Distributed Age-Layered Novelty Search

Babak Hodjat[1], Hormoz Shahrzad[1], and Risto Miikkulainen[1,2]

[1]Sentient Technologies, Inc.
[2]The University of Texas at Austin
babak,hormoz,risto.miikkulainen@sentient.ai

Abstract

Novelty search is a powerful biologically motivated method for discovering successful behaviors especially in deceptive domains, like those in artificial life. This paper extends the biological motivation further by distributing novelty search to run in parallel in multiple islands, with periodic migration among them. In this manner, it is possible to scale novelty search to larger populations and more diverse runs, and also to harness available computing power better. A second extension is to improve novelty search's ability to solve practical problems by biasing the migration and elitism towards higher fitness. The resulting method, DANS, is shown to find better solutions much faster than pure single-population novelty search, making it a promising candidate for solving deceptive design problems in the real world.

Introduction

Novelty search is a new approach to population-based search motivated by the creativity and diversity of biological evolution (Stanley and Lehman, 2015; Lehman and Stanley, 2010a). Instead of optimizing a fitness objective, novelty search maximizes phenotypical diversity. Individuals are seeking novel niches to fill, developing emergent problem-solving abilities in the process. Novelty search is particularly powerful in domains that are deceptive, where it is necessary to discover low-fitness stepping stones first before actual solutions can be reached. Many tasks in artificial life are deceptive in this way, including behaviors that require developing learning, memory, and communication abilities (Lehman and Miikkulainen, 2014). Novelty search is thus a promising approach to constructing complex behavior in artificial life domains.

This paper aims to improve novelty search as a general problem-solving method in two ways. First, a distributed version of novelty search is developed. The idea is that search progresses in parallel in separate islands, and the best (i.e. most novel) individuals are periodically exchanged between them. Such a distribution is motivated by biological evolution, potentially leading to an implementation that can account for biological phenomena more accurately. It is also a well known diversity-maintenance technique in evolutionary algorithms in general (Whitley et al., 1999). However, the main motivation in this paper is to make novelty search computationally more powerful. Distribution makes it possible to scale novelty search to a much larger pool of individuals, and to take advantage of diversity between different novelty search runs. It also makes it possible to harness available parallel computing resources to serve novelty search. It therefore makes it possible to use novelty search to solve harder problems faster.

Second, a principled way of guiding novelty search towards high-fitness solutions is developed. Novelty is still the primary selection mechanism, but fitness is used to bias the process in two ways: (1) by migrating only the most fit individuals across the islands, and (2) by selecting the more fit of two similar individuals into the elitist pool. Such subtle biases do not prevent novelty search from creating and retaining novel individuals, but they make it more likely to create individuals that are solutions to the given problem.

Together these two extensions result in a powerful version of novelty search that can be used to solve difficult design problems in the real world. As a demonstration, in this paper they were implemented in an existing distributed evolution system called EC-Star (O'Reilly et al., 2013). This hub-and-spoke architecture manages a number of clients running separate evolutionary searches. A key feature of EC-Star is age-layering (Hodjat and Shahrzad, 2013): candidates are first evaluated in a small number of samples, and if they are promising, with more samples. Age-layering makes evolution more efficient, decreasing run times an order of magnitude or more (Shahrzad et al., 2016). It is also well-suited for evaluating novelty across many separate novelty searches.

The resulting method, Distributed Age-Layered Novelty Search, or DANS, is demonstrated on two challenging engineering design tasks: the 11-bit multiplexer, and the eight-input sorting network. The results show that DANS can find better solutions much faster than single-population novelty search. It is therefore a promising artificial life approach for solving deceptive problems in the real world.

Background and Related Work

The idea of divergent search, of which novelty search is an example, is first discussed, followed by existing work on combining novelty with fitness. The distributed evolution platform of EC-Star with age-layering, used to implement DANS, is then reviewed.

Objective vs. Divergent Search

In the traditional objective-based search, a population of candidate solutions are evolved to maximize a specific measure, or objective, called a fitness function. Individuals are selected for reproduction, and offspring are accepted into the population, if they score high in that function. The idea is that evolution thus gradually discovers better and better individuals, until it finds some that optimize the chosen objective.

Even though objective-based search is effective in many cases, there are two problems with it that are especially relevant in artificial life. First, it is not always clear how the objective function should be defined. The desired behavior may consist of many aspects that interact (such as speed, energy consumption, effectiveness, quality of the result), and some of them may be difficult to express formally (such as believability, creativity, elegance). Second, the domains are often deceptive, i.e. optimization requires creating individuals that do not perform well, but can serve as stepping stones in constructing those that do. This effect can be seen in many cognitive tasks that require memory, learning, or communication (Lehman and Miikkulainen, 2014), but it is also clear in the process of creating interesting images (Secretan et al., 2011).

Divergent search methods have recently emerged, mostly in the field of artificial life, as a potential solution to these issues. The idea is not to incrementally approach an optimum of a specified fitness function, but instead create as much diversity in the search as possible. This idea has been expressed in several forms, including empowerment (Salge et al., 2013), entropy maximization (Wissner-Gross and Freer, 2013), and behavioral diversity (Mouret and Doncieux, 2012). The particular formulation used in this paper is novelty search (Stanley and Lehman, 2015; Lehman and Stanley, 2010a), where individuals are rewarded based on how different they are from other individuals encountered so far. The novelty ρ for an individual x is defined as

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} \mathrm{dist}(x, \mu_i), \tag{1}$$

where μ_i is the ith nearest neighbor of x according to the distance metric dist. Note that distance is measured in the phenotypic, i.e. behavioral, space, not in the genotypic space. The motivation comes from biology: behavioral niches that are novel will survive, regardless of what their genetic coding is.

Novelty search can be surprisingly effective in solving problems, especially those that are deceptive. For instance, evolving a robot to run through a simulated maze, using distance to the goal as the fitness, is easy if the maze is relatively simple. However, dead ends close to the goal make it deceptive, and objective-based search often gets stuck. In contrast, novelty search will create individuals that explore all the different parts of the maze, and eventually will find a way to the goal even though such solutions require traveling away from it occasionally (Lehman and Stanley, 2010a).

Novelty search also provides an interesting abstraction of biological evolution. There is no specific goal in biological evolution; instead individuals and species survive if they find a niche that they can exploit. Life therefore rapidly spreads through the available niches, leading to species that are highly adapted to their environment, and to tremendous diversity overall. Extinction events can serve to accelerate this process by selecting for highly evolvable individuals and species (Lehman and Miikkulainen, 2015).

One aspect of biological novelty search that is not captured by current methods is that such search takes place simultaneously and in parallel across the space of solutions. Individuals and species do not necessarily compete with everyone in that space, but with those that are local to them. In other words, biological novelty search is distributed. It may result in discovering similar individuals multiple times, but it may also result in discovering more diverse individuals. This distribution is the first design principle of DANS in this paper.

Combining Novelty and Fitness

The second design principle of DANS is incorporating fitness as a component into divergent search. Even though biology may not have a goal, engineering problem solving does. At the very least there needs to be a mechanism for detecting viable solutions produced by the divergent search, but there may be a benefit in guiding it as well. Diversity and novelty is necessary for discovering the stepping stones, but since we know what we ultimately want to achieve, it may be possible to guide the search towards promising areas, without diluting its power.

Several approaches for combining fitness and novelty have been proposed, and shown to be effective in solving practical problems (Gomes et al., 2015). Many of them combine a fitness objective with a novelty objective in some way, for instance as a weighted sum (Cuccu and Gomez, 2011), or as different objectives in a multi-objective search (Mouret and Doncieux, 2012). Another approach is to keep the two kinds of search separate, and make them interact through time. For instance, it is possible to first create a diverse pool of solutions using novelty search, presumably overcoming deception that way, and then find solutions through fitness-based search (Krcah and Toropila, 2010). A third approach is to run fitness-based search with a large number of objec-

tive functions that span the space of solutions, and use novelty search to encourage search to utilize all those functions (Cully et al., 2015; Mouret and Clune, 2015; Pugh et al., 2015). A fourth category of approaches is to run novelty search as the primary mechanism, and use fitness to select among the solutions. For instance, it is possible to add local competition through fitness to novelty search (Lehman and Stanley, 2011). Another version is to accept novel solutions only if they satisfy minimal performance criteria (Lehman and Stanley, 2010b; Gomes et al., 2013).

This paper advances the techniques for combining novelty and fitness in this fourth category, in two ways. First, novelty search is run in each of the parallel and distributed clients, but periodically their most novel solutions are harvested at the system level for those that are the most fit—they are then injected into the populations of the parallel searches to bias them towards high fitness. This approach is an extension that utilizes the distributed nature of DANS. Second, in selecting an individual to keep from the least novel pair, each client search prefers the fitter choice, creating a fitness bias within each search. The results show that the resulting fitness-biased novelty search is more powerful than the standard version.

Distributed Evolution Through EC-Star and Age Layering

Age-layered fitness calculation is an approach suitable for data problems in which evolved solutions need to be applied to many fitness samples in order to measure a candidates fitness confidently (Hodjat and Shahrzad, 2013). Age layering is an elitist approach: best candidates of each generation are retained to be run on more fitness cases to improve confidence in the candidates fitness. The number of fitness evaluations (i.e. samples shown) in this method depends on the relative fitness of a candidate solution compared to others at the current state of the search.

Note that this age-layering technique is distinctly different from similarly named Age-Layered Population Structure (ALPS) method (Hornby, 2006). ALPS partitions populations into layers according to generations, with the main goal of maintaining diversity. Age layering in this paper is more closely related to the Early Stopping method in evolutionary robotics, where a complex evaluation is terminated if it is guaranteed not to produce offspring even if evaluated fully (Nolfi and Floreano, 2000; Bongard and Hornby, 2010; Bongard, 2011).

EC-Star (O'Reilly et al., 2013) is a massively distributed evolutionary platform that uses age-varying fitness as the basis for distribution, and thus makes it possible to distribute large data problems through sampling, hashing, and feature reduction techniques. The available data is divided into smaller chunks, each contributing to the overall evaluation of the candidates.

In EC-Star, age is defined as the number of fitness samples upon which a candidate has been evaluated. EC-Star uses a hub-and-spoke architecture for distribution, where the main evolutionary process is moved to the processing clients (Figure 1). Each client, or Evolution Engine, has its own pool and independently runs through the evolutionary cycle. At each new generation, an Evolution Engine submits its fittest candidates to the server, or Evolution Coordinator, for consideration. The submission takes place typically after each candidate has been evaluated on fixed number of samples, called the maturity age.

The Evolution Coordinator maintains a list of the best candidates so far. EC-Star achieves scale through making copies of genes at the server, sending them to Evolution Engines for aging, and merging the aged results reported back by the Evolution Engines. This process also allows the spreading of the fitter genetic material. EC-Star is massively distributable by running each Evolution Engine on a processing node (e.g. CPU) possibly with limited bandwidth and occasional availability (Hodjat et al., 2014). Typical runs utilize hundreds of thousands of processing units spanning across thousands of geographically dispersed sites.

In the Evolution Coordinator, only candidates of the same age-range are compared with one another (i.e. they are age-layered). Each age-range has a fixed quota, and a "shadow" of a candidate that has aged out of an age-layer is retained as a placeholder for filtering incoming candidates. In this manner, unreliable estimates do not dominate the evaluation process.

EC-Star with Age Layering is well suited for implementing DANS, as will be described next.

Distributed Age-Layered Novelty Search

In the DANS approach, the Evolution Engines (clients) are configured to use novelty search for their parent selection, while the Evolution Coordinator (servers) continues to make use of fitness to decide which individuals to allow in the server pool. Evolution Engines still receive individuals from Evolution Coordinators for aging, and these individuals are added to the local elitist pool, participating as parents in creating the next generation. Each Evolution Engine, however, solely operates on the basis of novelty rather than fitness, selecting the most novel individuals in the local pool as parents. The algorithm running in each Evolution Engine is summarized in Table 1.

For each sample in the data set, a hash representation of each individuals' behavior is logged. Each individual receives a sample from the data set as its input and generates an output that defines the action to be executed. For example, in the 11-multiplexer problem, where there are eight actions, each referring to one of the data bits in the multiplexer, the behavior logged is the address of the data bit that the individual outputs for the given input.

After maturity age, individuals judged to be the most novel are selected for the elitist pool. Instead of a global

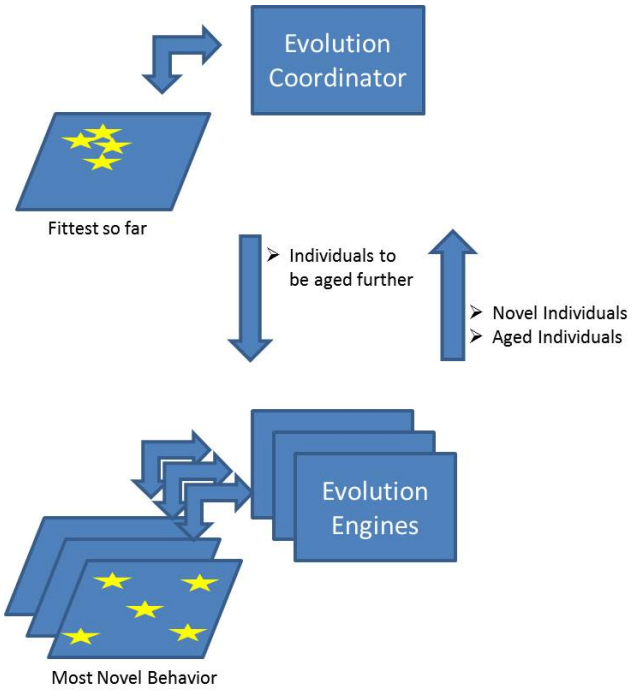

Figure 1: Implementation of Distributed Age-Layered Novelty Search (DANS) on the EC-Star System. DANS consists of a number of Evolution Engines running novelty search and a single Evolution Coordinator in a hub-and-spoke arrangement. Each candidate in each Evolution Engine is evaluated with a fixed number of samples (called the maturity age); each Engine then sends their most novel individuals to the Evolution Coordinator. The Coordinator maintains a list of these highly novel individuals ordered by fitness, and sends the most fit ones back to other Evolution Engines for further evaluation (i.e. aging) and evolution. In this manner, multiple novelty searches execute in parallel, exploring the space with more diversity, benefiting from each other's discoveries, biased towards areas with higher fitness.

archive, novelty is measured in the current population so that every individual's behavior log is based on the same set of examples. Using this log as Cartesian coordinates, Euclidean distances between individuals are calculated, and one of the individuals in the pair of individuals nearest one another is eliminated. This process is repeated until the quota for parents in the elitist pool is met. The resulting set of parents is then used to create the next generation.

Two different versions are implemented in forming the elitist pool. The first one is based purely on novelty: in the pair of most similar individuals, the one that's nearer to at least one of the other individuals in the pool is removed. The second one is based partly on fitness: in that pair, the one with the lower fitness is removed, thus subtly biasing the system to favor genes with better fitness. These two versions, called Pure Novelty and Hybrid, will be compared in the experiments that follow.

1. Receive a batch of individuals from the Evolution Coordinator.
2. If pool has capacity, fill it with randomly generated individuals.
3. Test each individual in the pool on a maturity-age number of fitness samples (each individual is run on the same sample as the others), and construct the representation of its behavior.
4. The individuals received from Coordinator have now been evaluated with more samples than before: report the results back to the Coordinator.
5. Calculate the minimum of pair-wise distances of all individuals in the pool and discard one of that pair based on distance from all other genes (the pure-novelty version), or fitness (the hybrid novelty/fitness version). Do this until only the elitist percentage of individuals remain.
6. Report the most novel individuals (i.e. the elitists from prior step) to the Coordinator.
7. Refill the pool by applying crossover and mutation operators on the elitist genes from Step 5.
8. Go to 1.

Table 1: The Evolution Engine Algorithm, i.e. the sequence of steps in advancing evolution for one generation.

Experiments

DANS was tested on two practical design optimization problems: the 11-Multiplexer and the eight-input sorting network. Each problem is described first, with its experimental setup, and then results.

The 11-Multiplexer Domain

Multiplexer functions have long been used to evaluate machine learning methods because they are difficult to learn but easy to check. In general, the input to the multiplexer function consists of u address bits A_v and 2^u data bits D_v, i.e. it is a string of length $u + 2^u$ of the form $A_{u-1}...A_1A_0D_{2^u-1}...D_1D_0$. The value of the multiplexer function is the value (0 or 1) of the particular data bit that is singled out by the u address bits. For example, for the 11-Multiplexer, where $u = 3$, if the three address bits $A_2A_1A_0$ are 110, then the multiplexer singles out data bit number 6 (i.e. D_6) to be its output (Figure 2).

A Boolean function with $u + 2^u$ arguments has 2^{u+2^u} rows in its truth table. Thus, the sample space for the Boolean multiplexer is of size 2^{u+2^u}. When $u = 3$, the search space is of size $2^{2^{11}} = 2^{2048} \approx 10^{616}$. However, since evolution can also generate redundant expressions that are all logically equal, the real size of the search space can be much larger, depending on the representation.

Following prior work on the 11-Multiplexer problem (Shahrzad and Hodjat, 2015), a rule-based representation was used where each candidate specifies a set of rules of

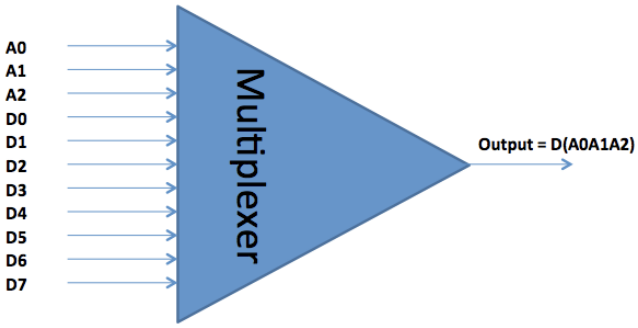

Figure 2: The 11-Multiplexer. The three address bits (top) specify one of the eight data bits (bottom) whose value will then be output. Multiplexers are a good test domain for design optimization because the search space is very large, but it is easy to check whether a design is valid.

the type

$$< \text{rule} > ::= < \text{conditions} > \rightarrow < \text{action} > .$$

The conditions specify values on the bit string and the action identifies the index of the bit whose value is then output. For instance, the following rule outputs the value of data bit 6 when the first three bits are 110:

$$< A_0 = 0 \ \& \ A_1 = 1 \ \& \ !A_2 = 0 > \rightarrow \ D_6.$$

These rules are evolved through the usual genetic operators in genetic programming (Berlanga et al., 2010).

In the 11-multiplexer experiments, each evolution engine has a pool size of 4000, an elitist percentage of 5%, and a maturity age of 128. That is, in each generation, each of the 4000 candidates is evaluated once with 128 randomly chosen multiplexer input samples, for a total of 512,000 evaluations per generation. At the top age-layer each candidate has thus seen 2048 samples. Crossover combines subsets of rules of each parent; mutations modify components of each rule. Fitness is defined as the number of samples an individual processes correctly, outputting the value of the data bit specified by the address bits in the 11-bit input sample. The novelty measure is the data bit address outputted by the individual for each sample. Each evolution is run until a valid multiplexer is found, i.e. one that outputs the correct bit for every possible combination of address bits.

Experiments were run comparing non-distributed runs to distributed runs with eight evolution engines per run. Two versions of distributed runs were compared: those with pure-novelty elitism, and those with hybrid novelty/fitness elitism. The non-distributed runs were implemented as hybrid runs on a single evolution engine, using the same parameters as the distributed version. Each experiments was repeated ten times, and the results averaged.

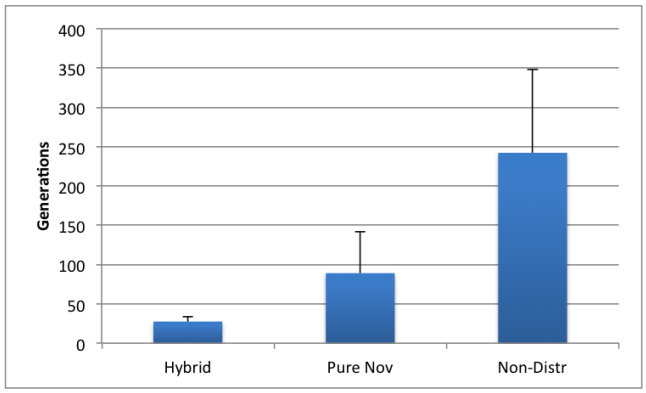

Figure 3: Performance of Hybrid and Pure-Novelty DANS vs. Non-Distributed Novelty Search on the 11-Multiplexer Problem. The plot shows the average and standard deviation of number of generations to find a valid solution. DANS significantly outperforms non-distributed novelty search, and the hybrid version of DANS the pure novelty version. The speedup is approximately linear in Evolution Engines, suggesting that DANS is an effective way to parallelize novelty search.

11-Multiplexer Results

The results are summarized in the bar graph shown in Figure 3. The main conclusion is that DANS significantly outperforms the non-distributed runs; within DANS, the hybrid elitism outperforms pure novelty. The hybrid version found a valid solution in 27.5 generations on average, pure novelty in 89.1 generations, and non-distributed evolution in 242.2 generations. Thus, distribution across the eight Evolution Engines makes evolution more reliable and speeds it up significantly, i.e. approximately linearly in the number of Evolution Engines.

The Sorting Network Domain

The second experimental domain is minimization of eight-input sorting networks. A sorting network of n inputs is a fixed layout of comparison-exchange operations (comparators) that sorts all inputs of size n (Figure 4; Knuth 1998). Since the same layout can sort any input, it represents an oblivious or data-independent sorting algorithm, that is, the layout of comparisons does not depend on the input data. The resulting fixed communication pattern makes sorting networks desirable in parallel implementations of sorting, such as those in graphics processing units, multi-processor computers, and switching networks (Kipfer et al., 2004; Baddar, 2009; Valsalam and Miikkulainen, 2013).

Beyond validity, the main goal in designing sorting networks is to minimize the number of layers, because it determines how many steps are required in a parallel implementation. A tertiary goal is to minimize the total number of comparators in the networks. Designing such minimal sort-

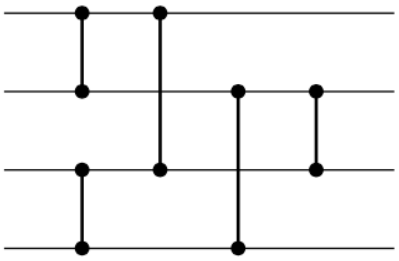

Figure 4: A Four-Input Sorting Network. This network takes as its input (left) four numbers, and produces output (right) where those number are sorted (small to large top to bottom). Each comparator (connection between the lines) swaps the numbers on its two lines if they are not in order, otherwise it does nothing. This network has four layers and five comparators, and is the minimal four-input sorting network. Minimal networks are generally not known for input sizes larger than eight, and designing them is a challenging optimization problem.

ing networks is a challenging optimization problem that has been the subject of active research since the 1950s (Knuth, 1998). Although the space of possible networks is infinite, it is relatively easy to test whether a particular network is correct: If it sorts all combinations of zeros and ones correctly, it will sort all inputs correctly (Knuth, 1998). Sorting networks are therefore a good domain to test the power of evolutionary algorithms; indeed many of the recent advances in sorting network design are due to evolutionary methods (Valsalam and Miikkulainen, 2013). The eight-input case is a good test case because the optimal network is known: it has six layers and 19 comparators (Knuth, 1998).

The sorting network representation for DANS is built on the rule-set representation of the 11-multiplexer. Each rule represents a layer of comparators; each condition within each rule identifies the input lines of the comparator; the action is not used. In this manner, it is possible to evolve sorting networks using the same methodology as for evolving rule sets. As a matter of fact, all Evolution Engine settings are the same as for the 11-Multiplexer experiments. In particular, the maturity age is 128 samples and the individuals in the top age layer have been tested with 2048 random samples.

The fitness of the network is primarily based on its ability to sort correctly, secondarily by the number of layers, and tertiarily by the number of comparators:

$$F = aS - (2^n L + C), \tag{2}$$

where a is a proportionality constant ($10000 * 2^{16}$ in these experiments), S is the number of samples the network sorts correctly, n is the number of lines (8 in these experiments), L is the number of layers, and C is the number of comparators in the network. Because all three of these goals need to be optimized simultaneously, sorting networks represent a more challenging and open-ended, as well as more deceptive, domain than the 11-multiplexer.

In order to measure the novelty in sorting behavior, note that there is no action to rely on, but instead the behavior needs to be constructed from the structure of the network itself. To this end, each input line is represented with a successive prime number, i.e. 1, 2, 3, 5, 7, 11, 13, and 17. The sorting network is run on the sample input, and for each pair of lines that it exchanges, the corresponding prime numbers are multiplied. The product of these values constitutes a hash for the phenotypical behavior on that sample. For example, if the sorting network has the structure

Layer 1: sort(line0, line3) and sort(line1, line2)
Layer 2: sort(line0, line4),

and the sample is 11010100 (with lines ordered 7..0), the network will rearrange it to 11000011. The phenotypical hash is then

(2 * 3) * (1 * 7) = 42.

A vector of these hash values for a number of samples (i.e. the maturity age) represents the behavior of the network, and the Euclidean distance between these vectors is used to measure novelty.

The sorting network experiments were all run until 1000 generations (which takes about two hours of total CPU time on an Intel i7 2.60GHz machine). The two versions of DANS and the non-distributed version were then compared in three dimensions (1) how fast they found a valid sorting network, (2) how many layers and (3) how many comparators did the best network found have. The results were averaged over ten runs.

Sorting Network Results

The DANS approach found valid sorting networks in 14.4 (hybrid) and 25 (pure) generations on average, compared to the non-distributed approach which took 98.1 generations on average (Figure 5. Similarly, DANS found solutions with significantly fewer layers than the non-distributed version (32.1), with the hybrid version significantly fewer than the pure novelty version (7.4 vs. 15.7; Figure 6). It was also most economical in the number of comparators: Whereas the non-distributed version used 56.2 comparators on average, the pure novelty version used 32.2 and the hybrid version only 21.7 (Figure 7).

Interestingly, two of the ten hybrid runs actually found optimal sorting networks, with six layers and 19 comparators, within the 1000 generations. These results suggest that the hybrid version of DANS could be used to discover new minimal networks, given sufficient computing effort.

Discussion and Future Work

The DANS approach can be seen as a highly robust artificial life system in which islands of evolution are searching for

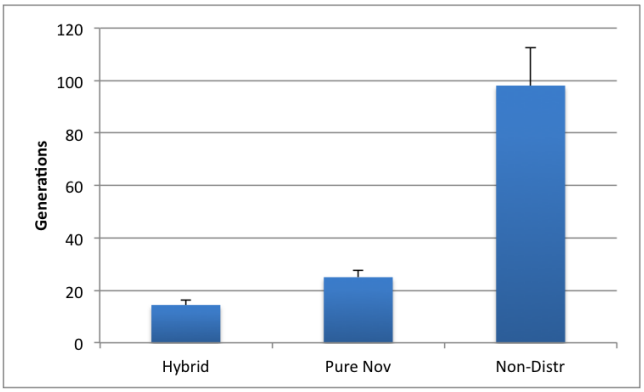

Figure 5: Number of Generations Hybrid and Pure-Novelty DANS and Non-Distributed Novelty Search Need to Discover a Valid Eight-Input Sorting Network. DANS significantly outperforms the non-distributed version, and hybrid version of DANS the pure novelty version.

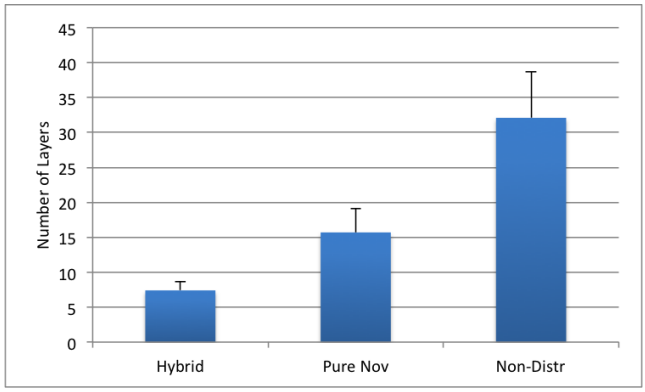

Figure 6: Number of Layers Discovered by Hybrid and Pure-Novelty DANS vs. Non-Distributed Novelty Search in 1000 Generations on the Eight-Input Sorting Network Problem. After validity, minimizing the number of layers is the main design goal; DANS significantly outperforms the non-distributed version, and hybrid version of DANS the pure novelty version.

behavioral niches to fill in the search space. Occasionally their most novel solutions migrate to a coordinator that aims to solve a particular problem, and therefore injects guidance into the islands in terms of the most fit of those novel individuals.

As a practical method for problem solving, DANS finds better solutions significantly faster than a similar non-distributed search: In the test problems in this paper, the speedup is approximately linear in the number of Evolution Engines. This result is remarkable because the search problem cannot be simply divided into subproblems that could be solved independently in parallel. Instead, the result is likely due to the larger total population of individuals and the increased diversity across multiple islands. The distribution makes it possible to combine fitness with novelty search effectively, by incorporating it into the migration between the islands, as well as in the selection of elitist individuals. DANS thus makes it possible to apply novelty search effectively to practical design problems.

Computationally, the system is scalable, and the coordinators can be federated (Hodjat et al., 2014). The system is also robust because it can tolerate temporarily losing its ability to coordinate (e.g, due to communication problems, or server outages, etc.), and it can even reconstruct its list of candidate solutions, should the data be lost at the coordinator. The system can also tolerate the loss of evolution engines. The approach can therefore be used to tackle big data problems that require massive amounts of computing to solve, such as minimizing large sorting networks or VLSI design in general, optimizing large-scale logistics and scheduling problems, protein folding and other biomedical optimization problems, and in general problems where each processing node can only have access to a subset of the data through sampling.

DANS can also be useful in dynamic problems where the fundamental attributes of the problem can change through time: the evolution engines operate based on the behavior of the solutions, striving to achieve maximal coverage of the search space, versus concentrating on subspaces defined by the peculiarities of the fitness landscape. DANS is thus a mechanism that converts a powerful principle in artificial life into a practical tool for solving challenging engineering design problems.

Conclusion

This paper presents DANS, a parallel distributed design for the novelty search algorithm, and a principled way of combining fitness with novelty. These extensions result in a system that can discover better solutions much faster than standard novelty search. It thereby shows how fundamental ideas in artificial life can be useful in problem solving in the real world. DANS should be most useful in finding optimal solutions to big-data problems such as those in engineering design.

References

Baddar, S. W. A. (2009). *Finding Better Sorting Networks*. PhD thesis, Kent State University.

Berlanga, F. J., Rivera, A., del Jesús, M. J., and Herrera, F. (2010). Gp-coach: Genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems. *Information Sciences*, 180(8):1183–1200.

Bongard, J. C. (2011). Innocent until proven guilty: Reducing robot shaping from polynomial to linear time. *IEEE Transactions on Evolutionary Computation*, 15:571–585.

Bongard, J. C. and Hornby, G. S. (2010). Guarding against premature convergence while accelerating evolutionary search. In

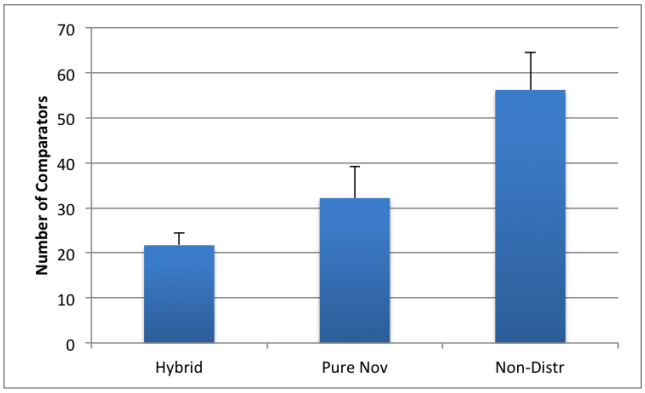

Figure 7: Number of Comparators Discovered by Hybrid and Pure-Novelty DANS vs. Non-Distributed Novelty Search in 1000 Generations on the Eight-Input Sorting Network Problem. Number of comparators is the third, and least important, component of fitness; again DANS significantly outperforms the non-distributed version, and hybrid version of DANS the pure novelty version.

Proceedings of the Genetic and Evolutionary Computation Conference.

Cuccu, G. and Gomez, F. (2011). When novelty is not enough. In *Proceedings of the 2011 International Conference on Applications of Evolutionary Computation - Volume Part I*, pages 234–243, Berlin, Heidelberg. Springer-Verlag.

Cully, A., Clune, J., Tarapore, D., and Mouret, J. B. (2015). Robots that can adapt like animals. *Nature*, 521:503–507.

Gomes, J., Mariano, P., and Christensen, A. L. (2015). Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 943–950, New York, NY, USA. ACM.

Gomes, J., Urbano, P., and Christensen, A. L. (2013). Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7:115–144.

Hodjat, B., Hemberg, E., Shahrzad, H., and OReilly, U.-M. (2014). Maintenance of a long running distributed genetic programming system for solving problems requiring big data. In *Genetic Programming Theory and Practice XI*, pages 65–83. Springer, New York.

Hodjat, B. and Shahrzad, H. (2013). Introducing an age-varying fitness estimation function. In Riolo, R., Vladislavleva, E., Ritchie, M. D., and Moore, J. H., editors, *Genetic Programming Theory and Practice X*, pages 59–71. Springer, New York.

Hornby, G. S. (2006). ALPS: The age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 815–822.

Kipfer, P., Segal, M., and Westermann, R. (2004). Uberflow: A gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA. ACM.

Knuth, D. E. (1998). *Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley Professional, 2 edition.

Krcah, P. and Toropila, D. (2010). Combination of novelty search and fitness-based search applied to robot body-brain co-evolution. In *Proceedings of the 13th Czech-Japan Seminar on Data Analysis and Decision Making in Service Science.*

Lehman, J. and Miikkulainen, R. (2014). Overcoming deception in evolution of cognitive behaviors. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014)*, Vancouver, BC, Canada.

Lehman, J. and Miikkulainen, R. (2015). Extinction events can accelerate evolution. *PLoS ONE*, 10(8):e0132886.

Lehman, J. and Stanley, K. O. (2010a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 2011:189–223.

Lehman, J. and Stanley, K. O. (2010b). Revising the evolutionary computation abstraction: Minimal criteria novelty search. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Lehman, J. and Stanley, K. O. (2011). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO 2011)*, Dublin, Ireland.

Mouret, J. B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv*, 1504.04909v1.

Mouret, J.-B. and Doncieux, S. (2012). Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, 20:91–133.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. MIT Press, Cambridge.

O'Reilly, U.-M., Wagy, M., and Hodjat, B. (2013). EC-Star: A massive-scale, hub and spoke, distributed genetic programming system. In Riolo, R., Vladislavleva, E., Ritchie, M. D., and Moore, J. H., editors, *Genetic Programming Theory and Practice X*, pages 73–85. Springer, New York.

Pugh, J. K., Soros, L. B., Szerlip, P. A., and Stanley, K. O. (2015). Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA. ACM.

Salge, C., Glackin, C., and Polani, D. (2013). Empowerment - an introduction. *CoRR*, abs/1310.1863.

Secretan, J., Beato, N., D'Ambrosio, D. B., Rodriguez, A., Campbell, A., Folsom-Kovarik, J. T., and Stanley, K. O. (2011). Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 19:345–371.

Shahrzad, H. and Hodjat, B. (2015). Tackling the Boolean multiplexer function using a highly distributed genetic programming system. In Riolo, R., Worzel, W. P., and Kotanchek, M., editors, *Genetic Programming Theory and Practice XII*, pages 167–179. Springer, New York.

Shahrzad, H., Hodjat, B., and Miikkulainen, R. (2016). Estimating the advantage of age-layering in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016)*, Denver, CO.

Stanley, K. O. and Lehman, J. (2015). *Why Greatness Cannot Be Planned: The Myth of the Objective*. Springer, Berlin.

Valsalam, V. K. and Miikkulainen, R. (2013). Using symmetry and evolutionary search to minimize sorting networks. *Journal of Machine Learning Research*, 14(Feb):303–331.

Whitley, D., Rana, S., and Heckendorn, R. B. (1999). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–48.

Wissner-Gross, A. D. and Freer, C. E. (2013). Causal entropic forces. *Physical Review Letters*, 110:168702.