Flood Evolution: Changing the Evolutionary Substrate from a Path of Stepping Stones to a Field of Rocks

David Shorten
University of Cape Town
Private Bag X3
Rondebosch 7701
dshorten@cs.uct.ac.za

Geoff Nitschke
University of Cape Town
Private Bag X3
Rondebosch 7701
gnitschke@cs.uct.ac.za

ABSTRACT

We present ongoing research that is an extension of novelty search, flood evolution. This technique aims to improve evolutionary algorithms by presenting them with large sets of problems, as opposed to individual ones. If the older approach of incremental evolution were analogous to moving over a path of stepping stones, then this approach is similar to navigating a rocky field. The method is discussed and preliminary results are presented.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Novelty Search

1. INTRODUCTION

This is a position paper detailing a current research topic and results of a case study are discussed. This topic is *flood evolution*, an extension to *novelty search*.

Assume one is trying to navigate across a field, from point A to point B, by only setting foot on rocks. If one is traveling along a path of stepping stones, one's journey will be terminated if the gap between any two stones is too large to be crossed. However, if one is moving across a field strewn with rocks and boulders, then in the instance that one reaches a rock from which no unvisited rocks are within the range of a single jump, one can backtrack and attempt to find an alternate route to the desired point.

There exists very little knowledge concerning what determines the difficulty, from the perspective of evolution, of jumping from one stone to another. This paper argues that, given this, it is beneficial for *Evolutionary Algorithms* (EAs) to be presented with a large field of stones through which they can determine their own path.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-2881-4/14/07$15.00.
http://dx.doi.org/10.1145/2598394.2605675.

It is necessary to point out that the usage of the stepping stone metaphor in this paper is subtly different to that in important related literature (eg: [4, 10]). In that context it is referring to an intermediate phenotype which allows for the evolution of a more complicated one. However, here it is referring to the act of providing an incentive towards the development of such a phenotype. Although intermediate phenotypes can emerge for reasons other than selective pressure [13], rewarding a phenotype requires a definition of it. Thus, the reward of a phenotype and the phenotype itself are two inextricably linked concepts. It is for this reason that it is justifiable to use the stepping stone metaphor in this different, but closely related, context. However, to avoid confusion, this concept shall be referred to as a *stepping stone incentive* for the rest of this paper.

2. THE GRAPH OF PROBLEMS

A good way of making the field of rocks metaphor more precise is to employ the mathematical abstraction of a graph. A stepping stone incentive can be viewed as a synonym for a problem which is presented to an EA. *Flood evolution* presents EAs with sets of problems. A set of problems, Π, can be represented as a graph $G = (E, V)$. Each problem $P \in \Pi$ can have an associated node $p \in V$. Given $P, Q \in \Pi$ and their corresponding nodes $p, q \in V$, then the edge $(p, q) \in E$ can be defined as existing if and only if, for a given a population of solutions γ, which contains a solution to P, it is easy to evolve a solution to Q using γ as the initial population. This definition of an edge is analogous to an evolutionary jump between stones. It is, however, imprecise, due to the stochastic nature of EAs [5].

A given graph representing a set of problems may have connected subgraphs. Assume one wants a solution to the problem B. Assume also that one has a set of problems Π and that $B \in \Pi$. Further assume that Π is constructed such that it is feasible to randomly instantiate a population of solutions γ such that γ contains a solution to a problem in Π. This problem can be labeled A, and in the graph representation of Π, the nodes representing A and B can be labeled a and b, respectively. If a and b are part of the same connected subgraph then one could apply an evolutionary algorithm which continuously attempts to jump from all solved problems to all others. This algorithm is guaranteed to find the solution to b and is an application of flood fill [8]. This approach shall be referred to as *flood evolution*.

It is intuitively likely that, the larger a given problem set, the more edges each node in the associated graph will have. Moreover, increasing the size of the problem set should increase the probability that a randomly instantiated popu-

lation solves a problem within it. From this it follows that increasing the size of a problem set increases the chance that a randomly instantiated population will be able to evolve a solution to a desired problem within it.

It is necessary to clarify what exactly is meant by two different problems. In this context, two problems are considered different unless they are identical in every way. So, for instance, two pole balancing problems [7] would be considered different if they had slightly different initial conditions.

3. BACKGROUND

Current theories in Biology state that the evolution of complicated features requires many incremental stepping stones [4, 3]. Moreover, the evidence for this viewpoint is overwhelming [12]. Although we have some understanding of the stepping stones which led to complicated life on Earth, our knowledge of them is sparse [18, 2].

There exists much exploration of the use of sequential, linear, stepping stone incentives in the EA literature [6, 15, 16, 17, 20]. This approach has often been termed *incremental evolution*. It has been shown that this approach allows certain EAs to solve problems which they would have otherwise been unable to.

The ideas presented here are largely inspired by the *novelty search* of Lehman and Stanley [10]. The authors view the work described in this paper as an extension of the ideas contained therein. In this paradigm the authors replace the more traditional fitness function, which rewards solutions based on how close they are to a single goal, with one which rewards solutions based on how novel they are. The consequence of this is that there is an evolutionary incentive for solving many problems closely related to the target problem.

Novelty search has been demonstrated to be a powerful method [10, 11, 14]. This paper proposes that a reason for this success is that it is closer to the rocky field metaphor. More specifically, it has two key differences from the older approach of incremental evolution:

1. There is a larger variety of intermediate problems.
2. The path through these intermediate problems is unspecified; evolution is allowed to figure out its own path.

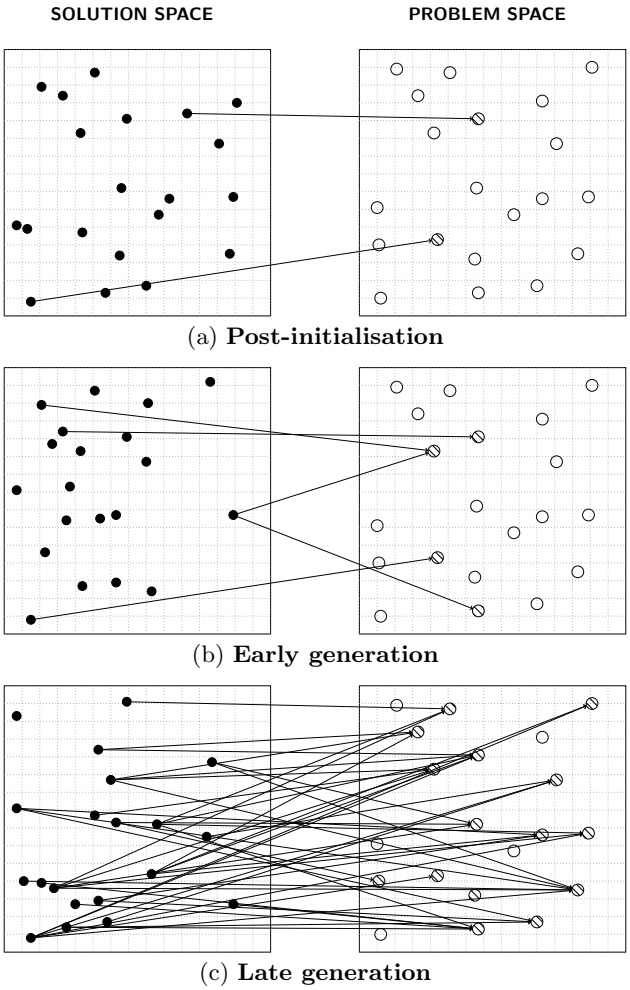
The authors interpret the success of novelty search as evidence towards the plausibility of flood evolution being a useful technique.

Similar ideas to what is being proposed here have been explored within the framework of *artificial life*. Lenski et al [12] ran simulations which rewarded genomic programs for executing one of nine logical operations. The reward received for executing them was proportional to the complexity of the operation. They were able to demonstrate that the evolution of complex operations requires simpler operations as intermediate goals. Arthur and Polak demonstrated a similar result with regards to the evolution of technology [1]. These results are interpreted as further evidence for the utility of providing evolution with a non-linear substrate of incentives.

4. FLOOD EVOLUTION

Similar to novelty search, flood evolution is an augmenting technique which can be applied to a given evolutionary algorithm.

At the start of evolution, a population of candidate solutions must be initialized. At the same time, a set of problems must also be instantiated (see section 5 for further discussion). Each individual candidate solution is always evaluated on all the problems in the in the problem set. A solution receives fitness for each problem which it solves. However, in order to provide an incentive for unsolved problems in the set to be solved, fitness sharing is implemented. In the present preliminary experiments, a solution receives fitness β/n^2, where β is a constant and n is the number of solutions which solve that particular problem. A solution's total fitness is the sum of all the fitness it receives from all the problems which it solves. The manner in which the authors foresee this strategy running is displayed in figure 1. These predictions are already partially confirmed (see section 6).



(a) **Post-initialisation**

(b) **Early generation**

(c) **Late generation**

Figure 1: The change in state of a hypothetical but representative evolutionary run over a problem space. The squares in the left column represent the solution space squashed into two dimensions. Similarly, the right squares represent the problem space. Nodes which are filled represent points in the solution space. Nodes which are empty represent unsolved points in the problem space and nodes with hatching represent solved points. An edge from a solution node to a problem node signifies that the solution solves the problem.

5. PROBLEM SETS

In order to implement flood evolution, it is necessary to have a method by which very large sets of problems can be easily instatiated. This paper proposes two case studies, the second of which is already partially implemented. Both use *Artificial Neural Networks* (ANNs) as their solutions. As such, this research falls under the umbrella of *Neuro-Evolution* (NE). A question which these case studies intend to address is the effect of having mixed problem sets which contain multiple problem types.

5.1 Maze-Solving

Maze solving is already an established testing environment for NE algorithms [10]. Furthermore, there exists freely-available software which implements NE over a maze-solving environment[1]. There also exist many algorithms for automatically constructing mazes [19]. This means that the construction of a problem field which consists of many mazes should be an easy task.

5.2 Polynomial-Interval-Finding

A polynomial-interval-finding problem set, Π, will have three variables constant across it. These are the scaling factor, s (see below), the numbers of inputs and outputs, m and n respectively, which all solution ANNs will have and the number of iterations t which each ANN will be evaluated over. Each $P \in \Pi$ is specified by a set of inputs H, a set of coefficients C, a set of exponents E as well as an interval on the real number line, described by a lower bound l and an upper bound u. The notation $P = (H, C, E, l, u)$ could be used. Necessary specifications are imposed on the sizes of H, C and E. More specifically, $|H| = tm$ and $|C| = |E| = tn$. An ANN is evaluated on a problem as follows. The first m inputs from H are supplied to the ANN, that is, the ANN receives the input vector $\langle h_1, h_2, \ldots, h_m \rangle$ where $h_a \in H$, $1 \geq a \leq mt$. The ANN will then produce the output vector $\langle o_1, o_2, \ldots, o_n \rangle$. In the next iteration the input vector will be $\langle h_{m+1}, h_{m+2}, \ldots, h_{2m} \rangle$ and the output vector will be $\langle o_{n+1}, o_{n+2}, \ldots, o_{2n} \rangle$. This continues until, in the t^{th} iteration, the input vector is $\langle h_{(t-1)m+1}, h_{(t-1)m+2}, \ldots, h_{tm} \rangle$ and, similarly, the output vector is $\langle o_{(t-1)n+1}, o_{(t-1)n+2}, \ldots, o_{tn} \rangle$. A result r can then be calculated as specified in equation (1).

$$r = \sum_{i=1}^{tn} c_i \left(so_i\right)^{e_i} \tag{1}$$

The problem is considered solved if $u < r < l$.

The motivation for this case study is that it is easy to implement and testing an ANN on it uses comparatively little computational resources.

6. SAMPLE RUN

An implementation of the polynomial-interval-finding problem set was made and incorporated into Joel Lehman's *Novelty Search C++* [1] as described in [9]. A few things to note regarding this incorporation:

1. The implementation of fitness search [10], as opposed to novelty search, was used.
2. The parameter values used for the rtNEAT implementation included in this package were the default ones. This was with the exception of the population size, which was increased from 250 to 1000.
3. All solutions were evaluated on all problems each generation.
4. In the original version of the package, each generation, one solution was added and one removed from the population. This number was changed to 100.
5. The parameters pertaining to the problem set are shown in table 1. These parameters were used as preliminary experiments showed that they allowed for the near complete solution of the set within a relatively small amount of wall time.

Table 1: Parameters used in creation of the polynomial-interval-finding problem set

set size	1000
l (lower bound)	uniform random numbers in range [-100, 100]
u (upper bound)	$l + 2.0$
t (iterations)	10
n (number inputs)	11
m (number outputs)	2
s (scaling factor)	10
I (input set)	uniform random numbers in range [0, 1]
C (coefficients set)	uniform random numbers in range [-10, 10]
E (exponents set)	random numbers from set {0, 1, 2, 3, 4}

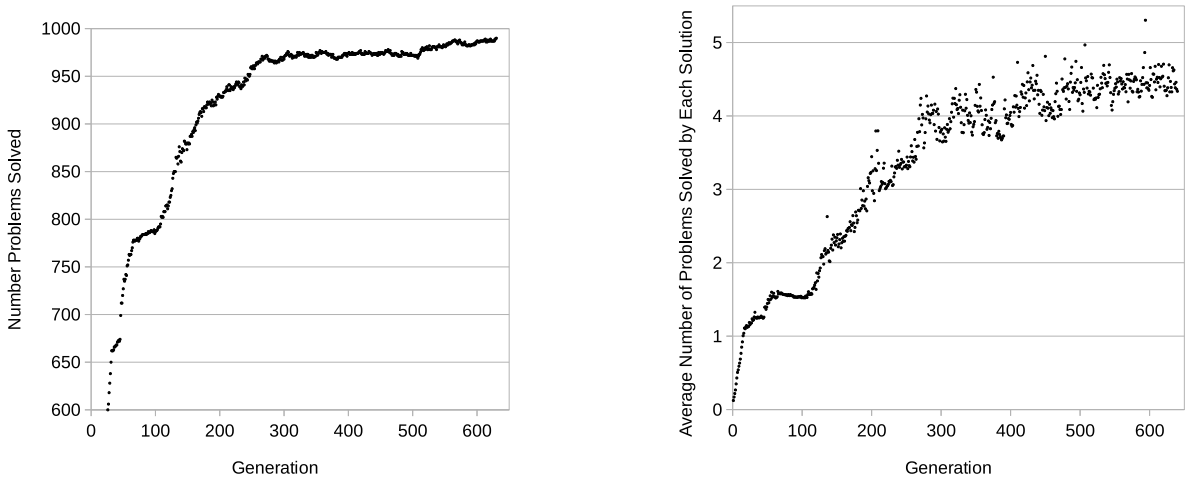
The implementation was run for 640 generations. Figure 2 displays two measures of the change in state of the algorithm during these generations. A number of interesting features can be observed. The first is that, despite the fact that the total number of problems solved did start to plateau around generation 300, progress was still being made right until the end of the 640 generations (figure 2(a)). Before generation 500, the largest number of problems solved in any generation was 978. The number of problems solved in generation 640 was 989. The second interesting feature is that *general* solutions were being found (figure 2(b)). In later generations, on average, solutions were solving more than four problems each. From generation 287 until the end of the run, there was at least one solution which was solving 15 or more problems.

The authors are aware that it is not possible to draw solid conclusions from a single run. These are, however, early, preliminary results and a more rigorous analysis is presently underway. Furthermore, other experimental runs produced similar results.

7. FUTURE RESEARCH

At present, a more detailed and rigorous analysis of the performance of flood evolution on polynomial-interval-finding problems is underway. An important goal is to establish whether flood evolution can solve problems which other NE techniques are unable to solve. We also intend to implement and test maze-solving problem sets. Another avenue of exploration which the authors are considering exploring is the coevolution of the problem set. Problems could be represented by genotypes and rewarded for producing more complicated individuals in the solution population.

[1] `http://eplex.cs.ucf.edu/software/NoveltySearchC++.zip`



(a) **Saturation of the problem space**

(b) **Generality of solutions**

Figure 2: Graphs showing the interaction between the problem and solution spaces during a run of flood evolution.

8. CONCLUSIONS

It is concluded that flood evolution is a promising area of research which has the potential to improve the power of NE algorithms as well as aid our understanding of the underlying mechanisms of evolution.

9. REFERENCES

[1] W. B. Arthur and W. Polak. The evolution of technology within a simple computer model. *Complexity*, 11(5):23–31, 2006.

[2] J. W. O. Ballard and M. C. Whitlock. The incomplete natural history of mitochondria. *Molecular ecology*, 13(4):729–744, 2004.

[3] C. Darwin and W. F. Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt, 2009.

[4] R. Dawkins. *The selfish gene*. Oxford university press, 2006.

[5] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. springer, 2003.

[6] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[7] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *IJCAI*, volume 99, pages 1356–1361, 1999.

[8] C. Y. Lee. An algorithm for path connections and its applications. *Electronic Computers, IRE Transactions on*, (3):346–365, 1961.

[9] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI)*, Cambridge, MA, 2008. MIT Press.

[10] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[11] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218. ACM, 2011.

[12] R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami. The evolutionary origin of complex features. *Nature*, 423(6936):139–144, 2003.

[13] M. Lynch. The frailty of adaptive hypotheses for the origins of organismal complexity. *Proceedings of the National Academy of Sciences*, 104(suppl 1):8597–8604, 2007.

[14] S. Risi, C. E. Hughes, and K. O. Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491, 2010.

[15] N. Saravanan and D. B. Fogel. Evolving neural control systems. *IEEE Intelligent Systems*, 10(3):23–27, 1995.

[16] A. P. Wieland. Evolving controls for unstable systems. In *Connectionist models: proceedings of the 1990 summer school*, pages 91–102, 1990.

[17] A. P. Wieland. Evolving neural network controllers for unstable systems. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 667–673. IEEE, 1991.

[18] J. J. Wiens. Missing data, incomplete taxa, and phylogenetic accuracy. *Systematic Biology*, 52(4):528–538, 2003.

[19] J. Xu and C. S. Kaplan. Image-guided maze construction. In *ACM Transactions on Graphics (TOG)*, volume 26, page 29. ACM, 2007.

[20] C. H. Yong and R. Miikkulainen. Coevolution of role-based cooperation in multiagent systems. *Autonomous Mental Development, IEEE Transactions on*, 1(3):170–186, 2009.