Genetic Programming on IXM

Eric

15 December 2009

Report

Introduction

A system of subpopulation Genetic Programming (GP) is implemented over Illuminato X Machina (IXM)[1] boards. The GP system is designed in the spirit of the IXM boards and attempts to explore the intuitions[2] which served as the impetus for the board's creation. A series of experiments assess the effects of varying traditional GP parameters, of applying coevolution and of various physical layouts on the relative success of the GP. The inherently redundant and distributed nature of subpopulation GP is shown to be a natural fit for the IXM environment and directions for future work are discussed. In the spirit of Reproducible Research[3], all source code and experimental data referenced in this work is made available to the reader (see reproduction) in the hopes of encouraging further work in this area.

Illuminato X Machina – telos and construction

An IXM[1] board is a small square board containing an atom processor, memory, a button, led lights, hardware timers, a power rail and four connectors arrayed around the perimeter of the board. Although each board is in essence a small self-contained computer they are intended to be used in groups. When multiple boards are connected through they communicate and share power with their neighbors through their side connections.

One goal of the boards is to enable experimentation with software developed under a new set of assumptions. On a grid of IXM boards the central single CPU, the shared memory and the global clock of a traditional

[1] http://illuminatolabs.com/IlluminatoHome.htm
[2] http://reproducibleresearch.net/index.php/Main_Page
[3] http://livingcomputation.com/rpc/

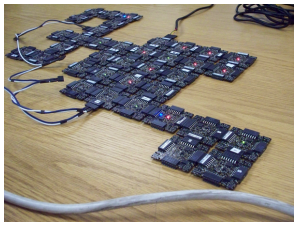

Figure 1: 37 IXM boards in operation

computer are relinquished in favor of a dynamic set of loosely coupled peers, each equipped with it's own personal clock processor and memory. In such an environment robustness of operation is achieved through the abandonment of privileged points in space and time and the duplication of functionality across a largely homogeneous populations of peers.

This work attempts to manifest these themes in a GP implementation which will run effectively in the IXM environment.

GP – implementation and defense of deviations from the norm

In an effort to better align with the IXM platform the GP system employed in this work differs from GP norms in a number of areas. These deviations will be defended as they arise. The source code for the GP implementation described below can be found in evolve-sketch.pde and coevolve-sketch.pde. Both files use the collector/ IXM library for communicating results back to a central source.

Representation and Fitness

This approach evolves individuals which are represented as variable length strings of integers and arithmetic operators in Reverse Polish Notation (RPN). The individuals are evaluated by their ability to match a "target" string also represented in RPN. The simplicity of the RPN stack machine facilitated quick development on the IXM boards.

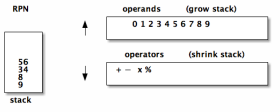

Figure 2: RPN stack machine

The only difference between the individual strings and the possible goal strings is the inclusion of the 'sine' operator in the goal strings. This unbalance encourages interesting behavior of the evolving individuals as they tackle the difficult task of approximating a trigonometric function using arithmetic operators – in effect evolving towards a Taylor Series.

Genetic Operators

New individuals are generated in the following three manners. In each of the following when an individual is needed as an input to the operation that individual is selected from the population through using tournament selection.

injection A new individual is created from whole cloth. Each element in the individual's RPN string is selected at random from the set of all valid RPN characters. This is the process used to generated an initial population immediately after board boot-up.

```
char possibilities[16] = BUILDING_BLOCKS;
ind.representation[0] = possibilities[random(15)];
for(int i=0; i < random(IND_SIZE); ++i) {
  ind.representation[i] = possibilities[random(15)];
  index = i;
}
```

mutation An modified copy is made of an existing individual. First an exact copy is made, then each step of the copies RPN string is mutated with a probability equal to $1/mutation\_prob$ where $mutation\_prob$ is a parameter of the GP system.

```
char possibilities[16] = BUILDING_BLOCKS;
for(int i=0; i<size(); ++i)
  if(random(size()) == mutation_prob)
    representation[i] = possibilities[random(15)];
```

crossover Single point crossover is used to combine to existing individuals to generate a new individual. First a crossover point is picked for each parent, then the first half from one parent is combined with the second half from the other parent as shown.

```
int mother_point = random(mother->size());
int father_point = random(father->size());
```

```
for(int i=0; i<mother_point; ++i) {
  child.representation[index] = mother->representation[i];
  ++index;
}
for(int i=father_point; i<father->size(); ++i) {
  if((index+1) >= (IND_SIZE - 1)) break;
  child.representation[index] = father->representation[i];
  ++index;
}
child.representation[index] = '\0';
```

sharing During sharing an individual is selected and is "shared" with all of the IXM board's neighbors.

Population Operations – avoiding privileged points

Up to this point the GP system we have introduced is largely standard and should be unsurprising. Where our system differs from traditional GP is in the timing and distribution of operations on the population of individuals. Since one of our goals is uniformity in both space and time we discard the notion of a fixed population cycle and instead perform all GP operations at constant frequencies. As such there are no discrete "stages" or "steps" in our GP.

Using hardware timers included on the IXM boards we scheduler the operations of mutation, injection, crossover, and sharing to recur at fixed frequencies. The frequency of these operations are parameters of the GP system. Whenever one of these operations returns a new individual (e.g. the product of crossover, or an individual shared by a neighbor board) the new individual is incorporated into the population and the current worst individual is removed form the population. The only time an individual will be removed from the population is when it is displaced in this manner.

Given the above setup all of the GP operations are constantly acting on the population in a semi-stochastic interleaved manner. No randomness is explicitly added to the operation scheduling (although this would be sympathetic with our themes) however as the boards periodically become too busy pending GP operations are can be delayed adding an element of randomness to the system.

Board Layout

The following illustrates the functional components of our GP framework as implemented out on a single board.

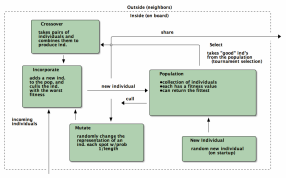

Figure 3: Layout of a single IXM board

Properties

The GP system as described has the following properties which are desirable for the IXM environment.

- all boards are peers
- any number of boards can be used effectively – including a single board
- increasing the number of boards increases the effectiveness of the GP system – following some asymptotic curve
- boards can be added to the GP system during execution and incorporated *on the fly*
- the system degrades gracefully as boards are removed from the system

Experimental Methodology

All experiments are run using the following methodology. A laptop side Ruby script (either experimenter.rb or coexperimenter.rb) communicates with an attached group of IXM boards using the libixm Ruby library and the `sfbprog` command distributed with the core IXM software. The Ruby scripts take a series of parameters and for each combination they

1. input parameters to the boards
2. initiate board execution
3. collect and save raw data output by the boards

4. timeout overrunning experiments

5. repeat

The experiments presented below had running times between 1 hour and close to 30 hours. The scripts are executed using the gnu screen program to allow persistent execution while the user is not logged into the machine.

The boards report all results using the collector/ IXM library. The collector library allows each board in a group to report parameters back to a central "collector" appending a "path" to the returned results. The path can be used to uniquely identify the board and assign it coordinates in the 2D geometry of the board layout.

All graphs generated as part of this report used the collector output as saved by the above Ruby scripts along with the group.rb and board.rb scripts. These scripts translated the raw data into 2D information in a form suitable for input to gnuplot.

Results

GP Parameters

Initial experimentation was aimed at ensuring both that our GP system was able to solve simple tasks and that both GP operations ('mutation' and 'crossover') improved GP performance.

These results were generated running evolve-sketch.pde on a single IXM board. Mutation and crossover frequencies of 10 milliseconds and 100 milliseconds (m.10, b.10, m.100, and b.100 respectively) were tested resulting in the runtimes shown below. Times shown are the average time taken to generate an ideal individual over 5 runs. The results indicate that both mutation and crossover reduce the runtime required for the GP to solve problems. In addition it appears that crossover is more effective against harder problems (e.g. "xxx**xxxx***+") while mutation is more effective against simpler problems (e.g. "7xxx**+").

file:graphs/gp-params.svg

Sub-populations and Sharing

After the basic GP operations had been verified we investigated the effect of distributing the GP over multiple boards. An series of runs were performed using sharing frequencies of 100 milliseconds and mutation and crossover frequencies of 10 milliseconds. Times shown are the average time taken to generate an ideal individual over 5 runs. Although the effect of adding a

second board was not dramatic there is clear evidence that the addition of a second board and a second population did increase the effectiveness of the GP.

file:graphs/sharing.svg

Sharing and Layout

Next the effects of different sharing rates run over a large group of 15 boards was investigated. The sharing experiments were run over two different board layouts – a straight line and an altered figure eight. The results for each layout are presented as well as a comparison between the two. In all experiments below the following three goals functions were used. Each GP parameter combination was allowed 10 minutes to attempt to fit each function. 10 runs were performed in each setup and all reported results are the average of the 10 runs.

- line

  The runtimes for each sharing rate by goal. All sharing rates are reported in milliseconds. In general the results seem to illustrate the a sharing rate of 1000 milliseconds performs best.

  - "xxx**xxxx***+"

sharing rate	ave. time to completion
10000	3.8355099
1000	3.2090558
100	3.2239733

  - "7xxx**+"

sharing rate	ave. time to completion
10000	4.9986461
1000	3.1983512
100	2.1230925

  - "xs55+55+**"

    This goal is equivalent to $100sine(x)$ which is not possible for our GP individuals to match as they do not have the sine function as

one of their operators. The average best score for each sharing rate is reported.

sharing rate	ave. time to completion
10000	invalid
1000	156.743
100	164.008

the reason that the above results for 10000 are labeled "invalid" is that is appears that some boards did not successfully have their goal reset from "7xxx**+" to "xs55+55+**" in these runs, so no data is available.

- eight

The runtimes for each sharing rate by goal. All sharing rates are reported in milliseconds. In general the results seem to illustrate the a sharing rate of 1000 milliseconds performs best.

  - "xxx**xxxx***+"

sharing rate	ave. time to completion
10000	3.6102906
1000	2.8806907
100	6.4068757

  - "7xxx**+"

sharing rate	ave. time to completion
10000	24.0118271
1000	3.1030883
100	1.9693912

  - "xs55+55+**"

    This goal is equivalent to $100sine(x)$ which is not possible for our GP individuals to match as they do not have the sine function as one of their operators. The average best score for each sharing rate is reported.

sharing rate	ave. time to completion
10000	255.311111111111
1000	183.966666666667
100	253.433333333333

- video results

  The following videos are provided to better illustrate the dynamic fitness levels across multiple boards during the previous runs. In these videos each board is represented as a bar in a 3d histogram. The placement of the bars mirrors the physical placement of the boards and the height of the bar is equal to the most fit individual on the board. Recall that fitness is calculated as the **difference** between an individual and the goal, so a lower fitness score is better.

  note: if the following don't begin playing automatically they can be saved to your desktop and played using a video player. On a mac you may need to use VLC if your default video player doesn't understand these files.

(Loading video)

Coevolution

Future Work

- meta-GP
- taking and giving boards
- splitting up the fitness space across the boards

Reproduction and Expansion of this work

[fn:4]

[fn:5]

[fn:6]