
Conor Ryan  
BDS Lab  
Department of Computer Science and Information Systems  
University of Limerick  
Limerick, Ireland  
conor.ryan@ul.ie

Atif Rafiq  
BDS Lab  
Department of Computer Science and Information Systems  
University of Limerick  
Limerick, Ireland  
atif.atif@ul.ie

Enrique Naredo  
BDS Lab  
Department of Computer Science and Information Systems  
University of Limerick  
Limerick, Ireland  
enrique.naredo@ul.ie

Abstract—We present Pyramid, a Hierarchical Genetic Algorithm that decomposes problems by first tackling simpler versions of them, before automatically scaling up to more difficult versions while also reducing the population size. Pyramid takes its name from the architectural phenomenon of the Mayan pyramid in Chichen-Itza, which, although constructed from the bottom up, operates in a top down manner through interactions with the sun. This gives a two stage approach: initially we create our pyramid of experiments, with the most complex fitness functions at the bottom, and with increasing complexity of decomposed version as we move up through the pyramid. Runs start at the top of the pyramid and populations descend through it, decreasing in size, being exposed to increasingly complex fitness functions, until, at the bottom layer, we have small populations with the original full fitness function. We conduct experiments comparing the performance of Pyramid for a suite of difficult unimodal and multimodal functions. The experimental results show that in three of four cases, Pyramid achieves the same or better fitness scores as standard algorithms, with substantially fewer evaluations, and that in two cases it actually performs statistically significantly better.

Index Terms—Hierarchical GAs, Incremental Evolution, Layered Learning, Individuals processed

I. INTRODUCTION

There have been many investigations into decomposition of problems for EAs, such as Automatically Defined Functions (ADFs) by Koza [1], Module Acquisition (MA) [2], Hierarchy Locally Defined Modules (HLDM) [3], macro-evolutionary algorithm [4] and Multi-Objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [5]. Problem decomposition in GAs is often not as clear as with Genetic Programming (GP), however, due to the general unavailability of reuse. This has led to some work, see Section II, that focuses on decomposing the fitness function rather than on the research task. That is, a modified, easier version of the fitness function is first tackled, after which successful individuals are then tested on the actual fitness function.

GP-focused decomposition techniques succeed essentially by changing the solution representation. While there are many approaches to this, for example PolyGP [6] and LLGP [7], they all essentially reduce the complexity of the problem by encapsulating useful information and making them available on later runs or generations.

We introduce Pyramid, which takes inspiration from GP-style techniques of decomposition by iteratively changing the representation of individuals in a Genetic Algorithm (GA). The change in representation is characterised by starting with relatively small individuals, up to five times shorter than the actual desired solution, and, at regular intervals, increasing their size, until they are the desired length.

An interesting side effect of this process is that Pyramid permits us to reduce the population size each time we increase the length of the individuals. In this work, we reduce the population to 20% of its size with each step in the hierarchy, with a result that, by the time the individuals are full length, the population size can be reduced a factor of up to 16 from its original size.

We compare Pyramid on a selection of GA Benchmark problems, including multi-modal function optimization and show that, on this set of benchmarks, Pyramid always achieves at least the same performance in terms of solution quality, but does so at a reduced cost ranging from 55% to 99% in terms of the number of individuals evaluated, meaning that with just only 2% of the evaluations Pyramid gets competitive results.

In the next section, we highlight the existing hierarchical techniques in EC and explain how they are different from each other. We then present Pyramid (Section III). Section IV is concerned with the methodology used for this study and the benchmark problems. Experimental results are discussed in section V. Finally, section VI concludes the paper with the conclusion and future work.

II. BACKGROUND AND RELATED WORK

While one inspiration behind this work is the way in which GP can effectively simplify a problem with machinery for module acquisition, we focus here on hierarchical methods that are generally applicable to all EAs.

Generally, all these methods attempt to overcome the bootstrapping problem, by having a population start with an easier
version of the problem before slowly exposing it to more complex versions.

Based on the concept of hierarchy, researchers have experimented with different approaches in order to overcome the bootstrapping problem and to enable the evolution of behaviours for complex tasks. We break up the approaches into three as Hierarchical Genetic Algorithm (HGA), Incremental Evolution (IE) and Layered Learning (LL).

A. Hierarchical Genetic Algorithms (HGAs)

HGAs consist of hierarchies of GAs that generally employ different fitness functions. Individuals can migrate up (and occasionally down) through the various levels. For example, a HGA in [8] is presented in the form of binary tree spawning three levels (shown in Fig. 1). Three different fitness functions were employed, ranging from coarse through intermediate and precise. The coarse model was used at the lowest level, and they found that HGA obtained the same results as classic GA but three times faster.

A similar structure has been presented by [9] but with the same fitness function at each level. The execution starts at the leaf nodes, with each population evolving separately until some termination criteria is met (300 generations, in their case) after which nodes with the same parent are merged together, and thus the population size increased at each upper-level. The procedure is then repeated until the root node is reached, and the best solution is extracted from the top in the same way as it would normally be. They found that sub-populations with hierarchical structure outperformed traditional one and described several hierarchies with varying numbers of levels and branches, although only reported on the performance of a hierarchy with three levels and a branching factor of two.

Later work [10] examined more complex hierarchies and used two additional multi-model functions for benchmarking. They reported best results with higher numbers of branches and levels, although did not investigate at what point the overhead becomes too expensive.

Another form of HGA is provided in [11] that can solve hierarchical problems. This incrementally forms a set of modules that are used to adapt the representation of the search, and can thereby gradually restrict the search space without excluding optimal solutions. They investigated the condition under which HGAs can efficiently solve hierarchical problems.

B. Incremental Evolution (IE)

IE starts with a population that is already trained on a simpler but related task, rather than starting from initial random population. The difficulty can be increased using different fitness function, known as Functional Incremental Evolution (FIE), or using different program parameters, known as Environmental Incremental Evolution (EIE). Fig. 2 shows a conceptual diagram of IE.

According to their results, the identification of effective sampling methods is an important factor in the development of efficient hierarchical algorithms.

C. Layered Learning (LL)

A bottom-up approach is employed by LL, where learning achieved at the lower layers helps facilitate learning required
at upper layers. Individuals at each layer have different capabilities. [15]

Soccer players were evolved by [16] using a two layer approach. Initially, agents had to learn to control the ball effectively, after which they were moved to the higher level that involved interacting with other players in a team.

A two layer GP system was proposed by [17]. In the first layer, the solutions of the sub-problems are found then using these solutions, the solution of the original problem is found in the second layer. The results of the two layer GP system is compared against Multi Gene Genetic Programming (MGGP) and Fast Function Extraction (FFX). However, the proposed method outperforms FFX in only 1 and MGGP in 2 out of 6 benchmarks, respectively.

Conventionally, LL necessitates the understanding of problem knowledge and how it can be decomposed into sub-tasks, but [18] investigated two approaches that do not require such knowledge and explored the use of LL in the context of GP.

Rather than using a single GP run, Wave [19] employed a sequence of heterogeneous GP runs collaborate to produce a joint solution. Each period (short GP run) produced an individual that was used to update the target values of the subsequent period, thus reducing the difficulty of the problem faced by later periods.

III. PYRAMID

Pyramid takes inspiration from several systems, as it uses increasingly more complex individuals as in [17] and increasingly more precise fitness functions as in [8]. Note that in its current form as shown in Fig. 3, Pyramid essentially gets increasingly more precise fitness functions for free, because the longer the individuals are, the more precise the fitness function is. Many hierarchies take a pyramidal form, as does many hierarchies take a pyramidal form, as does the ancient Mayan one, is shown in Fig. 3. Steps on the Pyramid like a snake slowly meandering down the staircase.

Thus, we initially construct our Pyramid, deciding the number of levels and how the population and genome sizes will vary, in such a way that the smallest genome, largest population and simplest fitness function are at the top, and the largest genome, smallest population and most complex fitness function are at the bottom. With each step in the Pyramid, we adjust the population size down and the genome size up.

![Fig. 3. Graphical representation of the Pyramid approach.](image)

**Algorithm 1: Pyramid**

**Input:** The given problem and the solution representation.

**Output:** Problem decomposition and scaling down population size.

1. /* The solution representation: */
2. /* Decompose the genome into k blocks */
3. /* Initial population using first block */
4. /* If full genome, then get best individual */
5. /* Assign fitness to each individual */
6. /* If full genome, then get best individual */
7. /* If full genome, then get best individual */
8. /* Otherwise, keep scaling up genome */
9. /* Average fitness from pop */
10. /* Count how many individuals get top fitness */
11. /* Count how many individuals get top fitness */
12. /* Count how many individuals get top fitness */
13. /* Count how many individuals get top fitness */
14. /* Count how many individuals get top fitness */
15. /* Count how many individuals get top fitness */
16. /* Count how many individuals get top fitness */

Even though this approach can be used in most of the population-based algorithms, in this work a GA is used to automatically evolve more complex solutions. The solution representation is given by the genome, which is a set of genes: \( g_1, g_2, \ldots, g_n \), with \( n \) the total number of genes in the genome. Pseudocode for Pyramid is given in Algorithm 1 and graphical representation of the Pyramid approach resembling the ancient Mayan one, is shown in Fig. 3. Steps on the Pyramid staircase are labelled H-LX, where X stands for the hierarchical level. In Pyramid, we consider two means to address problem decomposition: genome and population decomposition, at the right and left sides respectively.

Pyramid decomposes the genome onto \( k \) subsets of genes, named blocks: \( B_1, \ldots, B_k \). For instance, \( B_1 = [g_1, g_2, \ldots, g_m] \), which is used for the initial population using first block. Then, each individual \( I_i \) in the initial population is randomly initialized corresponding to the first block \( I_i = \text{rand}(B_1) \).

The next step, as with GA conventional approaches, is to evaluate the population, assigning a fitness score to each individual \( F_i = \text{fitness}(I_i) \), and getting new populations through standard genetic operations (crossover and mutation).
Fig. 6 shows how the population decreases and genome length increases. Depending on the threshold used we can build fewer steps means less scaling and higher populations at the height and width as shown in Fig. 4, this would correspond to the current population, it is promoted to belong to the next population of individuals. When this set reaches a pre-specified number, $\beta$ (detailed below), of individuals, then the current population is deleted and replaced with the promoted set.

The genomes for this smaller population are scaled up, by adding next block $B$ to deal with a more difficult version of the problem. Although, there are several strategies to join this new block to the current genome, in this work this block is joined by adding randomly choosing features at the end.

Finally, when each individual in the population contains a complete genome $B_1, ..., B_k$, the population is evolved until the evolutionary process meets the stopping criteria. Notice that the population as this point can be substantially smaller than the initial population; experiment in this paper reduce the population to just 20% of its size each step.

Pyramids can be designed with different shapes, considering height and width as shown in Fig. 4, this would correspond to different numbers of levels and thus steps in the algorithm. Fewer steps means less scaling and higher populations at the end. Depending on the threshold used we can build Pyramids with different slopes as shown in Fig. 5. Moreover, Fig. 6 shows how the population decreases and genome length increases.

IV. EXPERIMENTAL SET UP

Pyramid faces several set up questions, which we refer to here as Research Questions (RQs) due to the fact that we had to establish answers to these for the first time. These include: how many levels should be employed (RQ1) and when should individuals be promoted to lower levels (RQ2); this is the $\alpha$ value mentioned above. In addition, we needed to establish how many individuals should be promoted (RQ3); this is the $\beta$ value mentioned above, and how we should assign values to the new parts of these individuals (RQ4).

For RQ1, i.e., how many levels to use, we experimented with a range of levels from 2 to 5, and decided that the trigger point for promotion was that the top 20% of the population is five times better than the average fitness of the population. This was a somewhat arbitrary choice and future work will explore different values as well as strategies for setting this dynamically. Note that once the trigger point is reached, the levels stop evolving.

The trigger point chosen for RQ2 answered RQ3 for us, as our new population is made up of all the promoted individuals, which leaves us with RQ4, that is, what values should be chosen for the new parts of the genomes? We chose the simplest possible approach and randomly choose values, although, as noted in Section VI there are other approaches, particularly those that use information from previous runs. The lower-most layer is the most important because it returns the solution of the original problem. Therefore, its stopping criteria is different from other layers as it does not have to promote individuals to next-level. The lower most layer stops when the average fitness in the population doesn’t improve for 10 generations.

We used OpenGA [20], an open source c++ library for GAs to implement the Pyramid.

A. Problems

Four mathematical functions, two uni-modal (Sphere and Rosenbrock) and two multi-modal (Rastrigin and Griewank), as shown in Table I, were chosen to test the performance of our algorithm. These functions are well-known optimization problems [21] and have been frequently used in other hierarchical studies [9] [10]. The optimal value for all these function is 0.

V. EXPERIMENTAL RESULTS AND DISCUSSION

A. Experiments

Using the set up from Table II, a baseline experiment was initially run using a standard GA, followed by four variants of our algorithm, each with varying levels. These are referred
to as H-LX where X is the number of levels in the hierarchy.

Crossover (One Point) and mutation (Uniform) rates were set to 0.8 and 0.005, respectively, and, in addition, an elitism level is set to 5% of the population size. We used a proportion because the population size changes over time. Each experiment was repeated 50 times.

### B. Results

The correlation between best fitness and number of individuals processed was tested. Figures 7, 8, 9 and 10 show the resulting graphs for Sphere, Rosenbrock, Rastrigin and Griewank respectively.

Tables III, IV, V and VI are their statistical comparison. In all but the Griewank problem, Pyramid performs either the same or statistically significantly better with dramatically fewer evaluations. Each table lists down the results in two groups, i.e., Individuals Processed and Best Fitness.

In the first group, we show the maximum number of individuals that could be processed (and hence the upper bound on cost) and the actual number of individuals being processed (when the population converged). In the second group, the average (Avg), the standard deviation (std) and the p-value of the best fitness are listed. In cases where Pyramid performed statistically significantly better, results are indicated with a +, where it is worse, we use a - and, where there is no statistically significant difference, we use a *.

In all cases, Pyramid uses dramatically fewer individuals; typically, for HL-2, the best performing version of Pyramid, we use approximately 25% of the total individuals used by the baseline. In cases where HL-3 outperformed the baseline (Rastrigin and Sphere) the number of individuals is smaller again.

In general, the improvement in performance doesn’t extend
Fig. 7. Sphere Results: Comparison of Baseline with each H-LX. Clearly, Pyramid can achieve the same fitness but with fewer individuals processed.

Fig. 8. Rosenbrock Results: Comparison of Baseline with each H-LX. H-L2 can produce individuals with the same fitness as the baselines, but with less than half the individuals produced.

Fig. 9. Rastrigin Results: Comparison of Baseline with each H-LX. Both H-L2 and H-L3 outperform the baseline with fewer individuals processed, with H-L3 processing 91.5% fewer individuals.

For this reason, we did not build the Pyramid with more than 5 levels, although if the population didn’t reduce by quite so much each time, this may not always hold true, as we discuss in the future work section. The only problem in which Pyramid didn’t perform either better or the same was the Griewank problem, but recall that no effort was put into trying to optimize the $\alpha$ and $\beta$ values to control when individuals
move to the next level; clearly there is scope to optimize this. Overall, these results indicate that Pyramid finds the fitness as good or better than the baseline in all but one case, and does so with remarkably fewer individuals processed.

An interesting feature of Pyramid is its ability to scale down the population size, thus reducing the computational effort while still getting competitive results in most cases. Nevertheless, this reduction come at a cost as, in a small
number of cases when the population size is too low, then
the method may no longer have any significant improvements.
This issue will be studied in a future work. In general,
however, for the problem domains addressed in this work, this
problem did not manifest itself.

VI. CONCLUSIONS AND FUTURE WORK

We have presented a Pyramid, a hierarchical system that
reduces the population size at each lower level with increasing
the genome length. We conducted our experiments using four
mathematical functions (two unimodal and two multimodal)
and show that on three of these problems Pyramid performs
either better (two) or the same, but with reductions of between
55% and 90% in the number of individuals processed.

There are some parameters related to this approach we need
to analyse to find what are the best values, such as the number
of levels and the rate at which the population size decreases
need to be examined, in addition to automatically determine
the number of levels and the task of each level.

The Griewank problem in particular indicates that higher values
of $\alpha$ are likely to be useful in some cases.

Future work will analyse how the Pyramid parameters
should be set according to the problem addressed to obtain
reasonable and competitive solutions while using smaller
populations.

ACKNOWLEDGMENT

This work is supported by the Science Foundation of Ireland
(SF) research grant 16/IA/4605. The authors would like to
thank Aidan Murphy, Sheraz Anjum and Michael Tetteh for
their help with the statistically comparing results and detecting
some bugs from the code.

REFERENCES

programming using local modules. Secretary of the SFB 531, 1999.
algorithms,” in Proceedings of the 2001 Congress on Evolutionary
223–231.
[5] Q. Zhang and H. Li, “Moea/d: A multiobjective evolutionary algo-
rum based on decomposition,” IEEE Transactions on evolutionary
ming for a cooperative robot soccer problem,” in European Conference on
tiple models for optimization,” in International Conference on Parallel
algorithm with hierarchical execution,” in 2016 International Conference
pariion of level-wise hierarchical multi-population genetic algorithm,”
Journal of Information and Telecommunication, vol. 1, no. 1, pp. 66–78,
2017.
algorithms,” in International Conference on Parallel Problem Solving
tonomous controllers for unmanned aerial vehicles using multi-objective
genetic programming,” in Cybernetics and Intelligent Systems, 2004
of robotic controllers for complex tasks,” in 2012 IEEE International
Conference on Development and Learning and Epigenetic Robotics
[16] ——, “A layered approach to learning client behaviors in the robocup
for symbolic regression problems,” Applied Soft Computing, vol. 62,
problems,” in European Conference on Genetic Programming. Springer,
mental erosion of residual error,” in Proceedings of the Companion
Publication of the 2015 Annual Conference on Genetic and Evolutionary
oopeng, a c++ genetic algorithm library.; in 2017 IEEE International
Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2017,
pp. 2051–2056.