b_{e,n}, \ell\}$ enters B.

We apply an initial action for (e,n). If Case 1 or 3 applies we initialize all active requirements which are affected, i.e. whose procedure is halted by the enumeration of $\gamma(b_{e,n})$ into B. (Note that this may cause an active requirement of *higher priority* than (e,n) to be initialized.) We place these requirements back to the inactive list. If Case 2 applies we do not do this, as we did not change B, and the initial action in case 2 on the A-side does not affect any higher priority requirement, because of ℓ. Finally in any case we initialize all lower priority active requirements.

4. Verification

Lemma 4.1. *During the construction, every enumeration into A is accounted for. Hence the weak truth table reduction works, and $A \geq_{wtt} \emptyset'$.*

Proof. Looking at the list of positive actions (i) to (vi), we see that only (iii), (iv), (v) and (vi) are relevant. (iii) happens at most once for each I_k.

Only the pairs (e,n) with $e < k$ can modify I_k via (iv), (v) or (vi). For each e at most one such pair (e,n) will be diagonalizing at any one time. For each such pair (e,n) (that is diagonalizing via a `PRESERVER`- or a `SPOILER`-strategy), it will modify I_k at most one time before it is initialized. We only consider all pairs (e,n) such that $\varphi_e(n) > \max I_k$; otherwise (e,n) has no effect on I_k. Also we must have $k > \max\{\ell_{e,n}, b_{e,n}\}$ (since $k > M_A$ in Case 2, and in in the other two cases we play the `PRESERVER`-strategy so nothing smaller than $\max\{\ell_{e,n}, b_{e,n}\}$ can be played). If (e,n) is never initialized then e will never need to be considered again. If the pair (e,n) is initialized not due to a positive action (or due to (ii) or (vi)) then $\ell_{e,i}$ is increased large for every $i \geq n$, and so e will cease to be active in I_k again, and so we may neglect this case.

So to find the size of I_k we need to consider an upperbound for the quantity:

$$\sum_{e<k} (\#\text{ times } (e,n) \text{ is initialized for some } n \text{ by a positive action})$$

Assume (e,n) is initialized due to a positive action; then we ask what can this action be? Since (e,n) was initialized by a positive action, we must have some small number less than $\max\{\ell_{e,n}, b_{e,n}\}$ entering A or B: Cases 1 and 3 are clear, and in Case 2 if the `PRESERVER` runs out of moves, and since k was played by either the `SPOILER` or the `PRESERVER`, then we must also have a number smaller than k entering $\emptyset'$.

So in any case we must have a number smaller than k which enters A or B. But this is not enough, because for instance $\gamma(k-1)$ could be lifted many times due to other requirements; so we need to argue that in fact we have a change in $\emptyset' \upharpoonright k$. This number is put in by one of (i), (iii), (iv) or (v). If it is (i) or (iii) or (iv) then we clearly have a change in $\emptyset' \upharpoonright k$. If it is due to (v) and performed by another pair (e',n'), we check if (e',n') is of lower priority than (e,n), at the point of the action. If this is so then this change must be above $\varphi_e(n) > \max I_k$. If (e',n') is of higher priority then there must also have been a recent change in $\emptyset'$, and in fact less than k. In any case we can blame this initialization on a change in $\emptyset' \upharpoonright k$. □

Lemma 4.2. *A requirement (e,n) is initialized only finitely often.*

Proof. Let (e,n) be the least such that (e,n) is initialized infinitely often. Thus there is some stage s_0 large enough such that after s_0,

- A and B are stable up to $\varphi_e(n)$.
- Every smaller pair (i,j) is never again initialized.

Claim 4.3. There are only finitely many pairs (p,q) such that at some point (p,q) is active and of higher priority than (e,n) and $\langle p,q\rangle > \langle e,n\rangle$

To see this, look at the first stage after s_0 where (e, n) is active. At this point there are only finitely many larger (p, q) of higher active priority. No new (p, q) can sneak in after this, because (e, n) is never satisfied, and so the only thing that can remove (e, n) is an initialization to (e, n). But then (e, n) will immediately be placed back into the active list above (p, q), if $p > e$. (Of course smaller (p, q) can sneak in above (e, n), but not a larger one).

Thus we may further assume that s_0 is large enough so that

- No new (p, q) with $\langle p, q\rangle > \langle e, n\rangle$ appears in the active list above (e, n).
- All (p, q) in the active list above (e, n) is never again initialized.

Thus after s_0, the set of requirements of higher active priority than (e, n) is fixed and does not change. From this point on, what cam result in an initialization to (e, n)? Since all higher priority requirements are stable, this initialization must be due to a positive action. It cannot be due to coding, by assumption on s_0, so (i) and (iii) are out. Similarly (ii) is not possible (even if it is done be a lower priority requirement). In fact nothing is possible. □

Lemma 4.4. *Each R_e is satisfied, and the tt-degrees of A and B form a minimal pair.*

Proof. Since (e, n) is initialized finitely often, the final state of (e, n) is either permanently diagonalized, or it is certified, or we wait forever in the case of non-convergence. □

Lemma 4.5. *Each $\gamma(k)$ is lifted finitely often, and so $B \geq_T \emptyset'$.*

Proof. Suppose that $\gamma(m)$ is stable for all $m < k$. Note that only requirements R_e for $e < k$ can lift $\gamma(k)$. Coding only happens at most once. So it must be moved infinitely often due to an initial action by some pair.

By Lemma 4.2, for each n, the first n elements of the active list/queue is eventually stable. Furthermore each $\ell_{i,j}$ for each stable pair (i, j) of the active list is also eventually stable. How many stable elements of the list can have $b_{i,j} \leq k$? Only finitely many. Let $M - 1$ denote the largest position of the list which is occupied by a stable pair with $b_{i,j} \leq k$. Once everything below M is stable, we claim that $\gamma(k)$ cannot be moved. Suppose it is moved by some pair (e, n). Before the action, the pair (e, n) must occupy a position $> M$ (by the stability of the first M elements of the list). However every requirement between the M^{th} position and (e, n) must have $b > k$, which means that they would be initialized, and so the pair (e, n) must move to position M itself, a contradiction. □

References

[1] Eric Allender, Harry Buhrman, Luke Friedman, and Bruno Loff, Reductions to the set of random strings: The resource-bounded case, in *Proc. 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12), 2012*, Lecture Notes in Computer Science.

[2] Mingzhong Cai, Rod Downey, Rachel Epstein, Steffen Lempp and Joseph Miller, Random strings and truth table degrees of Turing complete c.e. sets, in preparation.

[3] Alexander Degtev, *tt*- and *m*-degrees, *Algebra i Logika*, 12 (1973), 143-161. (trans 12 (1973), 78-89)

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

[5] Rod Downey and Jeffrey Remmel, Classification of degree classes associated with r.e. subspaces, *Ann. Pure and Appl Logic*, 42 (1989) 105-125

[6] Rod Downey and Sebastian Terwijn, Computably Enumerable Reals and Uniformly Presentable Ideals, *Archive for Mathematical Logic* Vol. 48 (2002), 29-40.

[7] Johanna Franklin and Frank Stephan, Schnorr trivial sets and truth-table reducibility. *The Journal of Symbolic Logic*, 75:501–521, 2010.

[8] Johanna Franklin, Noam Greenberg, Frank Stephan, and Guohua Wu, Anti-complexity, lowness and highness notions, and reducibilities with tiny use, *Journal of Symbolic Logic* 78 (2013), pp. 1307-1327.

[9] Carl Jocksuch and Jeanleah Mohrherr, Embedding the diamond lattice in the recursively enumerable truth-table degrees *Proc. Amer. Math. Soc.*, 94 (1985), 123-128.

[10] Martin Kummer. On the complexity of random strings. In *13th Annual Symposium on Theoretical Aspects of Computer Science* (*STACS '96*), Lecture Notes in Computer Science 1046, pages 25–36. Springer, 1996.

[11] Anil Nerode, General topology and partial recursive functionals, In *Summaries of talks presented at the Summer Institute for Symbolic Logic*, pages 247–251. Cornell University, 1957.

[12] Emil Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society* Vol. 50 (5) (1944), 284–316.

School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, PO Box 600, Wellington, New Zealand
E-mail address: `Rod.Downey@vuw.ac.nz`

Division of Mathematical Sciences, School of Physical & Mathematical Sciences, Nanyang Technological University, 21 Nanyang Link, Singapore
E-mail address: `kmng@ntu.edu.sg`