Pyramid: A Hierarchical Approach to Scaling Down Population Size in Genetic Algorithms

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 increasingly more simplified/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

This work is supported by Lero, the Irish Software Research Centre, and the Science Foundation of Ireland.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

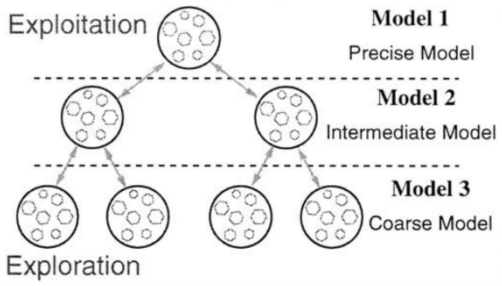


Fig. 1. HGA represented as a binary tree with three levels by [8]. Each level has a different fitness function and each node is a separate GA.

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.

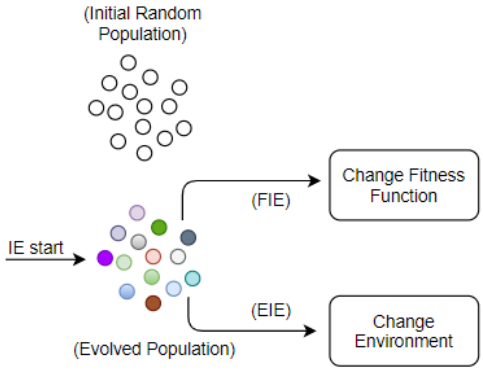


Fig. 2. IE requires an evolved population to start, where the difficulty of the population is increased by using a hard fitness function i.e., FIE or changing the environment for the population i.e., EIE.

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

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.

In [12], navigation controllers have been evolved for Unmanned Aerial Vehicle (UAV) using Multi-Objective GP. Both types of IE (FIE and EIE) were used. The goals of the UAV were to reach towards the radar and circle around it, with the path taken as efficient as possible with as few turns as possible. Four types of radars (two stationary and two mobile) were used. IE produced more successful runs and more successful controllers than direct evolution.

GP was applied to a tracking task which attempts to keep a moving object centered in the field of view of a camera mounted on a pan-tilt unit, in [13], where they employed a form of IE technique. The initial population is first trained to position the camera when velocity and object depth are held constant, before the resulting population is trained for various initial positions and object depths while velocity remains constant. Finally, the resulting population may be trained on the complete problem. Another control problem [14] used IE to divide the goal task in into several sub-tasks so that a controller for each task could be easily evolved. The sub-controllers were then combined through an additional evolutionary step to create a hierarchy of the single goal controller.

C. *Layered Learning (LL)*

A bottom-up approach is employed by LL, where learning achieved at the lower layers helps facilitate learning required

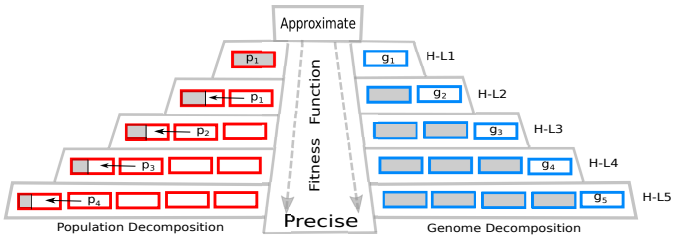


Fig. 3. Graphical representation of the Pyramid approach. On the left side we see the population size decreasing, while on the right, the genome length increases. The fitness function gets increasingly more precise as we descend.

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 ours. However, unlike most, in which the lower levels of the pyramid feed into the higher levels, we instead start at the top of the pyramid and work our way down, inspired in part by the Mayan Pyramid of Chichen-Itza in Mexico, which has the unique property that on each equinoxe, sun light creates the illusion of 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.

Algorithm 1: Pyramid

Input: The given problem and the solution representation.
Output: Problem decomposition and scaling down population size.

```
/* The solution representation:                                   */
genome = [g_1, g_2, ..., g_n]
/* Decompose the genome into k blocks                             */
B_1 ← [g_1, g_2, ..., g_m]
B_k ← [g_{m+1}, g_{m+2}, ..., g_n]
genome ← [B_1, ..., B_k]
/* Initial population using first block                           */
I_i ← rand(B_1)          // i = 1, ..., p, and p = popsize
/* Loop to add each B_j block in the genome                       */
while j < k              // j = 1, ..., k do
    /* Assign fitness to each individual                          */
    F_i ← fitness(I_i)
    /* If full genome, then get best individual                   */
    if j=k               // full genome then
        I_best ← max(fitness(I_i))
        break            // If stop condition is met
    /* Otherwise, keep scaling up genome                          */
    else
        /* Average fitness from pop                               */
        F̄ ← mean(F)
        /* Count how many individuals get top fitness             */
        if F(I_i) ≥ α · F̄
         // α is a fitness threshold then
            c ← c + 1    // c is a counter
        /* Promote top individuals to next level                  */
        if c ≥ β · p     // β is a pop threshold then
            /* update genome adding random genes at the end       */
            I_i ← [B_j , rand(B_{j+1})]
```

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: $g_1, g_2, ..., g_n$, 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 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: $B_1, ..., B_k$. For instance, $B_1 = [g_1, g_2, ..., g_m]$, 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 $I_i = rand(B_1)$.

The next step, as with GA conventional approaches, is to evaluate the population, assigning a fitness score to each individual $F_i = fitness(I_i)$, and getting new populations through standard genetic operations (crossover and mutation).

Fig. 4. Different approaches to decomposing a problem with Pyramid. On the left side, the conventional approach with just one level, in the middle, H-L3, which splits the solution representation into three levels, and on the right, a H-L5 Pyramid.

Fig. 5. A five level hierarhcy (H-L5) using different thresholds. The Pyramid on the left uses a threshold which considers more individuals, while at the far right, fewer are considered. It can be noted that by changing the threshold changes the Pyramid slope: the greater the slope, the faster the algorithm.

In the learning process using Pyramid, each time an individual scores α (see below for details) times better than the average fitness $\bar{F}$ of the current population, it is promoted to belong to the next population of individuals. When this set reaches a pre-specified number, β (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 α value mentioned above. In addition, we needed to establish how many individuals should be promoted (**RQ3**); this is the β 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

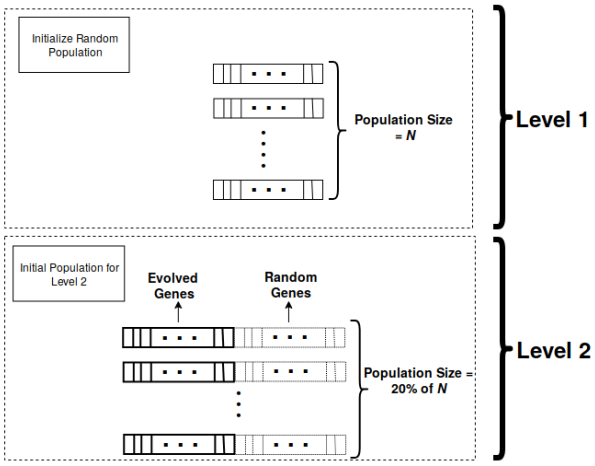


Fig. 6. A two level example of Pyramid: At each upper-level, population size reduces and the genome length increases.

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

TABLE I
CHOOSEN BENCHMARK FUNCTIONS

Name	Function
Sphere	$\sum_{i=1}^{n} x_i^2$
Rosenbrock	$\sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
Rastrigin	$10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$
Griewank	$1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$

TABLE II
PARAMETERS FOR PYRAMID: THE TARGET GENOME LENGTH FOR EACH H-LX IS 30, WHICH IS DIVIDED INTO DIFFERENT VALUES ACCORDING TO SPECIFIC EXPERIMENT. THE VALUES FOR CHROMOSOME LENGTH THAT SHOW A CALCULATION REFER TO THE PREVIOUS LENGTH PLUS THE ADDITIONAL LENGTH, E.G. 20+10=30 MEANS THE PREVIOUS LENGTH WAS 20 AND WE ARE ADDING 10 MORE TO GIVE A LENGTH OF 30.

Level		Chrom-length	Pop-size	Max-Gener
Baseline		30	5,000	3,000
H-L2	L1	15	5,000	1,500
	L2	15+15=30	1,000	1,500
H-L3	L1	10	5,000	1,000
	L2	10+10=20	1,000	1,000
	L3	20+10=30	200	1,000
H-L4	L1	7	5,000	700
	L2	7+8=15	1,000	800
	L3	15+7=22	200	700
	L4	22+8=30	40	800
H-L5	L1	6	5,000	600
	L2	6+6=12	1,000	600
	L3	12+6=18	200	600
	L4	18+6=24	40	600
	L5	24+6=30	8	600

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

TABLE III
RESULTS FROM THE **RASTRIGIN** FUNCTION. **+** INDICATES PYRAMID PERFORMED STATISTICALLY SIGNIFICANTLY BETTER, **-** THAT PYRAMID PERFORMED STATISTICALLY SIGNIFICANTLY WORSE, AND ***** THAT THERE IS NO STATISTICALLY SIGNIFICANT DIFFERENCE.

	Individuals Processed		***Best Fitness***		
	Maximum	Actual	Avg	std	p-value
Baseline	15000000	1397227	0.0183	0.0032	- - -
H-L2	900000	460469	0.0068 +	0.0015	<**0.01**
H-L3	6200000	119372	0.0184 *	0.0061	0.9193
H-L4	4472000	51021	0.1026 -	0.0303	<**0.01**
H-L5	3964800	44356	0.1756 -	0.0581	<**0.01**

TABLE IV
RESULTS FROM THE **GRIEWANK** FUNCTION. IN THIS CASE, THE RESULTS PERFORM STATISTICALLY SIGNIFICANTLY WORSE.

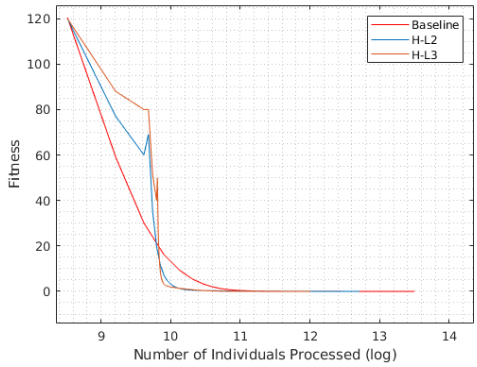
	Individuals Processed		***Best Fitness***		
	Maximum	Actual	Avg	std	p-value
Baseline	15000000	6517491	1.0012	0.0784	- - -
H-L2	900000	1369560	1.8770 -	0.5976	<**0.01**
H-L3	6200000	176310	6.3616 -	1.9758	<**0.01**
H-L4	4472000	30946	23.0932 -	0.0784	<**0.01**
H-L5	3964800	91126	37.9472 -	12.6262	<**0.01**

TABLE V
RESULTS FROM THE **SPHERE** FUNCTION. AS WITH TABLE III, **+** INDICATES PYRAMID PERFORMED STATISTICALLY SIGNIFICANTLY BETTER, **-** THAT PYRAMID PERFORMED STATISTICALLY SIGNIFICANTLY WORSE, AND ***** THAT THERE IS NO STATISTICALLY SIGNIFICANT DIFFERENCE.

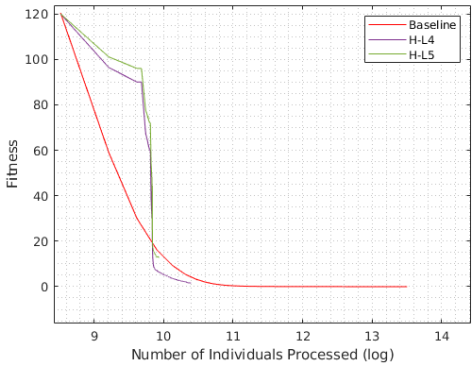
	Individuals Processed		***Best Fitness***		
	Maximum	Actual	Avg	std	p-value
Baseline	15000000	150242	1.7937	0.1458	- - -
H-L2	900000	37049	0.5644 +	0.0523	<**0.01**
H-L3	6200000	36680	0.3547 +	0.0867	<**0.01**
H-L4	4472000	34892	5.8534 -	2.7289	<**0.01**
H-L5	3964800	30375	15.9313 -	5.6826	<**0.01**

TABLE VI
RESULTS FROM THE **ROSENBROCK** FUNCTION. **-** THAT PYRAMID PERFORMED STATISTICALLY SIGNIFICANTLY WORSE WHILE ***** THAT THERE IS NO STATISTICALLY SIGNIFICANT DIFFERENCE.

	Individuals Processed		***Best Fitness***		
	Maximum	Actual	Avg	std	p-value
Baseline	15000000	134592	1.224	0.5709	- - -
H-L2	900000	59874	1.241 *	0.1234	0.21
H-L3	6200000	43915	1.453 -	0.2006	<**0.01**
H-L4	4472000	30946	1.644 -	0.3936	<**0.01**
H-L5	3964800	34201	2.089 -	0.5815	<**0.01**

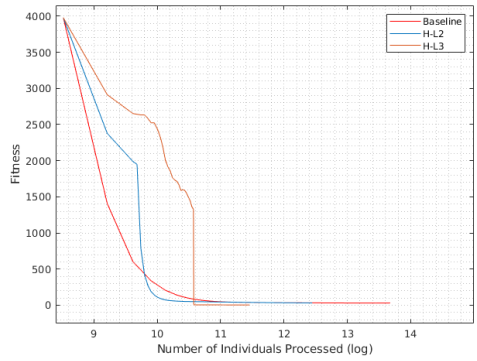


(a) Baseline vs. H-L2 and H-L3

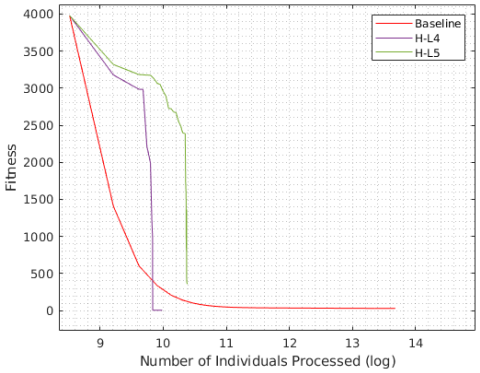


(b) Baseline vs. H-L4 and H-L5

Fig. 7. Sphere Results: Comparison of Baseline with each H-LX. Clearly, Pyramid can the same fitness but with fewer individuals processed.

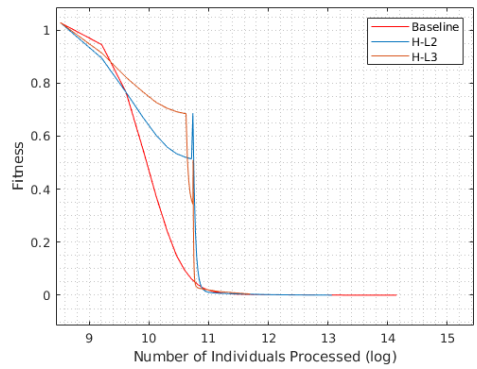


(a) Baseline vs. H-L2 and H-L3

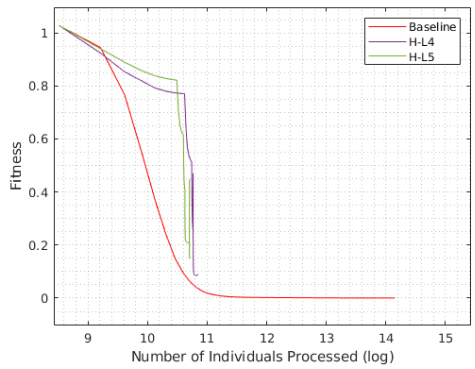


(b) Baseline vs. H-L4 and H-L5

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.



(a) Baseline vs. H-L2 and H-L3

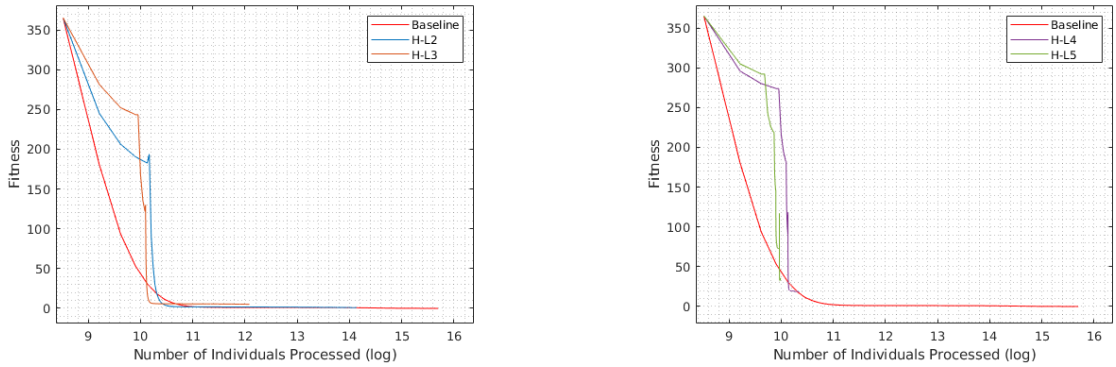


(b) Baseline vs. H-L4 and H-L5

Fig. 9. Rastrigin Results: Comparison of Baseline with each H-LX. Both H-L2 and HL-3 outperform the baseline with fewer individuals processed, with H-L3 processing 91.5% individuals less.

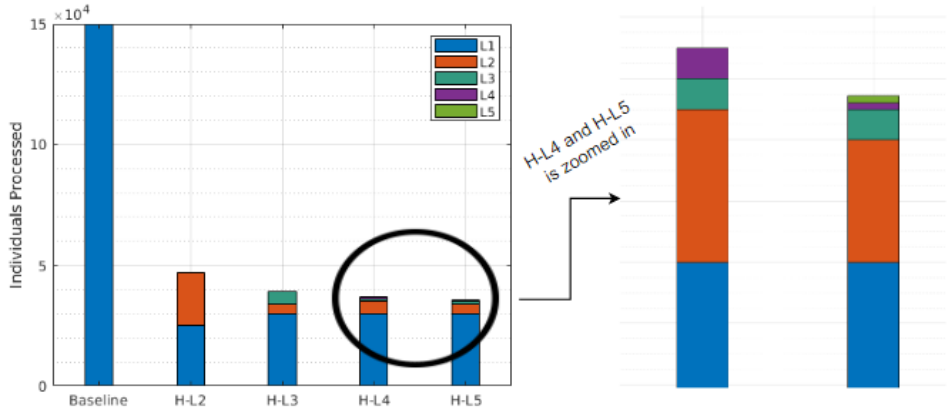
beyond HL-4 for any of the experiments.

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 α and β values to control when individuals

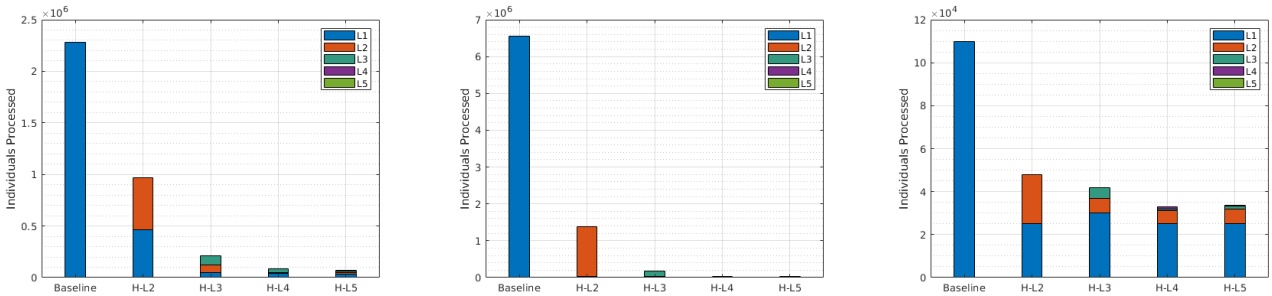


(a) Baseline vs. H-L2 and H-L3 (b) Baseline vs. H-L4 and H-L5

Fig. 10. Griewank Results: Comparison of Baseline with each H-LX. In this case, the baseline does perform better, indicating there is scope for modifying the 20% figure used for triggering the next level.



(a) **Sphere**



(b) **Rastrigin** (c) **Griewank** (d) **Rosenbrock**

Fig. 11. Individuals processed for each Level Experiment. There is clearly a significant reduction in number of individuals processed. For Sphere, we have zoomed in the H-L4 and H-L5 in order to make it more clear, as the individuals processed are too low that it is hard to see them. For Griewank, at H-L4, and H-L5, very low number of individuals are processed that the bar graph seems invisible.

move to the next level; clearly there is scope to optimize this.

A summary of the individual evaluations is shown in Table VII, where the baseline is 100% and $A\%$ stands for the actual percentage of evaluation of the current level H-LX, and $R\%$ stands for the percentage of reduction.

Figures 11 shows the number of individuals for each problem.

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

TABLE VII
INDIVIDUALS EVALUATION REDUCTION. FIGURES IN BOLD REPRESENT RUNS IN WHICH PYRAMID WAS PRODUCED EITHER THE SAME OR BETTER RESULTS THAN THE BASELINE.

Function	H-L2		H-L3		H-L4		H-L5	
	A%	R%	A%	R%	A%	R%	A%	R%
Sphere	**24.6**	**75.3**	**24.4**	**75.6**	23.2	76.8	20.2	79.8
Rosenbrock	**44.5**	**55.5**	32.6	67.4	23.0	77.0	25.4	74.6
Rastrigin	**32.9**	**67.1**	**8.5**	**91.5**	3.6	96.4	3.1	96.9
Griewank	21.0	79.0	2.7	97.3	0.5	99.5	1.4	98.6

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 α 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

[1] J. R. Koza and J. R. Koza, Genetic programming: on the programming of computers by means of natural selection. MIT press, 1992, vol. 1.
[2] P. J. Angeline and J. Pollack, "Evolutionary module acquisition," in Proceedings of the second annual conference on evolutionary programming. Citeseer, 1993, pp. 154–163.
[3] W. Banzhaf, D. Banscherus, and P. Dittrich, Hierarchical genetic programming using local modules. Secretary of the SFB 531, 1999.
[4] T. X. Chen, "Problem decomposition-based scalable macro-evolutionary algorithms," in Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546), vol. 1. IEEE, 2001, pp. 223–231.
[5] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," IEEE Transactions on evolutionary computation, vol. 11, no. 6, pp. 712–731, 2007.
[6] T. Yu and C. Clack, "Polygp: A polymorphic genetic programming system in haskell," Genetic Programming, vol. 98, 1998.
[7] S. M. Gustafson and W. H. Hsu, "Layered learning in genetic programming for a cooperative robot soccer problem," in European Conference on Genetic Programming. Springer, 2001, pp. 291–301.
[8] M. Sefrioui and J. Périaux, "A hierarchical genetic algorithm using multiple models for optimization," in International Conference on Parallel Problem Solving from Nature. Springer, 2000, pp. 879–888.
[9] T.-P. Hong, Y.-C. Peng, and W.-Y. Lin, "Multi-population genetic algorithm with hierarchical execution," in 2016 International Conference on Fuzzy Theory and Its Applications (iFuzzy). IEEE, 2016, pp. 1–4.
[10] T.-P. Hong, Y.-C. Peng, W.-Y. Lin, and S.-L. Wang, "Empirical comparison of level-wise hierarchical multi-population genetic algorithm," Journal of Information and Telecommunication, vol. 1, no. 1, pp. 66–78, 2017.
[11] E. D. de Jong, D. Thierens, and R. A. Watson, "Hierarchical genetic algorithms," in International Conference on Parallel Problem Solving from Nature. Springer, 2004, pp. 232–241.
[12] G. J. Barlow, C. K. Oh, and E. Grant, "Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming," in Cybernetics and Intelligent Systems, 2004 IEEE Conference on, vol. 2. IEEE, 2004, pp. 689–694.
[13] J. F. Winkeler and B. Manjunath, "Incremental evolution in genetic programming," Genetic Programming, pp. 403–411, 1998.
[14] M. Duarte, S. Oliveira, and A. L. Christensen, "Hierarchical evolution of robotic controllers for complex tasks," in 2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL). IEEE, 2012, pp. 1–6.
[15] P. Stone and M. Veloso, "Layered learning," in European Conference on Machine Learning. Springer, 2000, pp. 369–381.
[16] ——, "A layered approach to learning client behaviors in the robocup soccer server," Computer Science, vol. 412, pp. 268–7123, 1997.
[17] S. S. M. Astarabadi and M. M. Ebadzadeh, "A decomposition method for symbolic regression problems," Applied Soft Computing, vol. 62, pp. 514–523, 2018.
[18] D. Jackson and A. P. Gibbons, "Layered learning in boolean gp problems," in European Conference on Genetic Programming. Springer, 2007, pp. 148–159.
[19] D. Medernach, J. Fitzgerald, R. Azad, and C. Ryan, "Wave: Incremental erosion of residual error," in Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. ACM, 2015, pp. 1285–1292.
[20] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, and S. Nahavandi, "openga, a c++ genetic algorithm library," in 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2017, pp. 2051–2056.
[21] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimization problems," arXiv preprint arXiv:1308.4008, 2013.