Dynamic Ant: Introducing a new benchmark for Genetic Programming in Dynamic Environments

Technical Report UCD-CSI-2011-04

David Fagan, Miguel Nicolau, Erik Hemberg, Michael O’Neill, Anthony Brabazon

Natural Computing Research & Applications Group
Complex & Adaptive Systems Lab
School of Computer Science & Informatics
University College Dublin
Ireland

david.fagan@ucd.ie, miguel.nicolau@ucd.ie, erik.hemberg@ucd.ie,
m.oneill@ucd.ie, anthony.brabazon@ucd.ie

April 14, 2011

Abstract

In this paper we present a new variant of the ant problem in the dynamic problem domain. This approach presents a functional dynamism to the problem landscape, where by the behaviour of the ant is driven by its ability to explore the search space being constrained. This restriction is designed in such a way as to ensure that no generalised solution to the problem is possible, thus providing a functional change in behaviour.

1 Introduction

Explicit studies of the behaviour of Genetic Programming (GP) in dynamic environments are minimal to date [2], and this has been recognised as an open issue for the field of GP [14]. In this study we present a novel benchmark problem,
Dynamic Ant, and examine the behaviour of a grammar based form of GP on this benchmark.

The Ant Trail Problem has been viewed by many in the field of Evolutionary Computation (EC) as a complex problem [7]. The most famous and widely used example of such a trail problem is the classic Santa Fe Ant Trail problem. Harder variants also exist, such as Los Altos Trail and San Mateo Trail, which build upon the ideas behind the Santa Fe Trail, and extend them by adding the need to learn more complex behaviours. These trails are used as common benchmark problems within the GP Field [15, 5], as they provide problems which have been widely implemented, and results are available for many different GP Systems. Grammatical Evolution (GE) [13, 2] is a grammar based form of GP [8], that has established itself as one of the most widely used, and popular forms of grammar based GP over recent years [9].

It has been noted that solutions evolved with GE in a static environment, have performed well when applied to real world dynamic situations. This is of great interest, as the problems the solution came from are based in static environments. Current research interests, have led to an investigation into applying GE to dynamic problem domains, but first a definition of dynamic must be reached. To aid with this definition and before tackling complex real world problems, a strategy of establishing a suite of benchmark problems is desirable. Dempsey et al. [2] makes reference to a spectrum of dynamism, in a very in-depth review of the current state of the dynamic problem domain, and examines similar ideas by Branke [1] and DeJong [4], to name a few. The spectrum finally described is one from a problem where the change is predictable and small, to a problem which is completely random. Due to this spectrum we began to investigate the possibility of taking a standard GP benchmark, and using it as inspiration for a dynamic benchmark that could be placed on the spectrum. In [6] Langdon et al. define a dynamic ant problem which takes inspiration from the Santa Fe style trail, and constructs what they deem to be a dynamic trail, by taking core modules of the Santa Fe style trail and constructing random trails. There is one drawback to this approach, it was observed that a general solution to all trails could be found. Is this dynamic or can this be viewed more as a generalisation problem? In the opinion of the authors a problem can be said to be dynamic, if a functional change exists over time. This functional change should not allow for a generalised solution to exist for all possible functions, as the goal of a dynamic problem is to adapt and evolve towards the ideal solution for a problem at a given time. With this in mind we propose a new Dynamic Ant, which experiences a functional change over time.

The remainder of the paper is structured as follows, Section 2 introduces Dynamic Environments followed by an introduction to the Dynamic Ant problem in Section 3 and a brief overview of GE in Section 4. The experimental design is outlined in Section 5, before we present results in Section 6, followed by conclusions and future work in Sections 7 and 8 respectively.
2 Dynamic Environments

Goldberg [3] made steps into the dynamic domain in 1987, however, until recently the problem domain has remained relatively untouched as was noted in [2]. Recent works by [2, 10, 1, 4, 11], have begun to tackle what is a dynamic environment, however the definition is fraught with pitfalls. One way to view a dynamic environment, would be a road around the world. As a person drives along the road they desire the best solution for the current road conditions. As the condition of the road changes, so the demand on the type of vehicle changes. If this problem was subject to evolution one might consider the following ideas. Would it be considered more dynamic to evolve towards a jeep type vehicle, which performs well on all surfaces, and is finely tuned towards current environment needs, but remains a jeep at its core, which could be considered a generalisation problem. Should the population try to maintain some form of memory of previously good cars, so that it can seed the population, based upon what car is most suitable to the current situation. Should the problem just try and evolve the best solution to the current environment.

Is this problem even to be considered dynamic? With this in mind the idea of a functional change within the environment is the key concept to what the authors consider Dynamic. Problems within the dynamic domain should require a distinct solution at each time and no general solution should exist, such as the jeep above. How this functional change is viewed is subject to debate, but it would have to fall within the constraints of the grammar, and the possible choices it contains. One can’t expect an ant to suddenly be able to avoid poison pills, without it first having the ability to distinguish between the good and bad pieces of food.

Defining a definition of a dynamic problem is difficult enough, detecting a change, and the scale of change, and how to deal with these changes are the next problem. Many approach to this problem were examined in [2], and current approaches such as memory of previous solutions, and sentinels on the search space to detect change were examined. Recent work by [16], has set about ways to quantify rate of change, and magnitude of change, and to use these to determine how best to deal with change. It is deemed that, a large magnitude of change benefits from continuing from the current state of the program, using knowledge already discovered to help guide the search, where as a small magnitude of change reacts better to a random restart, as this will allow the system to escape possible local optima.

The idea of change is critical to a dynamic environment, but how exactly can this change be described. There are many aspects of a problem we can change that can be described as dynamic. By changing the constraints of a problem we can add dynamism, but the number of constraints within a problem can be limitless. You could change constraints that define the problem landscape, such as the size of the area available to explore, or the primitives given for a controller can be relaxed, to allow for more complex behaviours, or insertion of domain knowledge. By adding and removing terms to the grammar, it is possible to also dynamically change the fitness landscape of the problem, and
open up previously inert regions that now become reachable. All these changes can cause a functional change to a problem, but the other aspect of dynamism to be considered is the frequency and degree of change. Too large a degree of change in a problem can cause a problem to become extremely hard to solve, such as adding the ability for ants to fly, and have them navigate a 3D space shooting targets, would be a very drastic change, and may bring the problem to too complex a level. The frequency of change can also effect performance on problems, too short a period between change can lead to evolution not being able to evolve towards a solution before a change, and too long a period in a certain problem can cause convergence to a part of the solution space, which makes it very hard for evolution to adjust to change.

3 Dynamic Ant Problem

Dynamic Ant is a problem that came about after much discussion into what constituted a dynamic problem. The reasons for basing our first dynamic problem in the ant domain were twofold, firstly it was based on a classic benchmark problem in the Santa Fe Ant Trail [5], and secondly the ant trail represents a challenging type of problem to solve [7]. Originally the Ant was based upon work by [6], in which a form of dynamic ant was proposed, where the building block, or common modules, that made up the Santa Fe Trail where extracted, and then randomly put together to form a trail. However this design didn’t fit in with the idea that a dynamic problem should evolve a functional change, and that this change should lead to distinct behaviours. As the authors of [6] noted, their approach generated a solution which could successfully complete any generated trail as well as the original Santa Fe Trail.

With this in mind, we set about designing a trail which would remain static, and the energy given to the ant, or the number of moves possible, would be set so that a change in energy would drive a functional change in the ants behaviour. The trail designed is shown in Fig 5, while the trail appears simple it is so by design, to enable detailed investigation into the behaviour of the ant, as we increase the number, and difficulty, of obstacles that need to be negotiated to maximise food consumption. In order for our dynamic ant to fall in line with the idea of a functional change, specific energy levels had to be chosen in order for the ants behaviour to be manipulated. The classic behaviour of an ant is to maximise the amount of food it eats. By manipulating the energy available to the ant we could make it attractive for the ant to skip food, as it would be less efficient to eat food down a certain path. The starting position for the ant would be pointing east, while positioned at the start of the trail, in the top left corner. The Dynamic Ant uses a similar grammar to all other Santa Fe type trails as shown in Fig 1.

Looking at the Trail the initial energy level chosen was 20, this would take the ant along the top of the trail, but not as far as the turn to go down the trail. We experimented with adding more gaps, but functionally it had no effect on the ants behaviour. At energy 20, the ant must learn to skip the loop and simply
move();

or

if(food_ahead == 1){
    move();
else{
    move();
}

move straight, as is seen in the program in Fig 2. Once we had established this first energy level, we set about identifying a level which would drive the ant to go around the top right corner and stop short of the first gap. For this a value of 42 was chosen so as to stop the ant short of the gap. This energy level led to the program seen in Fig 3, this drives the ant into all corners. After this an energy level of 60 was chosen to get the ant past the first of the gaps on the right hand side. The addition of this gap led to some interesting behaviours being observed. Solutions were now learning the ability to get a maximum score on energy 60, but also on the energy 20 level as the importance of skipping gaps was learned. Next we wished to reach the next obstacle, which was added to the trail the corner with the gap. Increasing the energy level to 100 showed the most consistent achievement of this target, to get around the bottom corner. Finally an energy level was required to solve the trail, this was deemed to be 140 after much testing, and an example of a solution for 140 is shown in Fig 4. This solution uses all the available energy to consume all available food. However, this solution does not reach maximum values on any other solutions.

During the testing of the ant problem, the variety of solution found was extremely interesting, and some evidence of bloat was found in the solutions generated, but this led to an increase in performance. In Fig 2 there are two solutions given. The first move(); is the solution that is converged to after initialisation, however after one cycle through the energy levels we began to see the second example in Fig 2 become more visible, as it allows for easier adaption to other energy levels, by adding the redundant else statement.
if(food_ahead == 1){
    move();
} else{
    right();
}

Figure 3: Example solution at Energy 42

4 Grammatical Evolution

Grammatical Evolution (GE), is a grammar based form of Genetic Programming (GP). Whilst GP relies upon constructing parse trees and performing operations on the parse tree, GE takes inspiration from DNA-Protein mapping in its approach to generation of solutions. GE relies upon the use of a list of integers known as a chromosome or genotype. This chromosome is then taken and mapped to a phenotype or solution. The process of mapping is the application of a set of rules or grammar, to the genotype as described below. In GE, we begin the mapping process by finding the start symbol in the grammar. This non terminal (NT) in the case of the example grammar shown in Fig. 6, \( <e> \) is then evaluated using Eq. 1. By taking the first codon value of the GE chromosome (12), and the number of expansions possible for the state \( <e> \) (2), we get the first expansion of the tree, where \( <e> \) expands to \( <e><o><e> \) \((12\%2)\). From this point on, the leftmost NT is always expanded first in the derivation process. This action will continue to be performed until no NTs remain to be expanded in the derivation tree. An example of this mapping is shown in Fig. 7, based on the example grammar shown in Fig. 6, where the order of expansion is indicated by a set of numbers on the arrows between the blocks on the diagram, in the form of 1\((12\%2)\) where 1 is the expansion order and 12\%2 is the application of Eq. 1.

\[
\text{New Node} = \text{Codon value} \% \text{Number of rules for NT} \quad (1)
\]

This process allows great flexibility for GE, which can produce solutions in any language. To date it has been used to generate solutions to problems in languages including Lisp, Scheme, C/C++, Java, Prolog, Postscript, English and architectural structures to name a few. The big power in GE is the ability for the user to manipulate the problem, by simply tweaking the grammar such as adding domain knowledge to a problem. A process which might require a complex ADF in GP.

5 Experimental Design

We wish to test the hypothesis, that cycling through different functional requirements of the ant problem will not be detrimental to performance of GE in the dynamic ant problem, and that GE can move within the solution space,
if(food_ahead() == 1){
mmove();
}
else {
if(food_ahead() == 1) {
if(food_ahead() == 1) {
right();
}
else {
mmove();
}
}
else {
left();
if(food_ahead() == 1) {
mmove();
}
else {
left();
if(food_ahead() == 1) {
mmove();
if(food_ahead() == 1) {
mmove();
}
else {
mif(food_ahead() == 1) {
mmove();
}
else {
left();
mmove();
}
}
right();
}
}
else {
left();
if(food_ahead() == 1) {
mmove();
}
else {
left();
mmove();
}
}
}
}

Figure 4: Example solution at Energy 140
Figure 5: Dynamic Ant Trail

\[ <e> ::= <e> <o> <e> | <v> \]
\[ <o> ::= + | * \]
\[ <v> ::= 0.5 | 5 \]

Chromosome ::= 12,8,3,11,7,6,11,8,4,3,
\[ \quad 3,11,15,7,9,8,10,3,7,4 \]

Figure 6: Example Grammar and Chromosome.
Figure 7: Standard GE Genotype to Phenotype Mapping.

to optimal solutions for the current period of activity in the problem. We also wish to verify that the optimal solutions for each energy level are distinct, and that no universal solution exists for all possible states of the problem. We begin by investigating the structure of the code produced by GE, and observe the behaviours, by taking the best solutions found during the design phase of the experiment, and following their paths through the trail. We will then move on to investigate the performance of the problem in a cyclic environment, where we will measure performance, in terms of examining the average best fitness and average mean fitness over different time periods and cycles. This will be used to see if GE retains knowledge, within the population, to help the population to adjust to change.

We adopted GEVA v1.1 [12] for the experiments, which was enhanced with a temporal framework, to enable GE to have the aspect of time available to the fitness function to conduct this study. The evolutionary parameters adopted on the cyclic part of the problem are presented in Table 1. We decided upon setting a limit to the total number of evaluations within a run, and then adjusted the number of evaluations per cycle, to enable us to observe the behaviour of the problem with different frequencies of change, within the problem. By adjusting the evaluations per cycle, we vary the number of generations spent at each energy level within a cycle. For each setup we performed 50 runs, using the same range of random seeds.
Table 1: Parameter settings adopted on all problems examined.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Evaluations</td>
<td>100,000</td>
</tr>
<tr>
<td>Evals per cycle</td>
<td>500, 2500, 5000, 10,000</td>
</tr>
<tr>
<td>Generations</td>
<td>200</td>
</tr>
<tr>
<td>Population size</td>
<td>500</td>
</tr>
<tr>
<td>Replacement strategy</td>
<td>Generational with elitism (10%)</td>
</tr>
<tr>
<td>Selection</td>
<td>Tournament size=5</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>0.02 (integer mutation)</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>0.9 (variable single point)</td>
</tr>
<tr>
<td>Initial chromosome length</td>
<td>100 codons (random init)</td>
</tr>
</tbody>
</table>

Table 2: Max food consumed for defined num. of moves.

<table>
<thead>
<tr>
<th>Moves Possible</th>
<th>Food Remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>59</td>
</tr>
<tr>
<td>42</td>
<td>39</td>
</tr>
<tr>
<td>60</td>
<td>27</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>140</td>
<td>0</td>
</tr>
</tbody>
</table>

6 Results

Initial tests for this problem were done specifically to establish base lines for performance, and verify the energy levels established in the experimental design. From these runs the best programs were extracted, examples of which are seen in fig 2, 4, and run at all energy levels to verify that no general solution existed. Optimal performance levels were established, and these are noted in table 2.

Once the base line was established, we then performed the experiments as described in Section5, and the graphs are presented below. Fig 8, 9, 10, 11, show the average best fitness across 50 runs. Fitness is expressed as the number of food pieces left on the trail, once all moves have been exhausted. Fig 12, 13, 14, 15, show the average mean fitness of the population, across the 50 runs. These graphs are included to show how the population, as a whole, is performing in the problem.

7 Conclusions

From the above results, certain trends are apparent. GE’s performances continues to improve over the 200 generations, regardless of the frequency of change, or the number of cycles. By looking at all the graphs, we can see how the lowest points on the graph continue to improve, in both average best fitness and average mean fitness as we cycle. This points towards the population continuing to learn and retain knowledge.
Figure 8: Graph of Average best fitness with 5 generations per cycle over 200 generations

Figure 9: Graph of Average best fitness with 25 generations per cycle over 200 generations
Figure 10: Graph of Average best fitness with 50 generations per cycle over 200 generations

Figure 11: Graph of Average best fitness with 100 generations per cycle over 200 generations
Figure 12: Graph of Average mean fitness with 5 generations per cycle over 200 generations.

Figure 13: Graph of Average mean fitness with 25 generations per cycle over 200 generations.
Figure 14: Graph of Average mean fitness with 50 generations per cycle over 200 generations

Figure 15: Graph of Average best fitness with 100 generations per cycle over 200 generations
If we examine the high peaks in Figs 12, 13, 14, 15, which show the average mean fitness of the population, we can see how the amount the population drifts from the previous mean is reduced as we cycle. We can also note how, the more we cycle, the lesser this deflection from the optimal becomes. Even in the very high frequency runs of 5 gens per cycle, we continue to see this downward trend of the population towards better solutions.

Another verification to the continued gain of knowledge in the population, can be seen in how the slope of the graphs, continue to improve e.g become steeper as we cycle. At the beginning the slope is very erratic, but as we cycle the drops become more direct towards the established level of knowledge, thus showing that the population has retained some knowledge.

Finally it is worth noting that, no matter the cycle length, after 200 generations all setups, bar the 5 generations per cycle, have achieved an average best fitness and average mean fitness within the same region. Suggesting that once a setup allows longer than 1 generation at each energy level, that the higher frequency of change does not retard the systems ability to reach good solutions.

8 Future Work

This work has help to define what we mean by a dynamic problem, and has proven to be the perfect stepping stone for our research into the dynamic problem domain. As mentioned before the first goal of this dynamic research is to establish a taxonomy of dynamism, and fit problems into this, so that we can establish a suite of benchmark problems, before moving the research to real world complex problems. The Dynamic Ant is the first of these problems and it is hoped to take other known static benchmark problems, and investigate if dynamic versions of such problems, firstly meet our functional change criteria, and secondly where they place within the taxonomy of dynamic problems.

9 Acknowledgments

This research is based upon works supported by the Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References


the Second International Conference on Genetic Algorithms on Genetic
Erlbaum Associates Inc.

editors, ISMIS, volume 1609 of Lecture Notes in Computer Science, pages


fitness. In W. Banzhaf, R. Poli, M. Schoenauer, and T. Fogarty, editors,
Genetic Programming, volume 1391 of Lecture Notes in Computer Science,

K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Gold-
of the Third Annual Conference, pages 193–201, University of Wisconsin,

Grammar-based genetic programming: a survey. Genetic Programming

genetic programming and evolvable machines. Genetic Programming and


A. Brabazon. GEVA: Grammatical evolution in java. SIGEVOlution, 3(2),
2008.

Programming in a Arbitrary Language. Genetic programming, Kluwer Aca-

in genetic programming. Genetic Programming and Evolvable Machines,