Eric Holloway and Robert Marks "Observation of Unbounded Novelty in Evolutionary Algorithms is Unknowable."
Artificial Intelligence and Soft Computing, pp. 395-404. Springer, Cham, 2018

Observation of Unbounded Novelty in Evolutionary Algorithms is Unknowable

Eric Holloway(✉) and Robert Marks

Department of Electrical and Computer Engineering, Baylor University,
Waco, TX 76706, USA
{Eric_Holloway,Robert_Marks}@baylor.edu

Abstract. Open ended evolution seeks computational structures whereby creation of unbounded diversity and novelty are possible. However, research has run into a problem known as the "novelty plateau" where further creation of novelty is not observed. Using standard algorithmic information theory and *Chaitin's Incompleteness Theorem*, we prove no algorithm can detect unlimited novelty. Therefore observation of unbounded novelty in computer evolutionary programs is nonalgorithmic and, in this sense, unknowable.

1 Artificial Life and Endless Novelty

Evolutionary emergence is inspirational in two fields of computer science: artificial life and evolutionary computing [1]. The goal of the more theoretical field of artificial life is creation of conditions that lead to unbounded novelty explosion seemingly apparent in biology [2–4]. Evolutionary computing attempts to harness these innovative powers to solve engineering problems.

But conditions for unbounded novelty have yet to be discovered. All approaches hit a point where novelty ceases to be produced [5]. Many think the root cause of the "novelty plateau" is the reliance on an objective fitness function which causes evolution to become stuck on local minima [6] or hit the fitness upper bound [7]. Open ended evolution seeks to bypass the plateau by removing the fitness function [8–10]. The fitness function is replaced with a type of swarm intelligence [11–13] where each individual in the swarm makes its own decisions which may or may not cause it to survive. The environment is infused with rich information so the swarm can both grow in novelty and complexity and then contribute its own information to future generations [14,15].

Finding the necessary and sufficient conditions to produce boundless novelty is a goal of current artificial life research [6,16,17]. It is impossible to observe an artificial life simulation forever to know whether it can produce boundless novelty. To claim a simulation produces boundless novelty, the claim must be proven from initial conditions. Establishing such "conditions" is equivalent to foundational "axioms" in the development of mathematical disciplines [18,19]. These unbounded evolutionary axioms are then considered to be the basis for a proof or demonstration of ever increasing unbounded novelty.

L. Rutkowski et al. (Eds.): ICAISC 2018, LNAI 10841, pp. 395–404, 2018.
https://doi.org/10.1007/978-3-319-91253-0_37

Insofar as the conditions are the same as axioms, we prove axioms for unbounded open evolution do not exist in the sense that ever increasing novelty cannot be observed. The proof assumes there are no inconsistencies, such as false positives and false negatives, in the observation process. Even in the absence of inconsistencies, the bound on novelty for a set of axioms can still be very high. The observable novelty plateau for a set of conditions can thus be highly elevated – but not unbounded.

1.1 Identifying Unlimited Novelty

Computational open ended evolution assumes that bitstrings of ever increasing novelty are computable. Indeed, bitstrings of increasing novelty will be contained in the trivial enumeration:

Listing 1.1. An algorithm that generates an endless amount of novel bitstrings, to illustrate endless novelty production is easily accomplished.

```
for i in range(∞): print binary(i)
```

Listing all binary bitstrings will generate all novel bitstrings. Lacking is a method to identify which bitstrings contain increasing novelty. Doing so for an unlimited number of novel bitstrings is not possible due to *Chaitin's incompleteness theorem* [20,21].

2 Prefix Free Complexity and Algorithmic Information

Algorithmic information theory [20,22–25], the study of the mathematics of algorithms, is a useful tool for insight into artificial life and evolutionary computing [8,26–29]. We here apply it to the analysis of open ended evolution.

Kolmogorov complexity [20] is the length of the shortest program y^* that generates a bitstring x when executed on a universal Turing machine, $\mathcal{U}$,

$$K(x) = \min_{y:\, \mathcal{U}(y)=x} \ell(y). \tag{1}$$

where $\ell(y)$ is the length, typically measured in bits, of the program y. Minimization is over all y programs to find the shortest program y^* that outputs x so that

$$K(x) = \ell(y^*).$$

Chaitin refers to y^* as the *elegant program* for x [30]. We use prefix free (a.k.a. self-delimiting) code [31,32] to simplify analysis. Without prefix free Kolmogorov complexity, the chain rule of Kolmogorov complexity is [33]

$$K(a,b) = K(a|b) + K(b) + O(\log K(a,b)). \tag{2}$$

We can understand the source of the logarithmic term in (2) by analyzing the set generating program p, which generates the set $\{a,b\}$ from the elegant program

for b (denoted b^*) and the elegant program that generates a given b (denoted a^*_b). To generate $\{a, b\}$ from the elegant programs a^*_b and b^*, the set generator program p will need to know the size of the length of the smaller elegant program. Since size of the length of the smaller elegant program is at most a logarithm of the length, the logarithmic term is added to (2).

Ming and Vitányi show [33] that if we use $K(a|b, K(b))$ with prefix free coding, then the logarithmic accuracy term becomes a constant,

$$K(a, b) \;=\; K(a|b, K(b)) + K(b) + c.$$

The result can be further simplified by observing that an elegant program for b gives us both b and $K(b)$. Knowing $K(b)$ we find the first short program b^* that generates b and $K(b)$ by running all programs of length $K(b)$. The shortest program that halts and outputs b is b^*, and we know this procedure will halt since we know $K(b)$. We can therefore replace $\{b, K(b)\}$ with b^*. The complexity expression where this is done is denoted

$$Kc(a|b) \;:=\; K(a|b^*).$$

2.1 Defining the Set Generating Program

The constant c in

$$Kc(a, b) \;=\; Kc(a|b) + Kc(b) + c \tag{3}$$

is the size of the program p_c needed to generate the set $\{a, b\}$ from elegant programs b^* and the elegant program that generates a given b^* (denoted $a^*_{b^*}$). We rewrite (3) using the notation[1]

$$Kc(a, b) \underset{c}{=} Kc(a|b) + Kc(b). \tag{4}$$

To understand how the set generation program p_c works, assume we have the elegant program b^* for b, and the elegant program $a^*_{b^*}$ that generates a when given b^* as input. If b^* does not allow the construction of a program $a^*_{b^*}$ that is shorter than the elegant program a^* for a, then $a^*_{b^*}$ is a^*.

The concatenation of b^*, $a^*_{b^*}$ and stopcode s gives the input[2] $i = b^* a^*_{b^*} s$ for the calculation program p_c. The set generation program p_c's operation is as follows.

---

[1] Some authors use the notation "$\overset{+}{=}$" in lieu of "$\underset{c}{=}$" [31,34].

[2] Even though i is not prefix free (i.e. it will halt once b^* is executed, leaving $a^*_{b^*}$ unread), the program that is run on the Turing machine $\mathcal{U}$ is still prefix free because i is appended to p_c, so the full execution on $\mathcal{U}$ is $\mathcal{U}(p_c i)$. Since p_c is prefix free, then so is $p_c i$, as it will only halt once the entire string is read.

1. p_c executes i on a Turing machine $\mathcal{U}$ until $\mathcal{U}(i)$ halts. Since b^* is prefix free, that means $\mathcal{U}(i)$ has output b.
2. The remaining portion of i is run with the previous portion of i as input, which equates to running $a^*_{b^*}$ with b^* as input, which produces a.
3. Once the stopcode s is encountered, p_c collects b and a from the multi-part execution of i, and outputs the set $\{a, b\}$.

2.2 Kolmogorov Complexity is not Computable

No program can tell us what the Kolmogorov complexity is for every bitstring [30,35]. That there is no such program can be easily shown using proof by contradiction.

Assume that there is such a program, the True Kolmogorov Complexity Printer (TKCP). We can then use it in the program in Listing 1.2 to produce a contradiction.

Listing 1.2. Code showing a Klomogorov Complexity printer results in a contradiction. `len(self)` means the length of this program, including all code for the subfunctions such as TKCP.

```
def contradiction():
    for i in range(∞):
        bs = binary(i)
        if TKCP(bs) > len(self):
            return bs
```

The `contradiction` program iterates through all integers until it finds one that has a binary encoding with a greater Kolmogorov complexity than the size of `contradiction` (which includes the code for all subfunctions such as `TKCP`). Once the program finds such an integer, it outputs the binary encoding of this integer bs^*. Thus, we have a program with a smaller size than the Kolmogorov complexity for bs^* which nevertheless outputs bs^*. Yet, the definition of Kolmogorov complexity is the length of the shortest program that outputs bs^*. This is a contradiction. Therefore, TKCP cannot exist.

2.3 Chaitin's Incompleteness Theorem

Chaitin's incompleteness theorem is based on a similar argument but deals with axioms and proofs instead of programs. The theorem similarly shows an axiomatic system cannot prove the Kolmogorov complexity of a bitstring above a certain limit, and that this limit is dependent on the size of the axioms plus a constant for the length of the proof assembling program.

Listing 1.3. A program demonstrating there is a limit to proving lower bounds on Kolmogorov complexity. `len(self)` means the length of this program, including all code for the subfunctions.

```
def contradiction(axioms):
    L = len(self) + len(axioms)
    for i in range(∞):
        bs = binary(i)
        goal = "K("+bs+")>"+str(L)
        for proof in all_proofs(axioms):
            if proves(proof, goal):
                return bs
```

Listing 1.3 shows again we end up with a contradiction, since a program that is shorter than the Kolmogorov complexity of bs^* generates bs^*. The difference between this listing and the previous is the function accepts a set of axioms it can use to prove a lower bound on the bitstring's Kolmogorov complexity.

3 Limits on Identifying Novelty

3.1 Definitions of Novelty

There are many different approaches to defining novelty. In a biological setting, novelty refers to new aspects in an organism that not homogonous or homologous with ancestral organisms [36] and fulfill unique functions [37]. The field of information retrieval defines novelty as a new information nugget in a user's interest set that is also contained within the document set [38] and sentences that contain information not contained in previous sentences [39]. In the field of computer generated art, novelty is defined using Dorin and Korb's definition [40] which characterizes a system S_2 that can reliably generate patterns that cannot be created by another system S_1, where S_1 is the existing worldview of an audience.

Within the fields of anomaly [41], fault [42], and outlier detection [43], the systems are generally trained or designed to recognize the typical behavior, and flag atypical behavior. Atypical behavior is not necessarily novel, since it may be well understood as in fault detection where a variety of failure states are derived ahead of time. Novelty is atypical behavior that is unanticipated, and these disciplines provide different techniques for defining the typical region, ranging from rule based, to statistical to complex nonlinear regression models such as neural networks [44–46].

A digital organism can be measured as novel with reference to an existing population. Novelty is measured by a distance function from the rest of the population where the distance function is domain specific [47]. Selecting for novelty is also used in evolutionary computation [48]. In the case of digital life and evolutionary computation, the typical population changes as the algorithm progresses, but there is still the desire to find individuals that do not fit the latest typical population.

3.2 Commonalities of Novelty

In all these domains, novelty has common characteristics. Novelty is

- defined relative to a typical population,
- measured using a distance, and
- unanticipated.

Within a computational domain, everything can be represented by a bitstring. We can measure a bitstring's distance from a population in many ways. A large distance signifies something outside of the typical population. We can characterize the typical population by the smallest program that can produce the population, the size of which is the Kolmogorov complexity of the population.[3] When a member is added to the population that requires the program size to increase, then the member is atypical albeit not necessarily novel. Note if a member is novel, then it is atypical, thus it will increase the population's Kolmogorov complexity.

3.3 Necessary Condition for Novelty

We can measure atypical information in bitstring b_N by the conditional Kolmogorov complexity in reference to the smallest program that generates the current population $\{b_1, b_2, \ldots, b_{N-1}\}$. If the bitstring b_N contains new information then the conditional complexity is positive.

$$Kc(b_N|b_1, b_2, \ldots, b_{N-1}) > 0. \tag{5}$$

Since Kolmogorov complexity corresponds to the length of a program in bits, its measure is restricted to positive integers. Therefore (5) can equivalently be written as

$$Kc(b_N|b_1, b_2, \ldots, b_{N-1}) \geq 1. \tag{6}$$

Randomness also meets this definition, which makes (6) a necessary but not sufficient definition for novelty. For the purpose of this proof, however, a necessary condition is all that is needed.

The distinction between necessary and sufficient conditions is an important qualification, since many bitstrings are random, and their addition will most likely increase the $Kc(\cdot)$ without increasing the true novelty in the population. It may even be true that every new bitstring added to a population increases $Kc(\cdot)$. In either case, whether the new bitstring is entirely random or if it contains true novelty, $Kc(\cdot)$ will be increased. Increasing $Kc(\cdot)$ alone is therefore not sufficient to indicate the addition of novelty. However, we will see that regardless of the source of $Kc(\cdot)$ increase, randomness or novelty, the fact that $Kc(\cdot)$ increases when novelty is added makes it impossible to detect novelty beyond a limit.

[3] While Kolmogorov complexity is an exact metric, and has to account for both meaningful structure and random noise in the population, the Kolmogorov sufficient statistic can be used to measure just the meaningful structure in the population.

3.4 Unbounded Novelty Detection is Nonalgorithmic

Assume there exists a set of axioms that can identify novelty when novelty is added to a population. When a new bitstring is generated, we use the axioms to derive a proof that the new bitstring is novel. Note these axioms are not directly calculating the conditional Kolmogorov complexity of the new bitstring. The axioms are only being used to prove the new bitstring b_N is novel with respect to prior observations. However, based on the argument in Sect. 3.3, proving b_N is novel indirectly entails (6), namely that the conditional complexity of b_N is positive.

If we have a set of axioms that always identifies novelty added to the population, then when novelty is added, (6) states the conditional complexity of b_N is at least one bit. With this knowledge, we can use the Kolmogorov complexity chain rule [33], as defined in (4), to estimate a lower bound on the population's joint Kolmogorov complexity, substituting 1 whenever we detect novelty and 0 otherwise. Assuming we've identified M instances of novelty during the generation of N bitstrings, we can lower bound the joint complexity using (6).

$$\begin{aligned} Kc(b_1, b_2, \ldots, b_N) &\underset{c}{=} Kc(b_1) + Kc(b_2|b_1) + \ldots \\ &\quad + Kc(b_N|b_1, b_2, \ldots, b_{N-1}) \\ &\underset{c}{\geq} M. \end{aligned} \tag{7}$$

Since c is the size of the set generating program p_c, as detailed in Sect. 2.1, and thus is positive, then by removing c we can make the inequality exact

$$Kc(b_1, b_2, \ldots, b_N) \geq M. \tag{8}$$

As proven in Sect. 2.3, Chaitin's incompleteness theorem states we cannot prove $K(b) > \mathcal{L}$ for some $\mathcal{L}$ that is based on the proof axioms. However, if we can detect unbounded novelty, then M in (8) becomes arbitrarily big, and at some point we can prove $Kc(b_1, b_2, \ldots, b_N) \geq M > \mathcal{L}$, which is a contradiction.

Therefore, we can only detect novelty a finite number of times. Furthermore, this limit is not much larger than the size of the axioms as can be inferred from the foundation of Chaitin's incompleteness theorem.

Additionally, this argument works for novelty generating algorithms. If we have an algorithm that we know always creates novel bitstrings, then the conditional complexity in (6) is always greater than zero, and eventually the algorithm will create an M that is greater than $\mathcal{L}$ thereby contradicting Chaitin's incompleteness theorem.

4 Conclusion

The great novelty and diversity resulting from biological evolution suggests there is an algorithm that can produce the same. Finding this algorithm is the goal of evolutionary computation and artificial life research. Yet, it is not sufficient to produce endless novelty, but also identify novelty when it occurs.

However, a reliable method of identifying an endless amount of novelty would also imply the ability to calculate a lower bound of arbitrary size on Kolmogorov complexity. Since every axiomatic system has a limit to the lower bound it can calculate, due to Chaitin's incompleteness theorem, the reliable method of novelty detecting introduces a contradiction. This same contradiction results if we have a program we know only generates novel bitstrings.

As such, we must conclude there is no reliable method of identifying an endless degree of novelty, nor of producing only novel bitstrings. We can only reliably detect novelty to a finite amount, and not significantly more than the Kolmogorov complexity of the axioms used for detection.

The bound on novelty observation can be high, so our analysis does not specifically preclude observation of novelty in the evolutionary process - but does prove there are limitations. There is also the assumption of consistency. If the novelty requirement is relaxed to also allow inconsistencies such as the occurrence of false positives and false negatives, then observing endless novelty might still be possible such as labeling all bitstrings generated by the algorithm in Listing 1.1 as "novel".

References

1. Mitchell, M., Forrest, S.: Genetic algorithms and artificial life. Artif. life **1**(3), 267–289 (1994)
2. Huneman, P.: Determinism, predictability and open-ended evolution: lessons from computational emergence. Synthese **185**(2), 195–214 (2012)
3. Komosinski, M., Rotaru-Varga, A.: From directed to open-ended evolution in a complex simulation model. Artif. Life **7**, 293–299 (2000)
4. Sayama, H.: Seeking open-ended evolution in swarm chemistry. In: 2011 IEEE Symposium on Artificial Life (ALIFE), pp. 186–193. IEEE (2011)
5. Li, J., Storie, J., Clune, J.: Encouraging creative thinking in robots improves their ability to solve challenging problems. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 193–200. ACM (2014)
6. Soros, L., Stanley, K.O.: Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria. Artif. life **14**, 793–800 (2014)
7. Basener, W.F.: Exploring the concept of open-ended evolution. In: Biological Information: New Perspectives, pp. 87–104. World Scientific (2012)
8. Bedau, M.A., McCaskill, J.S., Packard, N.H., Rasmussen, S., Adami, C., Green, D.G., Ikegami, T., Kaneko, K., Ray, T.S.: Open problems in artificial life. Artif. life **6**(4), 363–376 (2000)
9. Ruiz-Mirazo, K., Peretó, J., Moreno, A.: A universal definition of life: autonomy and open-ended evolution. Orig. Life Evol. Biosph. **34**(3), 323–346 (2004)
10. Ruiz-Mirazo, K., Umerez, J., Moreno, A.: Enabling conditions for open-ended evolution. Biol. Philos. **23**(1), 67–85 (2008)
11. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: from Natural to Artificial Systems, vol. 1. Oxford University Press, Oxford (1999)
12. Ewert, W., Marks, R.J., Thompson, B.B., Yu, A.: Evolutionary inversion of swarm emergence using disjunctive combs control. IEEE Trans. Syst. Man Cybern. Syst. **43**(5), 1063–1076 (2013)

13. Roach, J., Ewert, W., Marks, R.J., Thompson, B.B.: Unexpected emergent behaviors from elementary swarms. In: 2013 45th Southeastern Symposium on System theory (SSST), pp. 41–50. IEEE (2013)
14. Roach, J.H., Marks, R.J., Thompson, B.B.: Recovery from sensor failure in an evolving multiobjective swarm. IEEE Trans. Syst. Man Cybern. Syst. **45**(1), 170–174 (2015)
15. Taylor, T.: Exploring the concept of open-ended evolution. In: Proceedings of the 13th International Conference on Artificial life, pp. 540–541 (2012)
16. Jakobi, N.: Encoding scheme issues for open-ended artificial evolution. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 52–61. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61723-X_969
17. Channon, A.: Three evolvability requirements for open-ended evolution. In: Artificial Life VII Workshop Proceedings, Portland, OR, pp. 39–40 (2000)
18. Mueller, I.: Euclid's elements and the axiomatic method. Br. J. Philos. Sci. **20**(4), 289–309 (1969)
19. Shenoy, P.P., Shafer, G.: Axioms for probability and belief-function propagation. In: Yager, R.R., Liu, L. (eds.) Classic Works of the Dempster-Shafer Theory of Belief Functions. STUDFUZZ, vol. 219, pp. 499–528. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-44792-4_20
20. Chaitin, G.J.: Algorithmic information theory. IBM J. Res. Dev. **21**(4), 350–359 (1977)
21. Raatikainen, P.: On interpreting Chaitin's incompleteness theorem. J. Philos. Log. **27**(6), 569–586 (1998)
22. Chaitin, G.J.: Information, Randomness & Incompleteness: Papers on Algorithmic Information Theory, vol. 8. World Scientific, Singapore (1990)
23. Grünwald, P.D., Vitányi, P.M., et al.: Algorithmic information theory. In: Handbook of the Philosophy of Information, pp. 281–320 (2008)
24. Seibt, P.: Algorithmic Information Theory. Springer, Heidelberg (2006). https://doi.org/10.1007/978-3-540-33219-0
25. Van Lambalgen, M.: Algorithmic information theory. J. Symb. Log. **54**(4), 1389–1400 (1989)
26. Chaitin, G.: Proving Darwin: Making Biology Mathematical. Vintage, New York (2012)
27. Chaitin, G.J.: Toward a mathematical definition of life. In: Information, Randomness & Incompleteness: Papers on Algorithmic Information Theory, pp. 86–104. World Scientific (1987)
28. Gecow, A.: The purposeful information. On the difference between natural and artificial life. Dialogue Univers. **18**(11/12), 191–206 (2008)
29. Pattee, H.H.: Artificial life needs a real epistemology. In: Morán, F., Moreno, A., Merelo, J.J., Chacón, P. (eds.) ECAL 1995. LNCS, vol. 929, pp. 21–38. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59496-5_286
30. Chaitin, G.J.: The Unknowable. Springer Science & Business Media, Heidelberg (1999)
31. Bennett, C.H., Gács, P., Li, M., Vitanyi, P., Zurek, W.H.: Information distance. arXiv preprint arXiv:1006.3520 (2010)
32. Calude, C.S.: Algorithmic randomness, quantum physics, and incompleteness. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 1–17. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31834-7_1
33. Ming, L., Vitányi, P.M.: Kolmogorov complexity and its applications. Algorithms Complex. **1**, 187 (2014)

34. Vitányi, P.M., Li, M.: Minimum description length induction, Bayesianism, and Kolmogorov complexity. IEEE Trans. Inf. Theory **46**(2), 446–464 (2000)
35. Wallace, C.S., Dowe, D.L.: Minimum message length and Kolmogorov complexity. Comput. J. **42**(4), 270–283 (1999)
36. Muller, G.B., Wagner, G.P.: Novelty in evolution: restructuring the concept. Ann. Rev. Ecol. Syst. **22**(1), 229–256 (1991)
37. Pigliucci, M.: What, if anything, is an evolutionary novelty? Philos. Sci. **75**(5), 887–898 (2008)
38. Li, X., Croft, W.B.: An information-pattern-based approach to novelty detection. Inf. Process. Manag. **44**(3), 1159–1188 (2008)
39. Zhao, L., Zhang, M., Ma, S.: The nature of novelty detection. Inf. Retr. **9**(5), 521–541 (2006)
40. Kowaliw, T., Dorin, A., McCormack, J.: An empirical exploration of a definition of creative novelty for generative art. In: Korb, K., Randall, M., Hendtlass, T. (eds.) ACAL 2009. LNCS (LNAI), vol. 5865, pp. 1–10. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10427-5_1
41. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. (CSUR) **41**(3), 15 (2009)
42. Venkatasubramanian, V., Rengaswamy, R., Yin, K., Kavuri, S.N.: A review of process fault detection and diagnosis: part I: quantitative model-based methods. Comput. Chem. Eng. **27**(3), 293–311 (2003)
43. Hodge, V., Austin, J.: A survey of outlier detection methodologies. Artif. Intell. Rev. **22**(2), 85–126 (2004)
44. Pimentel, M.A., Clifton, D.A., Clifton, L., Tarassenko, L.: A review of novelty detection. Sig. Process. **99**, 215–249 (2014)
45. Reed, R., Marks, R.J.: Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. MIT Press, Cambridge (1999)
46. Thompson, B.B., Marks, R.J., Choi, J.J., El-Sharkawi, M.A., Huang, M.Y., Bunje, C.: Implicit learning in autoencoder novelty assessment. In: 2002 Proceedings of the 2002 International Joint Conference on Neural Networks, IJCNN 2002, vol. 3, pp. 2878–2883. IEEE (2002)
47. Lehman, J., Stanley, K.O.: Abandoning objectives: evolution through the search for novelty alone. Evol. Comput. **19**(2), 189–223 (2011)
48. Mouret, J.B.: Novelty-based multiobjectivization. In: Doncieux, S., Bredèche, N., Mouret, J.B. (eds.) New Horizons in Evolutionary Robotics. SCI, vol. 341, pp. 139–154. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18272-3_10