A STUDY OF GENETIC PROGRAMMING VARIABLE POPULATION SIZE FOR DYNAMIC OPTIMIZATION PROBLEMS

Leonardo Vanneschi
Department of Informatics, Systems and Communication (D.I.S.Co.), University of Milano-Bicocca, Milan, Italy

Giuseppe Cuccu
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Lugano, Switzerland

Keywords: Genetic programming, Variable size population, Dynamic optimization.

Abstract: A new model of Genetic Programming with variable size population is presented in this paper and applied to the reconstruction of target functions in dynamic environments (i.e. problems where target functions change with time). The suitability of this model is tested on a set of benchmarks based on some well known symbolic regression problems. Experimental results confirm that our variable size population model finds solutions of similar quality to the ones found by standard Genetic Programming, but with a smaller amount of computational effort.

1 INTRODUCTION

Many real-world problems are anchored in dynamic environments, where some element of the problem domain, typically the target, changes with time. For this reason, developing solid evolutionary algorithms (EAs) to reliably solve these problems is an important task. In the last few years, many contributions have appeared which studied dynamic optimization environments and developped new evolutionary frameworks for solving them. Nonetheless, the majority of those approaches are based on Genetic Algorithms (GAs) (Goldberg, 1989) or Particle Swarm Optimization (PSO) (Clerc, 2006) and the problem objective is to find the extrema (maxima or minima) of a target function that changes with time. On the other hand, very few contributions have appeared to date that study the ability of Genetic Programming (GP) (Koza, 1992) to reconstruct target functions on dynamic optimization environments.

In this paper we hypothesize that variable size population GP is a promising method for dynamic optimization problems. This idea is not new in evolutionary computation; for instance, it has been applied to PSO in (Fernandes et al., 2005). However, it has never been applied to GP before. We propose a variable size population GP model called *DynPopGP*. It is inspired by the one presented in (Tomassini et al., 2004). Simply speaking, it works by shrinking the population when fitness is improving and increasing its size, by adding new genetic material, when the evolution stagnates. Our hypothesis is that when the target function changes, evolution of the current population should stagnate. Thus the evolution should benefit from the creation of new genetic material, that should give the necessary amount of diversity to start the optimization of a new target function.

This paper is structured as follows: in Section 2 we discuss previous contributions in dynamic optimization. In Section 3 we discuss the reasons why it is not suitable to directly apply the GP model presented in (Tomassini et al., 2004) to dynamic optimization and we present *DynPopGP* that extends it. Section 4 contains a description of the test problems and presents the experimental setting used in this paper. In Section 5 we show the obtained experimental results. Finally, Section 6 concludes the paper and suggests directions for future research.

2 DYNAMIC OPTIMIZATION

Over the past few years, a number of authors have addressed the problem of EAs premature loss of diversity in dynamic environments in many different ways.

Vanneschi L. and Cuccu G. (2009).
A STUDY OF GENETIC PROGRAMMING VARIABLE POPULATION SIZE FOR DYNAMIC OPTIMIZATION PROBLEMS.
In *Proceedings of the International Joint Conference on Computational Intelligence*, pages 119-126
DOI: 10.5220/0002314701190126

Surveys of these studies can be found for instance in (Branke, 2001; Branke, 2003). More recent contributions include (Yang, 2004) where, based on the concept of problem difficulty, a new dynamic environment generator using a decomposable trap function is proposed; (Huang and Rocha, 2005), where a coevolutionary agent-based model is used; (Rand and Riolo, 2005) where the use of mutation for diversity maintenance is investigated and (de França et al., 2005), where the use of artificial immune networks for multimodal function optimization on dynamic environments is studied.

All the above quoted contributions treat the problem of tracking the extrema in a dynamic environment, where the target function changes with time and concern GAs, PSO or other EAs variants. Very few contributions have appeared to date dealing with the (more complex) problem of approximating/reconstructing target functions that change with time by means of GP. Noteworthy recent exceptions are: (Dempsey, 2007), where financial time series (index closing price data) are reconstructed by means of Grammatical Evolution, and (Tanev, 2007) where an approach for incorporating learning probabilistic context-sensitive grammar in GP is employed for the evolution and adaptation of locomotion gaits of a simulated snake-like robot. Nevertheless, both these approaches use Grammar-Based GP and employ it for very particular and complex applications.

The goal of this paper is different: first of all, we want to study standard tree-based GP (Koza, 1992), and one variant thereof using variable size populations; secondly, we want to present and employ (here for the first time) more simple, and thus easier to study, test problems. The proposed GP framework is presented in Section 3 and the used test functions in Section 4.

3 VARIABLE SIZE POPULATION GP

In 2003, an idea for counteracting the negative effects of *bloat* (Banzhaf and Langdon, 2002; Poli et al., 2008) and of premature convergence (Burke et al., 2002) on GP was presented. It consisted in reducing the size of populations at a linear rate (Fernández et al., 2003c; Fernández et al., 2003b). This was achieved by removing a fixed number of individuals at each generation. This technique was called *plague* and it has been shown to have some positive effects on GP systems. That idea started from the observation of a general behavior of GP over a wide set of problems: normally fitness improves quickly at the beginning of GP runs and, after a number of generations, improvements are more difficult to obtain. In this second phase, plagues allow to save computational effort, that would be wasted otherwise, since it does not bring appreciable advantages.

On the other hand, it is clear that even if a considerable amount of computational effort is saved, the blind deletion of individuals at each generation probably cannot lead to the discovery of better individuals than the ones found by the standard GP process. Furthermore, steadily decreasing populations produce a progressive loss of diversity, especially at the genotypic level. For this reason, in (Tomassini et al., 2004), an extension of the plague technique aimed at varying the population size in an intelligent way during the execution of each GP run, was presented. In that model, adds and suppressions of individuals are operated dynamically on the basis of the behavior of the GP system: population size is decreased while the algorithm is progressing (i.e. fitness is improving) and it is increased when the algorithm reaches the stagnation phase. In this way, when the algorithm is progressing, as much computational effort as possible is saved and this previously saved effort is spent only when it is really useful, i.e. when the algorithm is stagnating and new genetic material is needed. In (Tomassini et al., 2004) the decision whether to shrink or inflate the population was taken on the basis of the relationship between the best fitness value in the population at the current generation g (bf_g) and the one at the previous generation (bf_{g-1}). This value was stored in a variable the authors called *pivot*. Two versions of *pivot* are presented in (Tomassini et al., 2004): in the first one *pivot*$= \Delta_{g-1}/\Delta_g$ and in the second one *pivot*$= \Delta_{g-1} - \Delta_g$, where $\Delta_g = bf_{g-1} - bf_g$. The GP model using the first version of pivot was called DIV, while the one using the second version was called SUP in (Tomassini et al., 2004).

In Section 3.1 we discuss the reasons why DIV and SUP are not suitable to solve dynamic optimization problems and in Section 3.2 we present our new variable size population GP model, called *DynPopGP*, that extends DIV and SUP.

3.1 DIV and SUP in Dynamic Environments

Both the DIV and SUP methods introduced in (Tomassini et al., 2004) have the following characteristics:

(i) The decision on whether to shrink or inflate the population is taken only on the basis of the relationship between the best fitness values at the current generation and at the previous one. This decision does *not* depend on how good those fitness values are. In other words, this decision is the same independently from the fact that GP has found good solutions or bad ones.

(ii) The quantity of individuals that have to be added to or suppressed from the population depends on the current population size (in both DIV and SUP when individuals have to be suppressed, 1% of the population is suppressed and when they need to be added, a number of individuals equal to the 0.2% of the population is added). Thus, additions and suppressions are more violent when the population is large.

A consequence of point (i) is that, when applied to dynamic optimization, DIV and SUP behave exactly the same in case the algorithm stagnates on a particular target function (but the target function remains the same) and in case the target function changes. However, when the target function changes, and in particular if the new target function is "different enough" from the old one, we expect a more violent worsening in fitness than when the algorithm stagnates on a fixed target function. In particular, if we use an elitist algorithm (i.e. we copy the best, or a pool of good individuals, unchanged in the next population at each generation), the best fitness in the population cannot worsen if the target function stays the same. It can only worsen if the target function has changed (because the best individual at the previous generation may change its fitness). The algorithm we propose (*DynPopGP*) behaves in two different ways in these two different situations.

A consequence of point (ii) is that, when the target function changes, the population size may grow indefinitely. In fact, suppose DIV or SUP are optimizing a given target function and are in a stagnation phase. Then, the population keeps growing. Now, assume the target function changes. The population will continue growing up until the new genetic material necessary to optimize the new target function has been created. *DynPopGP* solves this problem by defining a new function to quantify the amount of individuals that have to be added or deleted from a population.

For these reasons, we do not consider the DIV and SUP models any longer in this paper. A detailed experimentation showing the practical advantages of *DynPopGP* compared to DIV and SUP is definitely needed in the future.

3.2 New Variable Size Population Model

The *DynPopGP* algorithm we propose can be summarized by the pseudo-code in Figure 1, where we consider minimization problems (i.e. small fitness values are better than large ones)[1]. This algorithm uses a number of parameters (*trg_fit*, *old_got_trg*, *new_got_trg*, *stand_by_size*) and functions (*update_pop_size*, Δ_{pop}) that we describe below. Empirical values for these parameters, coming from a set of preliminary experiments, have been used in this work. More detailed sensitivity analysis on these parameters definitely deserves to be conducted in the future.

```
begin
  Generate a population of N random individuals;
  best = best individual in the population;
  old_got_trg = false;
  for g := 1 to maxgen do
     new_got_trg = (fitness(best) ≤ trg_fit);
     if (not new_got_trg)
       then
         elitism (i.e. copy of the best);
         selection;
         reproduction / crossover;
         mutation;
         best = best individual in the new population;
         new_got_trg = (fitness(best) ≤ trg_fit);
         if (old_got_trg)
            then
              // The old best had reached the target, while
              // the new best has not reached it:
              // the target function has surely changed.
              // Set the population size to the initial size
              update_pop_size(N - current_pop_size);
            else
              // Neither the new best, nor the old best
              // have reached the target: update the
              // population size using the Δpop function
              update_pop_size(Δpop());
         endif
       else
          if (not old_got_trg)
          then
            // The new best has reached the target, while
            // the old best had not reached it. This means
            // that the target has been found now.
            // I have to spend as few computational effort
            // as possible until the target function changes
            // (or the process terminates).
            // I set the population size to a prefixed
            // "stand-by" value
       update_pop_size(stand_by_size - current_pop_size);
          endif
       endif
       old_got_trg = new_got_trg;
  endfor
end
```

Figure 1: Pseudo-code for the *DynPopGP* algorithm.

[1]The authors are aware that in case of minimization problems the term *fitness* is rather incorrect. Nevertheless, they keep using it for simplicity.

trg_fit represents a fitness value (target) that approximates the optimum. In this work, we have used a value equal to 0.01.

old_got_trg (respectively *new_got_trg*) is a boolean variable whose value is *true* if the best fitness in the population at the previous (respectively current) generation is better than or equal to the target and *false* otherwise.

stand_by_size is a small value of the population size that is used when the optimum of a target function has been approximated in a satisfaisable way, and we have to wait for the target function to change. In this case, the population has to be as small as possible, so that we can save computational effort. In our work, we wanted to set this value as a function of the initial population size. We have chosen a value of *stand_by_size* equal to the initial population size divided by 4 because, by means of a set of experiments, we have seen that this value represents a good compromise between saving computational effort (population shrinking) and keeping some good genetic material in the population.

$update\_pop\_size(x)$ is a function that adds $|x|$ individuals to the population if x is positive and suppresses $|x|$ individuals from the population if x is negative. When $|x|$ individuals have to be suppressed, the population is sorted. The $2 \cdot x$ worst (in terms of fitness) individuals in the population are considered and, among these individuals, the x largest ones (in terms of number of tree nodes) are suppressed (as it happened for the DIV and SUP algorithms). When $|x|$ individuals have to be added, they are randomly generated with the same initialization method that is used at the beginning of the GP run (ramped half-and-half in this work).

$\Delta_{pop}()$ is a function that returns the number of individuals that have to be to be added or suppressed from the population when neither the old best fitness value nor the new one approximate the optimal fitness value in a satisfaisable way. We want $\Delta_{pop}()$ to be a function of the current best fitness in the population and the current population size, which are the two basic principles that were not taken into account by the DIV and SUP algorithms. For doing this, we define two new functions: *best_fit_contribution* and *pop_size_contribution* and we multiply their returned values. We want the result of this multiplication to be immediately interpretable by a human, so we impose that the results of *best_fit_contribution* and *pop_size_contribution* belong to the range $[1,10]$. In this way, their product belongs to the range $[1,100]$ and it can be interpreted, for instance, as a percentage (which represents the respective contributions given by the best fitness value and the population size). The $\Delta_{pop}()$ function performs the following calculation:

$$\Delta_{pop}() = pivot \cdot strength \cdot best\_fit\_contribution() \cdot pop\_size\_contribution()$$

where:

pivot is a variable whose value is -1 if the best fitness in the population at the current generation is better then the one at the previous generation and $+1$ otherwise (in practice, the value of *pivot* determines if individuals have to be added or suppressed).

strength is a variable that determines how strong populations inflate and deflate have to be at each step, and it is used to rescale the value of $best\_fit\_contribution() \cdot pop\_size\_contribution()$. In this work, we use a value equal to 0.3. In this way, the maximum number of individuals that can be added to or suppressed from the population is 30 (given that the maximum possible value of $best\_fit\_contribution() \cdot pop\_size\_contribution()$ is 100). In fact, experimental evidence confirms that adding more than 30 individuals at a time to the population eccessively increments the computational effort without a corresponding gain in the quality of the generated solutions.

The $best\_fit\_contribution()$ function determines the contribution given to the $\Delta_{pop}()$ by the best fitness value reached. As we have said above, we want this function to return a value in the range $[1,10]$. Furthermore, we want it to return 10 (maximum contribution to the $\Delta_{pop}()$) when the best fitness in the population is bad (fitness above a certain threshold, 60 in this work) and to return the minimum value when the best fitness in the population approximates the optimum in a satisfaisable way. The easier way to obtain this, is to define the $best\_fit\_contribution()$ as a linear function, for instance a straight line, that intersects the points (trg_fit, min_coeff) and (max_fit, max_coeff) where min_coeff is equal to 1, max_coeff is equal to 10 and max_fit is equal to 60. So, the $best\_fit\_contribution()$ function is defined by the pseudo-code in Figure 2.

best_fit_contribution() ::
  if (fitness(*best*) $\leq$ *trg_fit*) **then return** *min_coeff*;
  elsif (fitness(*best*) $\geq$ *max_fit*) **then return** *max_coeff*;
  else return
    $max\_coeff - min\_coeff \cdot \frac{fitness(best) - trg\_fit}{max\_fit - trg\_fit} + min\_coeff$
  endif

Figure 2: Pseudo-code for the *best_fit_contribution* function.

Finally, the $pop\_size\_contribution()$ function determines the contribution to the $\Delta_{pop}()$ given by the current population size. Analogously to the *best_fit_contribution*, we have used a linear function that returns the maximum possible value (10 in this work) when the current population size is minimal (i.e. it is equal to *stand_by_size*, that has been set to the initial population size divided by 4 in this work) and the minimum possible value (0 in this work) when the current population size is maximal (i.e. smaller or equal to *stand_by_size*).

4 TEST PROBLEMS AND EXPERIMENTAL SETTING

Some benchmark problems have been defined for testing the performances of optimization methods in dynamic environments. In particular, Branke (Branke, 2001; Branke, 2003) defines and uses *moving peaks* types of functions. In these benchmarks, hand-tailored fitness landscapes are defined and the positions of the extrema and their basins of attraction are modified with time. Similar problems are also used, for instance, in (Huang and Rocha, 2005; Rand and Riolo, 2005; de França et al., 2005).

However, this type of benchmark is not suitable for the present study. In fact, in this work, we want to study the ability of GP to reconstruct dynamic target functions and not follow moving extrema. With this goal in mind, it would make no sense to use moving peaks benchmarks as the ones presented in (Branke, 2001; Branke, 2003), given that, in those kinds of benchmark, extrema are moved by changing some additive or multiplicative constants to a (otherwise not changing) target function. If one uses GP with linear scaling (introduced in (Keijzer, 2003)), the moving peaks problem reduces to a static GP problem, given that linear scaling allows to reconstruct the shape of the target functions, offering a method to automatically determine additive and multiplicative constants.

For this reason, in this paper we define a new set of benchmark problems that can be used to test GP ability to reconstruct target functions in dynamic environments. These benchmarks are symbolic regression problems inspired by (Keijzer, 2003). In particular, maintaining the same terminology as in (Keijzer, 2003), we have considered test functions F_{12}, F_{13}, F_{14}, F_{15} and F_{16} (presented in (Keijzer, 2003) at page 9) and we have used them to build dynamic test problems in which the importance of the modification of the target function can be tuned. Even though presented in (Keijzer, 2003), we also report here the equations for these functions:

- $F_{12}(x,y) = xy + sin((x-1)(y-1))$
- $F_{13}(x,y) = x^4 - x^3 + y^2/2 - y$
- $F_{14}(x,y) = 6\ sin(x)\ cos(y)$
- $F_{15}(x,y) = 8/(2 + x^2 + y^2)$
- $F_{16}(x,y) = x^3/5 - y^3/2 - y - x$

As in (Keijzer, 2003), for all these functions the fitness cases are created by generating 20 random values (with uniform distribution) for x and y in the range $[-3,3]$.

We are aware that these test functions are bidimensional and thus do not represent real-life applications (typically characterized by many features and thus multi-dimensional), nevertheless, as reported in (Keijzer, 2003) at page 8: "The aim of this set of experiments is to demonstrate the practical implications of the use of the [method] studied here. Being of low dimensionality does not make the problems easy however. Many of the problems above mix trigonometry with polynomials, or make the problems in other ways highly non-linear".

Using these test functions, we have built three benchmarks for dynamic optimization that we have called BENCH1, BENCH2 and BENCH3. The target function at each generation is calculated by the algorithm in Figure 3, where given a test function F_i, with $12 \le i \le 15$ $succ(F_i) = F_{i+1}$ and $succ(F_{16}) = F_{12}$. The difference between these benchmaks is in the cardinality of the set of functions F used for calculating the target: for BENCH1, F contains two functions. These functions are F_{12} and F_{13} at generation 1. The target value is calculated performing the sum of these two functions for each couple of points (x,y). At each *period* generations, one of the two functions changes (i.e. it is deleted from set F and replaced by another function), while the other stays the same, in a cyclic way so that all the test functions are used.

begin
  Define a set of test functions $F = \{f_1, f_2, ..., f_n\}$
  for $g := 1$ **to** *maxgen* **do**
    For each fitness case (x,y), the target value is:
$$\sum_{i=1}^{n} f_i(x,y)$$
    if (g **mod** *period* = 0) **then**
      $\forall 1 \le i \le n : f_i := succ(f_i)$
    endif
  endfor
end

Figure 3: Pseudo-code for target calculation in benchmark problems BENCH1, BENCH2 and BENCH3 The difference between these benchmark is in the size of set F: $n = 2$ for BENCH1; $n = 3$ for BENCH2 and $n = 4$ for BENCH3.

BENCH2 is like BENCH1, except that F contains 3 functions, that are F_{12}, F_{13} and F_{14} at generation 1 and at each *period* generations, one of them changes, while the other two stay the same.

BENCH3 is similar, except that F contains 4 functions, that are F_{12}, F_{13}, F_{14} and F_{15} at generation 1 and at each *period* generations, one of them changes, while the other three stay the same.

In this way, BENCH1 has the more violent target modifications at each *period* generations, BENCH3 has the less violent modifications, while BENCH2 is in an intermediary situation.

In this work, we have used a value of *period* equal to 20. The other parameters used are as follows: population size of 200 individuals; function set equal to $\{+,-,*,/\}$ (exactly the same method as in (Keijzer, 2003) has been used to avoid divisions with denominator equal to zero and thus to ensure operators closure); terminal set composed by two floating point variables and four ephemeal random constants; maximum tree depth for initialization equal to 6; maximum tree depth for crossover and mutation equal to 17; tournament size equal to 10; standard subtree crossover (Koza, 1992) applied with probability 0.9; standard subtree mutation (Koza, 1992) applied with probability 0.1; maximum number of generations equal to 100 (in this way, given that *period*= 20, the process stops when the target function returns the same as at generation 1); generational GP with elitism (i.e. copy of the best individual unchanged in the next population at each generation). Fitness is the root mean squared error (RMSE) between outputs and targets. All the results reported in the next section have been obtained by performing 100 independent runs of each GP model (standard GP and *DynPopGP*) for each banchmark. With standard GP we indicate the canonic (fixed size population) GP process (Koza, 1992).

5 EXPERIMENTAL RESULTS

In Figure 4 we report average best fitness values over 100 independent runs against generations for standard GP (*stdGP*) and *DynPopGP* for BENCH1 (Figure 4(a)), BENCH2 (Figure 4(b)) and BENCH3 (Figure 4(c)). This figure clearly shows that the two GP models find solutions of similar qualities at corresponding generations for all the three studied benchmarks (standard deviation error bars, not shown here for simplicity, confirm that the differences between the curves in Figure 4 are not statistically relevant). Seen from this perspective, the two GP models might seem equivalent. However, as reported for instance in (Fernández et al., 2003a), comparing the performances of two GP models against *generations* may lead to wrong conclusions, given that GP individuals have a variable size and thus evaluating a generation for the two models may request a very different amount of computational resources.

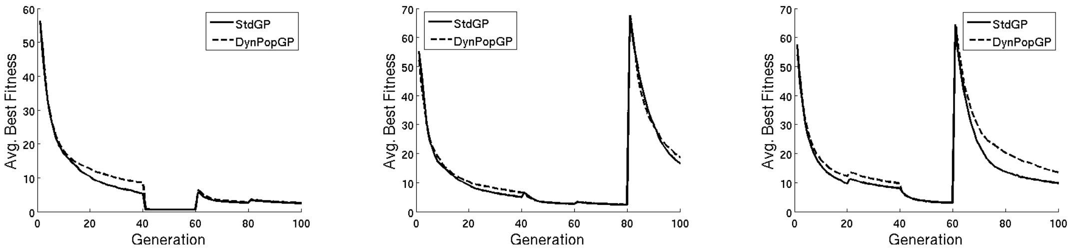
For this reason, in Figure 5 we report the values of the computational effort against generations (values averaged over the same 100 runs as in Figure 4). We have considered exactly the same definition of computational effort as in (Fernández et al., 2003a), i.e. the computational effort at a given generation g (E_g) is given by: $E_g = PE_g + PE_{g-1} + ... + PE_1$, where the partial effort at generation g (PE_g) is defined as the sum of the numbers of nodes of all the individuals in the population at generation g. Given that fitness calculation is often the most computationally expensive part of an EA and that in GP this calculation largely depends on the size of the individuals in the population, this measure clearly gives an idea of the computational complexity of executing a GP model (as claimed in (Fernández et al., 2003a)). Figure 5 shows that the effort spent by *DynPopGP* is smaller than the one spent by *stdGP* for all the three studied benchmarks. Standard deviations reported in figure as error bars seem to hint that these results are statistically significant.

Authors of (Fernández et al., 2003a) report results of the average best fitness against computational effort. We do the same in Figure 6, where it is clear that *DynPopGP* finds solutions of similar quality with a smaller computational effort than *stdGP*.

In Figure 7 we report the average population size at each generation (calculated using the same 100 runs as in the previous figures). We can see that the population size of *DynPopGP* is always smaller than the one of *stdGP* for all the three studied benchmarks. Nonetheless, we can notice that the population size of *DynPopGP*, tends to grow at each *period* generations (generation number multiple of 20), because of the modification in the target function. In some cases this growth begins slightly before the end of a period, probably because the particular target function had already been optimized and the stagnation phase was beginning. Another interesting thing to remark is that, after a first phase of population shrinking, which is common in the three benchmarks, the population growth is stronger for BENCH1 (which has the more violent target modifications) than for BENCH3 (which has the less violent target modifications), while the behavior of BENCH2 is intermediary. Furthermore, it is possible to see that for BENCH1 the population size continues to grow until the end of the run, while for BENCH2 and BENCH3 there is a new phase in which the population starts shrinking once again (at about generation 80 for BENCH2 and generation 60 for BENCH3).

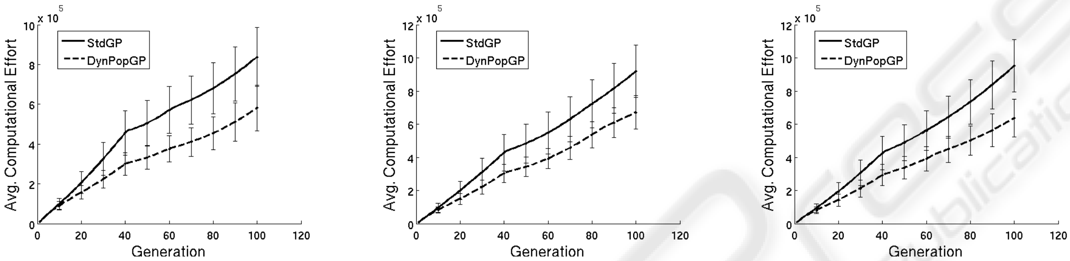
6 CONCLUSIONS

This paper investigates the usefulness of variable size population Genetic Programming (GP) on dynamic



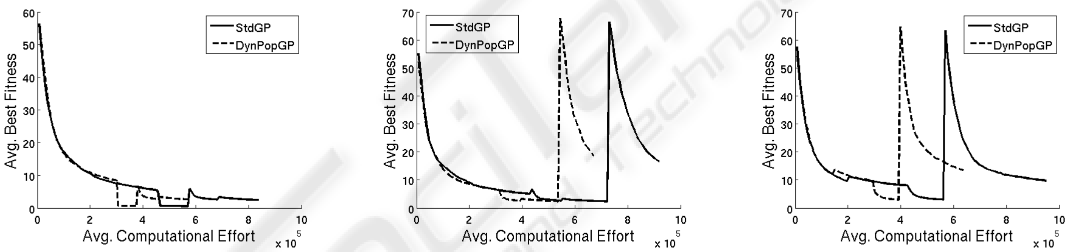
(a) (b) (c)

Figure 4: Average best fitness against generations for *stdGP* and *DynPopGP*. (a): BENCH1; (b): BENCH2; (c): BENCH3.



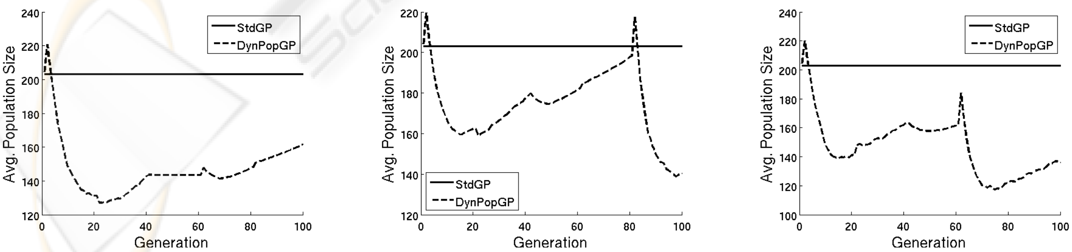
(a) (b) (c)

Figure 5: Computational effort against generations for *stdGP* and *DynPopGP*. (a): BENCH1; (b): BENCH2; (c): BENCH3;



(a) (b) (c)

Figure 6: Average best fitness against computational effort for *stdGP* and *DynPopGP*. (a): BENCH1; (b): BENCH2; (c): BENCH3.



(a) (b) (c)

Figure 7: Population size against generations for *stdGP* and *DynPopGP*. (a): BENCH1; (b): BENCH2; (c): BENCH3.

problems. The idea is not new in Evolutionary Computation (Fernandes et al., 2005), but, to the best of our knowledge, it had never been extended to GP before. In particular, we believe that a model inspired by the one presented in (Tomassini et al., 2004) should be suitable for this kind of problems.

Contributions of this paper are: first of all, we have motivated the fact that the GP model presented in (Tomassini et al., 2004), taken as it is, is not suitable for dynamic optimization problems. Successively, we have presented a GP model that extends the one introduced in (Tomassini et al., 2004) and that we candidate for suitably solving dynamic optimizaton problems. We have called that model *DynPopGP*. We have also defined a new set of benchmarks to test GP models for dynamic optimization, based on some symbolic regression problems used in (Keijzer, 2003). Finally, we have experimentally shown that *DynPopGP* allows GP to save computational effort compared to standard GP, while finding solutions of the same accuracy, at least for the studied benchmarks.

This work is clearly a first and preliminary step in this research track. The usefulness of GP (and in particular GP with variable size population) for dynamic optimization deserves further investigation. In particular, GP models have to be tested on hard real-life applications, typically characterized by a large number features and few samples and the issue of generalization to out-of-sample data deserves to be investigated.

REFERENCES

Banzhaf, W. and Langdon, W. B. (2002). Some considerations on the reason of bloat. *Genetic Programming and Evolvable Machines*, 3:81–91.

Branke, J. (2001). *Evolutionary Optimization in Dynamic Environments*. Kluwer.

Branke, J. (2003). Evolutionary approaches to dynamic optimization problems – introduction and recent trends. In Branke, J., editor, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 2–4.

Burke, E., Gustafson, S., Kendall, G., and Krasnogor, N. (2002). Advanced population diversity measures in genetic programming. In J. J. Merelo *et al.*, editor, *Parallel Problem Solving from Nature - PPSN VII*, volume 2439 of *LNCS*, pages 341–350. Springer.

Clerc, M. (2006). *Particle Swarm Optimization*. ISTE.

de França, F. O., Zuben, F. J. V., and de Castro, L. N. (2005). An artificial immune network for multimodal function optimization on dynamic environments. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 289–296, New York, NY, USA. ACM.

Dempsey, I. (2007). *Grammatical Evolution in Dynamic Environments*. PhD thesis, University College Dublin, Ireland.

Fernandes, C., Ramos, V., and Rosa, A. (2005). Varying the population size of artificial foraging swarms on time varying landscapes. In *International Conference on Artificial Neural Networks: Biological Inspirations*, volume 3696 of *LNCS*, pages 311–316. Springer.

Fernández, F., Tomassini, M., and Vanneschi, L. (2003a). An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–52.

Fernández, F., Tomassini, M., and Vanneschi, L. (2003b). Saving computational effort in genetic programming by means of plagues. In *Congress on Evolutionary Computation (CEC'03)*, pages 2042–2049, Canberra, Australia. IEEE Press, Piscataway, NJ.

Fernández, F., Vanneschi, L., and Tomassini, M. (2003c). The effect of plagues in genetic programming: A study of variable size populations. In Ryan, C., *et al.*, editor, *Genetic Programming, 6th European Conference, EuroGP2003*, Lecture Notes in Computer Science, pages 317–326. Springer, Berlin, Heidelberg, New York.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Huang, C.-F. and Rocha, L. M. (2005). Tracking extrema in dynamic environments using a coevolutionary agent-based model of genotype edition. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 545–552, New York, NY, USA. ACM.

Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In C. Ryan *et al.*, editor, *Genetic Programming, Proceedings of the 6th European Conference, EuroGP 2003*, volume 2610 of *LNCS*, pages 71–83, Essex. Springer, Berlin, Heidelberg, New York.

Koza, J. R. (1992). *Genetic Programming*. The MIT Press, Cambridge, Massachusetts.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk. (With contributions by J. R. Koza).

Rand, W. and Riolo, R. (2005). The problem with a self-adaptative mutation rate in some environments: a case study using the shaky ladder hyperplane-defined functions. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1493–1500, New York, NY, USA. ACM.

Tanev, I. (2007). Genetic programming incorporating biased mutation for evolution and adaptation of snakebot. *Genetic Programming and Evolvable Machines*, 8(1):39–59.

Tomassini, M., Vanneschi, L., Cuendet, J., and Fernández, F. (2004). A new technique for dynamic size populations in genetic programming. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC'04)*, pages 486–493, Portland, Oregon, USA. IEEE Press, Piscataway, NJ.

Yang, S. (2004). Constructing dynamic test environments for genetic algorithms based on problem difficulty. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1262–1269. IEEE, Piscataway NJ, USA.