Genetic Programming, 7th European Conference, EuroGP 2004, Maarten Keijzer *et. al.*
editors, Coimbra, Portugal, 5–7 April 2003, LNCS 3003, pp349–358, Springer-Verlag.

Global Distributed Evolution of L-Systems Fractals

W. B. Langdon

Computer Science, University College, Gower Street, London, WC1E 6BT, UK
http://www.cs.ucl.ac.uk/staff/W.Langdon

Abstract. Internet based parallel genetic programming (GP) creates fractal patterns like Koch's snow flake. Pfeiffer, `http://www.cs.ucl.ac.uk/staff/W.Langdon/pfeiffer.html`, by analogy with seed/embryo development, uses Lindenmayer grammars and LOGO style turtle graphics written in Javascript and Perl. 298 novel pictures were produced. Images are placed in animated snow globes (computerised snowstorms) by www web browsers anywhere on the planet. We discuss artificial life (Alife) evolving autonomous agents and virtual creatures in higher dimensions from a free format representation in the context of neutral networks, gene duplication and the evolution of higher order genetic operators.

1 Introduction

For two years we have been running an experiment in distributed evolution in which small local populations within each user's web browser communicate via Javascript with a central server holding a variable sized global population (see Figure 1). Pfeiffer is intended as a prototype to show the feasibility of evolving agents on many small computers running across the Internet under the user's actions as a fitness measure. (A Java based model, albeit without interactive evolution, is sketched in [Chong and Langdon, 1999].) The agents are intended to be attractive and therefore they are animated in a snowstorm. Their form is given by a simple deterministic Lindenmayer grammar, whose initial seed is a Koch fractal snow flake (see Table 1).

The Biomorphs of [Dawkins, 1986] and the work of Karl Sims [Sims, 1994] in evolving virtual creatures are well known. Christian Jacob [Jacob, 2001] used the Mathematica language to evolve many classic Lindenmayer based virtual plants. The Wildwood system [Mock, 1998] allows interactive evolution of static plants, while [Hemberg *et al.*, 2001] uses it for architectural design. More recently [Ortega *et al.*, 2003] has used a linear genetic programming system (grammatical encoding [O'Neill and Ryan, 2001]) to force syntactic correctness of their Lindenmayer grammar. Note these constraints may possibly prevent massive neutral networks from forming in the fitness landscape.

The next two sections describe the GP L-system. Section 4 describes the two month trial. Section 5 describes the results. Section 6 is concerned with implementation problems. (After the trial, some of these were addressed.) The penultimate section (7) discusses where evolutionary agents might lead us. We conclude, in Section 8, that worldwide interactive evolution is feasible.

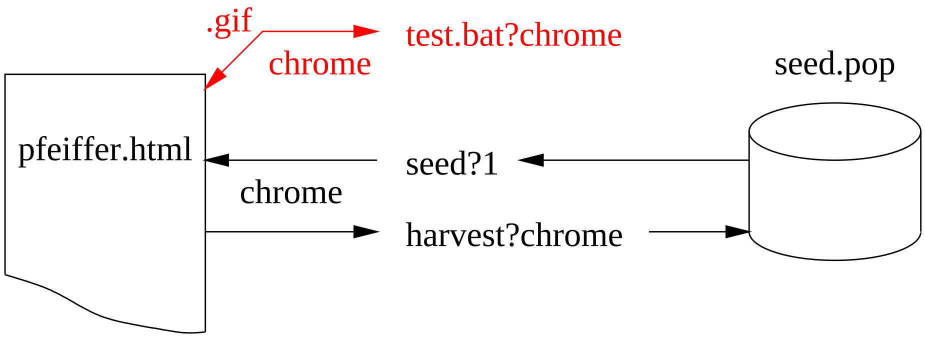


Fig. 1. Overview of Pfeiffer. The user interacts via HTML and Javascript run in their web browser (left hand side). Initial seeds (chromosomes) are either stored locally as a cookie or down loaded via the Internet via cgi program seed running on our web server. Seed returns a randomly chosen member of the global population. Evolution occurs in the user's browser. To visualise modified seeds they are passed to cgi script test.bat which interprets them as Lindenmayer grammars driving turtle graphics. The resulting graphic is convert to a .GIF file and returned to the user's browser for display. Should a user elect to save a seed, it is duplicated, stored on the user's machine as a cookie and added to the global population by cgi program harvest.

2 How Pfeiffer Works

Pfeiffer (cf. Figure 1) evolves agents and displays them moving in two dimensions across the screen of a www browser. The visual phenotype of each agent is given by a Lindenmayer system grammar. As the agents are moved across the screen they are subject to random encounters and changes which may effect their grammar. Each time the grammar is changed the agent's new shape is drawn on the screen. The user can save pretty shapes and delete ugly ones.

2.1 User Interaction

The primary goal of the user intervention is to use the user to provide selection pressure to drive the evolution of the agents. The user can save an agent by clicking on it. This causes an additional copy of the agent to be stored in the local population, overwriting a deleted agent (if any). Since the local population cannot hold more than 25 agents, if there are no deleted agents, adding an agent means overwriting an existing one. The agent selected by the user is stored in its "cookie" and appended to the global population. (Cookies allow the local population to be stored off line between sessions.) Once in the global population, the agent can be down loaded by other users and so distributed across the world. These user initiated actions exert selection pressure on the local and global populations.

The user can also use the cursor to "freeze" selected individuals. I.e. prevent them moving for ten seconds. This does not prevent them crossing over. As such it gives the user limited abilities to steer individuals towards each other and so perform mate selection.

While apparently simple, one must bear in mind that all the agents are updated and moved asynchronously. Thus the actual agent's seed can be different (for a short time) from the seed which grew into the fractal (phenotype) displayed on the screen. Time outs are used to make it clearer which user action has been initiated before starting a new one.

2.2 Central Server

The code running on the UCL server splits into two parts. Seed and harvest, which read and write seeds from and to the global population and test.bat.

Javascript has very limited graphics capabilities. It is designed to display predetermined graphics, which it down loads. Pfeiffer needs to be able to display arbitrary Lindenmayer grammars. This cannot be done in Javascript, instead it is done on the server and the results are down loaded. Test.bat interprets each new seed as a Lindenmayer grammar, generating a series of drawing instructions, which are immediately obeyed. The resulting picture is compressed and converted to .GIF format and passed back to the user's browser for display. Because of the data compression, this takes only a few seconds even on low band width connections. The delay appears to be mainly due to network latency, rather than lack of bandwidth.

3 Genetic Programming Configuration

The (up to) 25 individuals in the local population move across the user's web browser display and may run into each other. When this happens and if both individuals are mature (i.e. more than 25 seconds old) crossover occurs and the first parent's seed (chromosome) is replaced by that of its offspring. The new offspring closes up to and remains with the second parent for ten seconds. The 10s delay gives time for the offspring's .GIF to be down loaded from UCL. After 10s the child and parent randomly drift apart.

3.1 Genetic Representation

Each agent seed is a variable length linear text string. The default seed grows into the Koch snow flake (row 6 in Table 1). It is the 56 character string `v=60&str=F++F++F & it=2 & sc =5 & rules=('F','F-F++F-F')`. So it is no surprise that these characters appear often in both the local and global populations.

Spaces and control codes are removed before each seed is passed across the Internet to be interpreted. The string is split by `&` characters into parameters. They are are processed left to right. Thus if any parameter is repeated, the second "gene" is "dominant". Note with more flexible genetic operations (only a slight change to our crossover would be needed), this representation would support "gene duplication" and "translocation" [Goldberg, 1989].

Four parameters are recognised. They are `v` (angle), `str` (start string of grammar), `it` (depth of recursive expansion) and `sc` (side, in units of 0.2 pixels). `rules` is fixed. Each substring formed by splitting the seed at the `&` is further split by `=`. If the first part of the substring exactly matches one of the parameter names then its value is set to the text between the first and second (if any) `=`. If a parameter is missing, its default is used. The use of the defaults is effectively the same as if the default text where inserted at the start of every seed (albeit protected from genetic operators).

Once parameters have been decoded the Lindenmayer grammar is interpreted. First the start string `srt` Lindenmayer grammar is expanded `it` times. At each step every character which matches the left hand symbol of a rule is replaced by the corresponding right hand side. This yields a potentially very long string. To avoid tying up our server with infinite or very long recursions an arbitrary complexity limit of 2000 line segments steps was imposed.

The string is interpreted as a series of "turtle" drawing instructions:

`F` move forward `sc`/5 pixels
`+` turn clockwise by `v` degrees,
`-` turn anti-clockwise by `v` degrees,

3.2 Example of Interpreting a Grammar

Suppose the initial seed is the 59 characters `v=60 & str=F++F++F3t5F+r c sc=5 & rules = ('F','F-F++F-F')`.
After removing spaces we are left with 51 characters `v=60&str=F++F++F3t5F+rcsc=5&rules=('F','F-F++F-F')`. This splits (at `&`) into two parameters `v`=60 and `str=F++F++F3t5F+rcsc=5`. Defaults are given for missing parameter values (`it` and `sc`), while `rules` is fixed. So the grammar is `v`=60, `str=F++F++F3t5F+rcsc`, `it`=2, `sc`=5 and `rules=('F','F-F++F-F')`. Note how the original value of `sc` had been corrupted but is restored by the default. The start string is expanded twice (`it`=2) to give the final image.

Iteration		Size	Expansion	Line segments
0		3×3	`F++F++F3t5F+rcsc`	4
1		6×7	`F-F++F-F++F-F++F-F++F-F++F-F3t5F-F++F-F+rcsc`	16
2		15×17	`F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F3t5F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F+rcsc`	64

3.3 GP Genetic Operations and other Parameters

Only crossover was active during the two month period. However mutation had been used extensively in the previous two years. Crossover is "homologous" in the sense that the start of the parents' seeds are aligned before two cut points are randomly selected. However to allow seeds to change length, 50% of the time the segment cut from the middle of the second parent is either one character longer or one shorter than the segment in the first parent it replaces. If the second parent is shorter than the second cut point, the inserted fragment must be shorter than the segment it replaces. Leading to a bias towards creating shorter offspring. The first parent is replaced by its offspring.

Wildwood [Mock, 1998] only allows crossover within the L-system production rules and chooses crossover points to ensure `[]` brackets are matched. Our robust interpreter ensures operation even if crossover violates syntactic rules and allows all the parameters to evolve.

First parent
```
v =l-72&strF+'+F+4+F+F&&st F+2& 'c s 5tetulF+ =-F Fe ,- F&F(Fl+&=
```
Second parent
```
v =l=72&&strF '+F+4+F+F&&st F+2& 'c s 5tetulF+ =-F Fe,,- F&F(Fl++=
```
Offspring, replaces first parent
```
v =l-72&strF+'+F+4+F+F&&st F+2& 'c s | 5tetulF+ =|-F Fe ,- F&F(Fl+&=
```

Fig. 2. Example crossover. Length of first parent 67, first cut point at 37, remove 10 characters, insert 11 characters. 68 characters in offspring.

4 How Much Has Pfeiffer Been Used

From December 2001 to November 2003, 43,681 Lindenmayer grammars were evolved by 740 user sessions from 171 different sites. Usage of Pfeiffer has been highly non-uniform. The length of individual sessions is approximately log-normal. However activity also varies in time reflecting the system's development. E.g. usage increased after July 2003 when support for additional browsers (Mozilla, Netscape 6 and Microsoft) was implemented.

In the two months from Wednesday 10 September 2003 to Sunday 9 November, 5,653 Lindenmayer grammars were evolved, interpreted and down loaded by 87 user sessions from 69 different domains. Surprisingly "commercial" sites dominate with only eight obviously academic sites. Usage has been truly international and covers at least 14 countries (be, ca, com edu net org, dk, it, lb, lt, mx, nl, pl, sa, sg, uk, za). The mean number of .GIF down loaded per session was 65. 50% of sessions down loaded 18 or more (corresponding to loading all or nearly all of the initial population from the network). It is disappointing to note that half user sessions last 39 seconds or less. This suggests a serious problem in starting Pfeiffer. Possibly many users are not prepared to wait the 38 seconds on average needed to down load the initial population and start the HTML page.

5 What Has Pfeiffer Learnt

The global population grew roughly linearly during the two month trial. Table 1 tabulates the changes. Of the 43 new seeds, 29 can be attributed to genuine users and 14 to web spiders. 298 totally new phenotypes were evolved by local populations however only one was saved in the global population by a user.

Most users added none or only one seed to the global population but two users added five each. If we group the seeds in the global population by their appearance (phenotype) the new members have been allocated roughly in proportion to the the increase in population (145/102). This suggests no clear user preference. Perhaps this was in part due to poor user interface, leading the Javascript to misinterpret the user's intention.

The asynchronous nature of the update of the agent's phenotype after crossover leads to a lack of direct connection between the user and the system. After the trial the user interface was improved. E.g. various measures such as time outs and visual cues have been implemented to address the problems of the user "clicking but nothing happens" and to overcome the inherent delay in updating an agent's appearance after crossover.

While there were fluctuations in program size, lengths did not change dramatically. This lack of bloat is also consistent with noisy or weak selection.

During the two months, 5653 seeds were interpretted and down loaded. Apart from a few special cases, these contained the same characters as the initial global population, in the same proportions (within 2%). Most (3279 58%) of the .GIF images were copies of the default Koch snow flake. 1531 (27%) images where identical to another image held in the global population. 843 (15%) new images were down loaded. As usual there was a degree of overlaping phenotypes, with these 843 down loads consisting of 298 totally new .GIF files. The usage of these novel shapes was very varied. Two novel .GIF files account for 307 (36%) of down loads but others were evolved just once.

Three .GIF are evolved much more often than their frequency in the initial global population. These are the given in rows 6 (default, Koch, 10×12), 3 (invisible, 49×1) and 4 (10×3) of Table 1). While this may be a response to user preference, another explanation is that crossover is finding many different Lindenmayer grammars with these phenotypes. E.g. by damaging the parameters, so either the defaults are used or `v` is set to a small value.

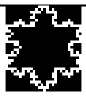
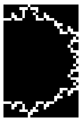
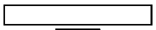
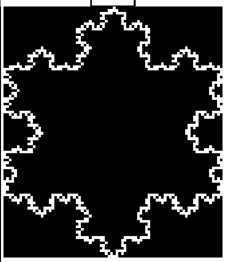
14 junk seeds were inserted into the global population by software robots or web spiders. Spider unknown.thorn.net was responsible for inserting rogue characters <PLAINTEXT> into the global population. Similarly % managed to slip in. Note while unwanted, Pfeiffer continues to function satisfactorily despite these.

6 Practicalities

The server software needs greater protection against web spiders and software agents. (This was added after the trial.)

Table 1. Global population at start and end of the two month trial.

Where a grammar does not specify a value, or the value is illegal the default (Koch) is used. There are considerable variations in start symbol `str` but in most cases the specified value is illegal and so replaced by the default. Size is the number of characters in the seed. (Where different grammars yield the same .GIF, minimum-median-maximum values are given.) `v` is the angle through which the turtle is turned (left or right, - or +). `sc` is the number of 0.2 pixels to move forward (on F). Steps is the number of turtle moves to interpret the grammar (at depth `it`).

Used	Image			Number			Grammar					
		size	bytes	Sep	Nov	inc	Size	`v`	`str`	`it`	`sc`	Steps
38		28×32	140	1	2	1	58	60		3	5	192
110		28×42	140	2	5	3	66-119-119	60	1.†	2	10	64
877		49×1	47	25	29	4	56-68-69	0	†	2	5	48
249		10×3	49	2	2	0	56	5–6		2	4	48
24		82×94	491	1	1	0	55	60		4	5	768
3279		10×12	64	59	91	32	0-64-69	60	†	2	5	48
170		15×14	73	6	7	1	57	60		2	5	48
31		14×14	76	2	3	1	67	-72	2.	2	5	80
11		45×16	90	1	1	0	93	9		2	5	48
21		24×17	98	0	1	1	56	60	3.	2	5	96

Default start string `str` is F++F++F.
1. F+F+F+F
2. +F+F+F+F+F
3. F++F++F3t5F+rc34+c++'l44FrF'15

†A few start symbols differ, sometimes radically, yet yeild the same image.

During the trial evolution was allowed to exploit a hole where by it generated 997 (2%) invisible agents. Crossover created these by ensuring v=0. The hole was fixed after the trial.

The evolution of large and complex fractal patterns suggest that Pfeiffer is already pressing up against the 2000 steps limit. After the trial, this limit was relaxed significantly without undue hardship.

We must admit to having been caught out by the significant start up delay. This was not a feature of the original Netscape 4 implementation but arose as a "finishing touch" during porting to other browsers. One of the original motivations for using Javascript as opposed to using Java was that Java applets had come with a painful start up overhead. This delay was removed after the trial.

6.1 Real time performance

On average each .GIF takes 300mS to generate and occupies 66 bytes. On average each seed takes 60 bytes. On a fast link (2Mb/S) the time to process each each new seed should be dominated by drawing the .GIF. Even on a 9k6 baud line, the average data transfer time should be only 140mS. Thus, ignoring processing time in the user's web browser and network latency, Pfeiffer should be able to process between 2 and 3 fractal images per second. In contrast typical maximum rates are 0.5-1.0 per second (mean 0.73). This suggests performance is dominated by network (including server) latency and browser overhead. To some extent these are outside our control. Together these suggest there is little performance gain to be had within the existing system design.

Pfeiffer gathers a lot of data but some additional data might be helpful. For example the type of computer/browser being used. Also mostly the data gathered is about successful operation, little is said about things that failed. On a less technical front, there is no attempt to gather user feed back. E.g. what did they think about it? Was it difficult to use? Why did they stop? Did they find it boring?

Surprisingly the computational power of a standard (350MHz) PC is only just sufficient. Smoothly animating the movement of a population of 25 .GIF images across a browsers screen can fully load it. One reason for moving to a Java implementation would be to display the L-system directly thus removing the network delays associated with crossover. However one could expect similar CPU loading issues with Java animations as with Javascript animations. This would need careful investigation.

6.2 Non-portability

It was somewhat disappointing to discover that so recent a language as Javascript is seriously non-portable. Not between different computers. (Almost all development was on computers of the same type, running the same or similar operating systems) but between different browsers. So code running in one browser would behave radically differently when run on the same machine in a different browser.

Portability was less of an issue on the server. Debugging and development tools are problematic in both server and browser. Development of Javascript was much the more painful.

7 Future: Breeding "Intelligent" agents

Our agents are very limited. We feel they need to be able to evolve to react to their environment. They need to be able to evolve to predict their environment. Of course this makes requirements of both the agent and the environment. Also, perhaps crucially, it needs to be able to effect the environment, and predict what those effects will do for it (and for others). While L-systems have been mainly used (as we have done here) to create static structures, they can describe networks. Those networks could contain sensory and active elements, they could contain processing elements (as in artificial neural networks [Gruau, 1994; Hornby and Pollack, 2002], or even GP like communicating computational processes).

There is a strand of thought in which intelligence came from a co-evolutionary struggle between members of the same species [Ridley, 1993]. If true, can intelligence arise in isolated agents? Or are interacting/communicating agents needed?

A problem with simulated worlds has been hosting sufficient complexity so as to be challenging but still allowing agents to be able make predictions about what will happen next and what will happen to me or to others if I do this. The Internet hosts tides of data. This data is not random. It aught to be possible to harness it to give a suitable virtual environment.

We have fallen for the usual trap of constructing a two dimensional world (on the computer screen). However is there any hope of evolving artificial life (and thereby artificial intelligence) in two dimensions? Obviously three dimensions are sufficient but computer simulations offer many dimensions ($N \gg 3$).

8 Conclusions

Lindenmayer grammars can be used as the basis for a linear genetic programming (GP) system and evolve interesting fractal like patterns. Many new patterns have been evolved, some exploiting the L-system to produce some regularities and re-use of motifs. It is feasible to represent individual agent's genetic material (seed/chromosome) with a variable length text string without defined fixed semantic fields and using crossover at the character level. The representation allows a huge degree of redundancy to evolve. The "fitness landscape" clearly contains a huge degree of "neutrality" [Smith *et al.*, 2002] and evolution is using it. Yet, we are still only using a small part of a complete Lindenmayer grammar (e.g. the seeds do not use branching primitives, `[` or `]`). This loose representation allows the location etc. (as well as the meaning) of the L-system to evolve. Gene duplication, translocation and other genetic operations could be supported by such a general representation.

One of the original hopes was that "fitness" would arise intrinsically from the simulation. As well as user actions, one could imagine selection being based on aspects of the fractal patterns, their production or transfer across the Internet and their interaction with other agents. However, perhaps due to poor user interface exacerbated by the asynchronous nature of the image update system, evolution appears to have been undirected.

In terms of harvesting spare CPU cycles, the project confirms it can be done using Javascript and user's web browser but this experiment did not show it to be worthwhile. The project does hint at some successes. World wide distributed evolution is clearly feasible. (A single server prooved suficient but this must represent a bottleneck for much bigger experiments.) Perhaps more importantly one can recruit users (which are much more valuable than their CPUs) to assist in guided evolution. Finally animated tools are an attractive way to spread evolutionary computation.

Acknowledgements

I would like to thank Birger Nielsen for use of his perl Lindemayer interpretter `http://www.246.dk/lsystems.html`, Maarten Keijzer and the much put upon reviewers.

References

Chong and Langdon, 1999. Fuey Sian Chong and W. B. Langdon. Java based distributed genetic programming on the internet. In W. Banzhaf, *et al.*, editors, *GECCO*, p1229. Morgan Kaufmann. Full text in Birmingham UK technical report CSRP-99-7.

Dawkins, 1986. Richard Dawkins. *The blind Watchmaker*. Harlow : Longman Scientific and Technical, 1986.

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

Gruau, 1994. F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, 1994.

Hemberg *et al.*, 2001. Martin Hemberg, Una-May O'Reilly, and Peter Nordin. GENR8 - A design tool for surface generation. In Erik D. Goodman, editor, *GECCO Late Breaking Papers*, pp160–167, San Francisco, 9-11 July 2001.

Hornby and Pollack, 2002. Gregory S. Hornby and Jordan B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3):223–246, 2002.

Jacob, 2001. Christian Jacob. *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann, 2001.

Mock, 1998. Kenrick J. Mock. Wildwood: The evolution of L-system plants for virtual environments. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pp476–480, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.

O'Neill and Ryan, 2001. Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

Ortega *et al.*, 2003. Alfonso Ortega, Abdel latif Abu Dalhoum, and Manuel Alfonseca. Grammatical evolution to design fractal curves with a given dimension. *IBM Journal Research and Development*, 47(4):483–493, July 2003.

Ridley, 1993. Matt Ridley. *The Red Queen, Sex and the Evolution of Human Nature*. Penquin, 1993.

Sims, 1994. Karl Sims. Evolving 3D morphology and behaviour by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV Proceedings*, pp28–39. MIT Press.

Smith *et al.*, 2002. Tom Smith, Phil Husbands, Paul Layzell, and Michael O'Shea. Fitness landscapes and evolvability. *Evolutionary Computation*, 10(1):1–34, 2002.