DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch047

Exploring the Point-mutation Space of a von Neumann Self-reproducer within the *Avida* World

Tomonori Hasegawa[1] and Barry McMullin[1]

[1]The Rince Institute, Dublin City University, Ireland
tomonori.hasegawa2@mail.dcu.ie, barry.mcmullin@dcu.ie

Abstract

The architecture of machine self-reproduction originally formulated by John von Neumann is studied within the artificial life system *Avida*. We describe a hand-designed von Neumann style self-reproducer, and report initial results from an exhaustive search of its single-point-mutation space. Unsurprisingly, the majority of mutants are simply sterile, and have no long-term evolutionary potential; however, automated characterisation and classification of the minority, fertile mutants proves to be difficult. We identify specific limitations of the standard Avida analysis tool for this particular purpose, and outline how it may usefully be enhanced.

Introduction

The nature of machine self-reproduction was investigated by von Neumann, largely in the early 1950s (von Neumann, 1951, 1966). Inspired partly by Turing's abstract model of computing machines, von Neumann formulated a general architecture for self-reproduction, with a decomposition into active, constructive machinery and a separate, passive "description tape". This work significantly preceded the discovery of the structure of DNA in 1953, but reflects a similar abstract structure to that which is now known to support self-reproduction in biological organisms. Thus, in the von Neumann architecture, the active machinery may be considered as representing the *phenotype* and the passive description tape as the *genotype*.

This von Neumann architecture for self-reproduction is shown schematically in Figure 1. A parent machine (to the left) reproduces an offspring machine (to the right). A self-reproducer in this style consists of a phenotype P and a genotype G (or, in an individual, instantiated machine, these may be called "phenome" and "genome" respectively). P consists of a *programmable constructor* (A), a *copier* (B), a *control* (C), and arbitrary *"ancillary" machinery* (D). G is a tape that describes P (i.e. the assembly $A+B+C+D$), relative to the specific description language, or "decoding" implemented by A. In operation, A decodes G (to produce another instance of P=$A+B+C+D$), B constructs a copy of G, and C controls and co-ordinates these actions, ultimately detaching the complete offspring machine instance, $P+G$, identical to the parent and thus realising self-reproduction. As this basic architecture and self-reproducing functionality will be common for any arbitrary D (within the constructive capabilities of A, and assuming that D operations, whatever they may be, do not interfere with $A+B+C$) this implies the existence of an indefinitely large space of self-reproducing machines, all connected via spontaneous perturbations of the G component (which therefore correspond to heritable mutations). Excluding such perturbation, and in the absence of resource constraints, any specific strain of such machine can exhibit exponential population growth. This potential for exponential growth, combined with mutation (variation) and resource constraints will give rise to conventional, neo-Darwinian, selection and evolution.

A significant additional feature that the von Neumann style of machine self-reproduction can theoretically exhibit is the evolvability of the genotype-phenotype mapping itself (McMullin, 2000) — i.e., of the "decoding" function implemented by the component sub-machine A. This possibility arises provided A is itself described (in a self-consistent way) within the genotype, G, and in sufficient detail that there exist potential mutations (perturbations of G) that do change the decoding function implemented by the (expressed, mutated) A in the following generation. In general, this mutated A may or may not be capable of decoding the inherited genome (description tape) G in a way that still preserves the self-reproduction functionality; albeit, the prima facie likelihood is that such mutational events will fundamentally disrupt self-reproduction. For a change in the description tape to be truly inheritable, and for a consequent machine to be self-reproducing, the reproduction mechanism must somehow survive through mutational events, sustaining a genotype-phenotype mapping that is still applicable (i.e. "backward compatible") to the mutated description, so as to keep *self*-reproducing. The current paper is concerned precisely with exploring this possibility empirically, at least for one "toy" example of a von Neumann self-reproducer.

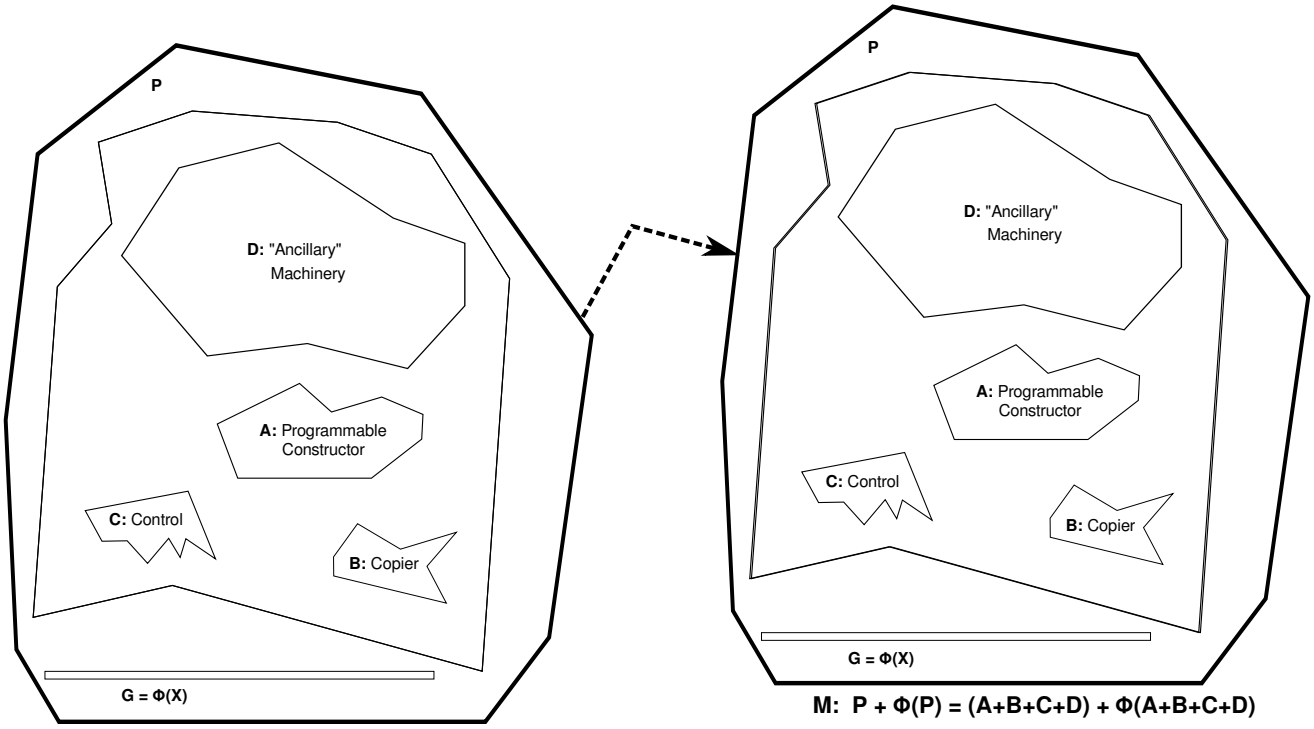

Figure 1: The schematic von Neumann style architecture of machine self-reproduction, excerpted from McMullin (2012).

The *Avida* World

Although von Neumann originally elaborated his abstract architecture in the form of cellular automaton works, it can also be successfully implemented within *Core World* type systems. In a typical Core World type of system, user-designed machine-code programs ("organisms") execute, reproduce, and compete for limited resources such as memory space and CPU time. As a result, those systems are expected to display evolutionary dynamics over time. Among others, the *Avida* system (Adami, 1997), which originated in the early 1990s, is an example of such a Core World formalism, but with an additional spatial structure. The latter is represented as a homogeneous two-dimensional grid of (virtual) microcontrollers (CPU + small local memory), each of which can instantiate a single running organism.[1] Typically initialised with one seed organism (*ancestor*), one can observe the population growth and evolution in each experimental run.

The standard Avida ancestor self-reproduces by means of self-inspection: the program copies its entire memory image word by word to create an offspring's memory image. This latter is (conceptually) divided off, and replaces the memory image in one of the neighbouring cells (see Ofria and Wilke (2004) for more detailed description). Once divided, the parent and the offspring organisms (each with a re-initialised/reset CPU state) continue execution according to their individual configuration (memory images). Such organisms will increase in number, and with mutation, variation may occur among the individuals of the population and give rise to evolution. Our investigation is conceptually similar to this standard Avida approach, except that the ancestor organism is designed with a von Neumann architecture instead of as a direct self-copier.

Computationally speaking, a genotype-phenotype mapping in this framework can simply be regarded as an arbitary, Turing computable mapping between integers (representing the parent and offspring phenotype memory images). In the light of this, a mutable genotype-phenotype mapping through a mutable programmable constructor depends on whether two different such mappings, related via a perturbation on the description tape, can maintain backward compatibility with each other.

We have previously described the implementation and characterisation of a specific prototype von Neumann organism in Avida, implementing such a self-reproduction architecture, as a basis for investigating evolvable genotype-phenotype mapping (Hasegawa and McMullin, 2012). This design will now be briefly outlined.

[1] Although the Avida world superficially resembles to von Neumann's early formulation of an abstract cellular automaton (CA) world, there are also fundamental differences. In the von Neumann CA, each node was a simple finite state automaton with no general purpose memory system; whereas each Avida node comprises a general purpose CPU and a substantial general purpose memory system.

The Prototype von Neumann Organism

The phenome of the prototype program is coded using possible word contents defined in the Avidan instruction set (see Table 1). Along with the preset standard 26 Avidan instructions, the `read` and `write` instructions are also enabled for this investigation, in order to facilitate flexible reading of the description/genome G, and writing (construction) of the offspring phenome (memory image). The instruction set configuration in Avida defines what word contents are executable in a particular run, by sequentially associating "op-codes" (mnemonics) with natural numbers according to the order in which they are listed. This set defines possible word contents as numbers from 0 to 27.[2]

For the purposes of this example prototype, the genotype-phenotype mapping (decoding) is chosen to be a simple, sequential, block code. This is inspired by the biological genetic code, relating the sequential primary structures of DNA and corresponding proteins. However, unlike the biological genetic code, where the "alphabets" are different, disjoint, and of different cardinality (nucleotides and amino acids respectively), in our case the alpabets are identical (the distinct values of a single memory location, limited just to the sufficient set to represent each implemented instruction with one op-code, i.e., the numerical range $0..27$, per Table 1). Further, choosing a fixed block size of one, this becomes essentially a sequential, monoalphabetic substitution function. This is conveniently implemented via a lookup table, of length 28. Each genome word is sequentially used as an index into this table to find the corresponding "decoded" word to be written into the (offspring) phenome. Note that, with such a coding scheme, a corresponding genome and phenome will always be of equal length; and that, unlike the biological genetic code, there is no redundancy (i.e., each possible phenome word value is represented by one, and only one, possible genome word value).

The designed prototype thus decomposes into the phenome, the first half, and the genome, the second half of the complete memory image. The phenome has five functionally separate regions, namely Decode Preparation, Decode Loop, Copy Preparation, Copy Loop, and Translation Table (see Figure 2 for the prototype's schematic design and Table 2 for its region allocation and correspondence; find the actual program at our website referenced at the end of the conclusion section). In terms of the generic von Neumann architecture introduced earlier, the Decode Loop along with the Translation Table correspond to the programmable constructor A, the Copy Loop corresponds to the copier B, and Decode Preparation and Copy Preparation correspond to the control C.

The prototype incorporates von Neumann's architecture except for the (arbitrary) ancillary machinery D, which is not essential for reproduction per se (i.e., D can be null, without violating the abstract architecture) and it is not relevant to our immediate investigation of mutations affecting the genotype-phenotype mapping (i.e. the programmable constructor A).

The designed prototype organism has a total memory image of 644 words (i.e. 322 for each half, Phenome and Genome). Note that because the phenome corresponds to the genome on a sequential one-to-one basis, one can locate a unique genotypic word corresponding to any given phenotypic word. In practice, the Phenome segment was designed first, with the Genome segment of the same length being a "black box". Then, the fully designed Phenome segment was reverse-translated into the corresponding Genome segment, so that the phenome and genome pair can produce the identical pair as an offspring.

Word Content	Op-code	Operation
0	`nop-A`	No-operations
1	`nop-B`	
2	`nop-C`	
3	`if-n-equ`	Flow Control Operations
4	`if-less`	
5	`if-label`	
6	`mov-head`	
7	`jmp-head`	
8	`get-head`	
9	`set-flow`	
10	`shift-r`	Single Argument Math
11	`shift-l`	
12	`inc`	
13	`dec`	
14	`push`	
15	`pop`	
16	`swap-stk`	
17	`swap`	
18	`add`	Double Argument Math
19	`sub`	
20	`nand`	
21	`h-copy`	Biological Operations
22	`h-alloc`	
23	`h-divide`	
24	`IO`	Input/Output and Sensory
25	`h-search`	
26	`read`	(Additionally Enabled Operations)
27	`write`	

Table 1: The instruction set configuration.

[2]The underlying memory word size is normally 32 bits wide. Numbers beyond the size of the instruction set are associated with the op-codes sequentially in a cyclic manner, so that every possible word value is also interpretable/executable as some instruction.

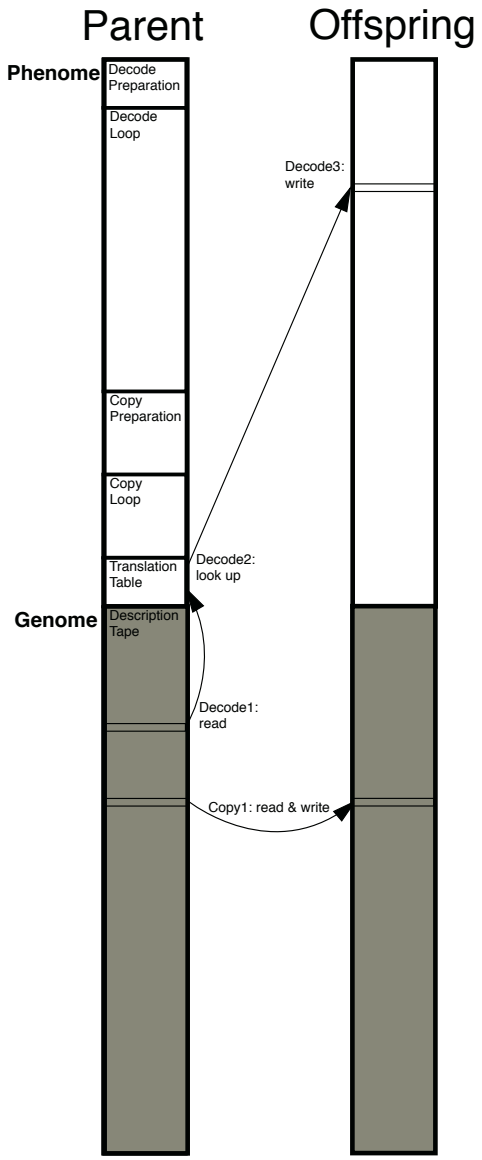

Figure 2: The schematic design of the self-reproduction by the prototype.

Region	Address (in Phenome)	Code Address (in Genome)
Decode Preparation	0–27	322–349
Decode Loop	28–193	350–515
Copy Preparation	194–245	516–567
Copy Loop	246–293	568–615
Translation Table	294–321	616–643

Table 2: The five regions of the prototype's phenome and the corresponding regions in the genome.

The Translation Table is a "data" segment in the Phenome: it is not to be directly executed (treated as containing instructions), but is referred to, as data, in implementing the decoding. Some concrete substitution code (permutation of the allowed word values 0..27) has to be arbitrarily chosen: we simply used the reversed sequence, 27..0 (i.e., 0 decodes as 27, 1 as 26, etc.).

Once the prototype is seeded in the Avida world, Decode Preparation and Decode Loop are initially executed and decode the genome to create the offspring's phenome. One step of decoding is as follows: a source word is read from the genome (Decode1), and a destination word is looked up via the Translation Table (Decode2) and is written in a corresponding location in the prospective offspring's phenome (Decode3). Subsequently, Copy Preparation and Copy Loop are executed and copy the genome to create the offspring's genome. One word is read and written at one step (Copy1). For a complete self-reproduction, it takes 52218 Avidan instruction cycles (steps). This is the prototype's *gestation time* (i.e. reciprocal of reproduction rate).

In previous work, we observed the prototype's behaviour and characteristics as an ancestor. The design of the prototype proved to work out correctly, demonstrating one implementable instance of a von Neumann style self-reproducer within Avida, a Core World type of system. A noticeable finding from this work was that the self-reproducer can degenerate into a pure self-copier only through one step of single-point mutation in the course of evolution. That is, there is at least one possible mutational pathway for the ancestor to become a self-copier.[3] Though this could presumably happen through a longer gradual evolutionary process, the finding in this context suggests that even one step of

[3]Thus, it is theoretically possible to vice versa also: that the prototype can degenerate into a self-copier means that it is likewise possible for a self-copier (or, at least, that particular self-copier) to evolve to a von Neumann self-reproducer.

single-point mutation can bring about unpredictable changes in the behaviour of the prototype.

Still, it is not clear yet how typically the prototype potentially degenerates (or exhibits other similar phenomena); therefore it is legitimate to focus on the particular von Neumann style self-reproducer and to extend the investigation of its mutational pathways. This study can reasonably be a springboard to better understand the von Neumann style self-reproduction within the system, considering the vast space of possible strains of organisms in the current Avida setting. The space appears to roughly consist of reproducers and non-reproducers; further, the reproducers include self-reproducers and other kinds of reproducers; furthermore, self-reproducers may comprise self-copiers, von Neumann style self-reproducers and some other classes.

Empirical Investigation: Point-mutation Space Search

For a better picture of the mutational pathways originating from the prototype, we first systematically counted out possible single-point mutants as candidates for organisms that have evolutionary potential, especially ones that maintain von Neumann style self-reproduction.

Candidates are obtained from the prototype's initial memory image by sequentially replacing a word content in each memory location of the genome with the other possible word contents listed in the instruction set and by expressing this change in a corresponding phenome. Each such candidate is an organism in Avida; or more properly, it is an initial memory image coupled with an initial virtual CPU configuration. Now, the genome size is 322 (the half of the whole length of the prototype), and the size of the current instruction set is 28. Considering the combination of the genotypic memory locations and the different possible word contents for each memory location (i.e., excluding the one originally existing in the location), the number of candidates is therefore 8694 ($= 322 \times (28 - 1)$).

For the purpose of characterising the prototype, the candidates should ideally be classified by the "mode" of reproduction. However, reproduction mode classification is not straightforward, since a reproduction mode is not simply determined by a single attribute of reproduction or lineages, measurable when an organism is run and traced. Nonetheless, there are at least two distinct self-reproduction modes: the pure self-copying by inspection as in the standard Avida ancestor and the von Neumann style as in the prototype. Also, one can speculate that there may be various modes aside from, or in between, these two. At any rate, to effectively judge the reproduction mode, it is generally useful to analyse in combination the candidates' attributes (e.g. gestation time) and their execution profiles (e.g., instruction pointer traces and execution counts of particular instructions, such as the `write` and `h-copy` instructions, which write some content in a memory location). They, however, do not yet necessarily guarantee what reproduction mode that a candidate uses. Rather, we need to observe how each offspring is created; besides, *viability* cannot be determined until the candidate's lineage is wholly traced for a proper number of generations.

As a preliminary step of the investigation, we attempted to classify the candidates by viability. Viability was judged initially by the Avida built-in analysis tool (specifically, using a `TRACE` command in a mode called *analyze mode*), which can trace a single lineage pathway starting from an incubated organism.

The *Analyze Mode*

Technically, according to the `TRACE` command in the Avida analyze mode, an organism running for a given generation is classified into one of four possible classes:

- If no division occurs within a predefined cut-off time, the mode concludes that the starting organism is *non-dividing* (meaning non-reproducing); or else,
- If division occurs and the immediate offspring is identical, the starting organism is classified as *immediate self-reproducing*; or else,
- If division occurs and the offspring is not identical to the immediate parent, but identical to any of the (already traced) ancestors, then the starting organism is classified as *indirect self-reproducing*; or else,
- If division occurs and the offspring is identical to none of the direct ancestors, the starting organism is classified as *reproducing* (meaning non-self-reproducing); this final class potentially triggers a recursive analysis one further generation deep, unless terminated by reaching a maximum depth.

This automated analysis assumes a cutoff time, a certain window of maximum runtime, so as to judge as non-dividing or not. In order to be classified as some reproducing class, division must have taken place (i.e., the `h-divide` instruction must have been executed) before the cutoff time is reached; otherwise, the analysis run terminates classifying the organism as non-dividing. The cutoff time is set twice as long as the prototype's gestation time (i.e. $104436 = 2 \times 52218$). This setting is reasonable considering the fact that the candidates can have varied gestation times. It is possible that, with a longer runtime, some candidates which are classified as non-dividing might be reclassified as reproducing. They are, however, unlikely to be selectively favoured over the original prototype ancestor in the Avida system, where fitness hinges on reproduction rate hence gestation time.[4] On the grounds of this, we can heuristically discard those classified as non-dividing.

[4] As opposed to these, the ones with shorter gestation time can potentially outnumber the others in the world where it boils down to the fitness, especially when the run is sufficiently long.

Out of the four classes, the non-dividing class indicates non-viability whereas the immediate self-reproducing class and the indirect self-reproducing class indicate viability. On the other hand, the reproducing class does not necessarily indicate viability or non-viability, as the offspring may or may not be self-reproducing; therefore only this class requires more generations to track down the lineage pathway in order to determine the viability. The analyze mode applies this classification repeatedly over generations as necessary to judge viability.

It is important to note that, in fact, the analyze mode only traces the offspring that is divided off from the parent at each division, assuming that viability means the ability to sustain self-reproduction on a single lineage pathway. To clarify this point, consider that each division can be regarded as making two offspring: one that used to be a parent, the other that has been divided off. The former replaces the parent and the latter replaces one cell in the neighbourhood, so spatially it appears that the parent remains sitting in the original cell and producing and placing one offspring into an adjacent cell. What is traced by this tool is the *divided-off* offspring, not the *sitting* one: only a single lineage pathway is revealed for each incubated organism (as opposed to the whole lineage with possibly multiple pathways).

First-division Patterns

To support the somewhat limited analysis by the built-in tool, we applied a more refined classification based on certain patterns characteristic of the first-division event (for all cases other than simple *non-dividing*). This reclassification helps clarify the lineage pathways one-step further.[5] As pointed out earlier, division makes two organisms: the two offspring that the initially placed organism produces. One of the two offspring should be labelled as the organism that used to be the parent, and the other offspring as the organism divided off from the parent, which treatment is somewhat arbitrary. There are patterns in these three memories, depending on which of them are the same and/or different. Here, assume that an organism has an initial memory (I; the parent's initial memory image) which becomes a final memory (F; one of the offspring's initial memory image) when dividing off a child memory (C; the other offspring's initial memory image). There are six division patterns at one attempt of division. Assume they are notated as follows using I, F, and C for the sake of convenience:

- I: Non-dividing;
- $I = F = C$: Self-reproducing. Both of the offspring's initial memory images are identical to the parent's initial memory image;
- $I = F \neq C$: One of the offspring's initial memory image is identical to the parent's, while the other offspring's is non-identical;
- $I = C \neq F$: Essentially the same as the previous one, but this pattern is treated as self-reproducing by the analyze mode;
- $I \neq F = C$: The two offspring's initial memory images are non-identical to the parent's, but identical to each other; and
- $I \neq F \neq C$: Neither of the two offspring's initial memory images is identical to the parent's or to each other.

The first attempts of division in the candidates' single lineage pathways revealed by the analyze mode were studied and the candidates were reclassified based on these patterns.

Results

The whole candidates distribute as shown in Table 3 when classified by "viability" using the built-in analysis tool and when by "fertility" based on the first-division patterns. As the analyze mode's ability to trace lineage pathways is not full, its "viability" judgment is different from what we originally meant: by viability we originally meant some continuous reproducibility with evolutionary potential. Here, in order to describe what is revealed by the analysis, we define *fertility* as follows: an organism is classified as fertile if its lineage is fully traced and in the lineage there is at least one strain that is classified as immediate or indirect self-reproducing.

Initially, of the whole candidates, the immediate self-reproducers (fertile, but only judged by tracing one offspring) accounted for nearly 10%, the pure non-reproducers (fertile, being non-dividing) for nearly 60%, and the rest 30% (fertile, but at least not immediate self-reproducing) remained unclassified.[6] The reclassification based on the first-division patterns revealed some of the unclassified that are infertile and discovered the others that have an untraced lineage pathway(s).

In other words, out of those initially classified as "viable", some lineages turn out not to have been fully traced; even though it is shown by the analysis tool that they have at least one viable reproduction pathway, they should be reclassified as unclassified. Those initially classified as "non-viable" are wholly reclassified as infertile without further scrutiny as they have no division executed. Out of those initially unclassified, some lineages turn out to have been fully traced, whether it be fertile or infertile, now putting fewer candidates in the unclassified.

[5] Although ideally all the attempts of division should be classified to investigate each organism's whole lineage, the automatisation of such classification requires considerable enhancement as discussed later.

[6] In hindsight, the analyze mode found no case of the indirect self-reproducing class among the candidates. As for the reproducing class, it found a few long cases with more than 300 recursions.

Immediate Classification	"Viable"	Unclassified	"Non-viable"	Total
Total	892 (10%)	2588 (30%)	5214 (60%)	8694

1st Division Pattern	Fertile	Unclassified	Infertile	Total
I	0 (0%)	0 (0%)	5214 (100%)	5214
$\mathbf{I = F = C}$	871 (100%)	0 (0%)	0 (0%)	871
$\mathbf{I = F \neq C}$	0 (0%)	158 (8%)	1745 (92%)	1903
$\mathbf{I = C \neq F}$	0 (0%)	21 (100%)	0 (0%)	21
$\mathbf{I \neq F = C}$	1 (3%)	11 (28%)	27 (69%)	39
$\mathbf{I \neq F \neq C}$	0 (0%)	646 (100%)	0 (0%)	646
Total	872 (10%)	836 (10%)	6986 (80%)	8694

Table 3: Automated mutant "viability" classification (Top) and mutant fertility classification with the first-division patterns (Bottom).

Consequently, the fertile account for 10%, the infertile for 80%, and the unclassified (fertile but not immediate self-reproducing; may be infertile, reproducing, or indirect self-reproducing) for 10%. The combination of classifications allowed us to clarify which lineages of the candidates have been adequately traced and how much we have known about their viability.

Discussion

The whole candidates were classified using the built-in analysis tool and based on the first-division patterns. Viability classification was not full lineage analysis due to some technical limitation. Nevertheless, viability was revealed to the extent that fertility classification revealed infertile, hence non-viable, candidates. The still unclassified candidates are all fertile but have untraced lineage pathway(s). With full lineage analysis, those candidates can be reclassified either as fertile (i.e., having some self-reproducing strain in lineage) or as infertile (i.e., having no self-reproducing strain in lineage). From there, we intend to distinguish viability of the candidates, which indicates not only fertility but also evolutionary potential. These unclassified candidates are not negligible since we do not know a priori how frequently the candidates are fertile or viable. The classification should ideally be automated and systematised, even applicable to different sets of candidates.

Search Limitation and Possible Enhancement

To emphasise the problem situation, the limitation of the current analysis tool surfaces when it hits either cases where (a) both of the two offspring that a parent produces are different from the parent or from each other, or where (b) one of the two offspring is the same as the parent but the other is different.[7] This fact implies that the built-in analysis tool assumes that even though an act of division produces two same offspring, tracing one of them suffices to analyse the lineage. Again, when the built-in analysis tool searches for descendant generations further down, what it recursively incubates for tracking is only one of the two offspring that is arbitrarily labelled as being divided off from the parent.

For example, suppose an organism produces one non-dividing offspring and one reproducing offspring in a deterministic environment without mutation. If the reproducing offspring produces a self-reproducing offspring, the original organism should be classified as (indirect) self-reproducing in that it exhibits the lineage. However, if the analysis tool is set to trace one of the offspring at the first-division, which happens to be non-dividing, then the other offspring, which happens to be reproducing a self-reproducing offspring, is not further traced, and the original organism ends up being classified as "non-viable". This is not a proper viability classification because self-reproducers may be viable, with an evolutionary potential.

In other words, for some candidates which start out producing different two offspring, only a subset of (or rather, a "branch" of) the whole expected lineage becomes revealed through the analysis tool. Therefore, we can neither guarantee their fertility nor viability. Fertility implies reproducibility, whereas viability implies fertility with evolutionary potential, something that, for example, leads to an exponential growth of population (e.g. as pure self-reproducers), as opposed to a linear population growth[8]. Importantly, not only individually self-reproducing strains but also "collectively" self-reproducing strains (i.e., strains that are self-reproducing from a lineage point of view) should be classified as fertile, and even as viable, being evolutionarily po-

[7] The case (a) applies to the pattern $I \neq F \neq C$ and the case (b) applies to the patterns $I = F \neq C$, $I = C \neq F$, and $I \neq F = C$, in the notation introduced earlier.

[8] The *pathological constructor* proposed by Baugh and McMullin (2012) is an example, the strain of which reproduces itself and an infertile strain. This would exhibit the pattern $I = F \neq C$.

tential.[9] All this renders the classification non-trivial.

The automated analysis proceeds with the time limit as mentioned earlier and implicitly with the recursion limit (i.e., how many generations to track down deeper). Aside from these limits, because of the lineage traceability limit discussed above, not all of the candidates have been analysed for viability, although the majority of the candidates have been analysed for fertility.

It is our next, ongoing step to modify the analysis tool with an enhanced ability to fully cover the whole lineage pathways expected from an organism (although of course practically there should be set a division time limit and a recursion limit in automatically revealing lineages).

Conclusions

In the previous research, we constructed and observed a von Neumann style ancestral self-reproducer with a genotype-phenotype mapping subject to evolution within the particular platform. The discovery of the self-reproducer's quick evolutionary degeneration into a self-copier raised a question: How likely is it such degeneration takes place? To answer this question, in the present research, we endeavoured to investigate the spectrum of single-point mutants of the particular self-reproducer in an attempt to classify *viable* candidates for evolvable genotype-phenotype mapping. The presented results are rather preliminary and should serve as a platform to further characterise the prototype self-reproducing in a von Neumann style. In the course, we consequently encountered a new situation where we ask: Given spectrum of mutants, can we classify them by viability?

The automated analysis, nevertheless, yielded an insight as to distinguishable *fertile* candidates, when combined with the classification of the first-division patterns. The fertile candidates (10%) need further scrutiny for reproduction mode and evolutionary potential, while the infertile candidates (80%) need not since they are logically non-viable. At any rate, while the vast majority (90%) of the whole candidates are classified as either fertile or infertile, the rest (10%) of the whole candidates yet remain unclassified.

The existence of those still unclassified candidates clarified the fact that there is a subtlety in the concept of viability. Specifically, the current analysis tool is only capable of revealing a single possible lineage pathway that an organism is expected to exhibit. To recapitulate, it traces only one (arbitrarily pre-determined) of the two offspring at each division, whereas the sub-lineage extending from the other offspring is not further traced. In this sense the analysis is not thorough. It is therefore necessary to automatise and systematise the classification of candidates based on viability, and further, based on the mode of self-reproduction, in the current system. The built-in analysis tool is being enhanced to thoroughly investigate lineage pathways. A better characterisation of the particular example of a von Neumann style self-reproducer requires to understand its mutational pathways from a viewpoint of viability.[10]

The experimental set of the Avida system presented in the current paper, including the ancestral program file and the instruction set configuration file, can be accessed at: `http://alife.rince.ie/evosym/ecal_2013_thbm.zip`.

[9] This distinction appears to be in resonance with that between individual autocatalysis and collective catalysis in artificial chemistry.

[10] Furthermore, to effectively find interesting reproduction modes in viable candidates, it may be practical to reclassify the candidates based on the likelihood of experiencing a change in genotype-phenotype mapping. In the design of the prototype, the regions where mutation most likely can cause any change in genotype-phenotype mapping are Translation Table and Decode Loop, both serving as a part of the programmable constructor A of von Neumann's architecture.

Acknowledgements

This work has been supported by the European Complexity Network (Complexity-NET) through the Irish Research Council for Science and Technology (IRCSET) under the collaborative project EvoSym. We would like to appreciate the original developers of the Avida platform. Furthermore we would also like to acknowledge the three anonymous reviewers for their valuable comments and suggestions to improve the consistency of this paper.

References

Adami, C. (1997). *Introduction to Artificial Life*. Springer, corrected edition.

Baugh, D. and McMullin, B. (2012). The emergence of pathological constructors when implementing the von neumann architecture for self-reproduction in tierra. In Ziemke, T., Balkenius, C., and Hallam, J., editors, *From Animals to Animats 12*, pages 240–248, Berlin. Springer.

Hasegawa, T. and McMullin, B. (2012). Degeneration of a von neumann self-reproducer into a self-copier within the avida world. In Ziemke, T., Balkenius, C., and Hallam, J., editors, *From Animals to Animats 12*, pages 230–239, Berlin. Springer.

McMullin, B. (2000). John von neumann and the evolutionary growth of complexity: Looking backward, looking forward. *Artificial Life*, 6(4):347–361.

McMullin, B. (2012). Architectures for self-reproduction: Abstractions, realisations and a research program. In Adami, C., Bryson, D. M., Ofria, C., and Pennock, R. T., editors, *Artificial Life 13*, pages 83–90. MIT Press.

Ofria, C. and Wilke, C. O. (2004). Avida: A software platform for research in computational evolutionary biology. *Artificial Life*, 10(2):191–229.

von Neumann, J. (1951). The general and logical theory of automata. *Cerebral mechanisms in behavior*, pages 1–41.

von Neumann, J. (1966). *Theory of self-reproducing automata*. University of Illinois Press. Edited and completed by A.W. Burks.