Artificial Life Evolution in a Simplified APL2 Environment

Manuel Alfonseca
Universidad Autónoma de Madrid, Dept. Ingeniería Informática
Manuel.Alfonseca@ii.uam.es
Main topic: Artificial Life

Acknowledgment:

This paper has been sponsored by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project numbers TEL1999-0181 and TEL97-0306.

Abstract

Artificial life based on simulation evolution is a flourishing field. One of its most publicized achievements of the nineties was TIERRA, by T. S. Ray. APL2 has been used successfully to implement a similar simpler artificial life application that shows most of the interesting features of TIERRA in an efficient way, providing a modifiable environment where it is easy to experiment.

Introduction

The simulation on a computer of artificial life subject to Darwinian evolution has been an important matter of research for a long time [4]. During the 1970s, this line of thought gave rise to the genetic algorithms [1, 3, 5, 8], which became an important area on its own right, which has given rise to interesting commercial applications.

In 1991, Thomas S. Ray made news in the general press with his TIERRA artificial life system. Essentially, what he did [6] was design and emulate a special virtual machine with 32 instructions, most of which are typical machine instructions (incrementing a register, adding or subtracting two registers, shifting bits, and so forth) with a few interesting changes and additions:

Jump and call addressing by template: these instructions are followed by a template of no-op instructions (of which there are two kinds). Control is passed to the nearest section of the program preceded by the complementary template.

- A memory allocation instruction, which returns space where the artificial life individuals (cells) may copy themselves. This instruction can be considered as a call to the operating system.
- A divide instruction, that creates an independent daughter cell using the space where the mother cell has written a copy of itself.

Every instruction is a five-bit combination. Numeric constants and instruction operands are not allowed in memory (but they may exist in the registers associated to each individual). Every possible 5-bit combination has a meaning.

Each individual acts as a parallel machine which tries to copy itself in another memory block. The operating system slices time between them. Mutations (bit flips) are performed with some frequency so that there is a certain probability that the daughter cell will not be exactly equal to the mother cell, thus evolution may act. There is also a "death" routine that uses a stack of organisms to determine which of them should be destroyed at a given time.

After being subject to the execution of billions of instructions, the TIERRA environment generated many different creatures: shorter equivalents of the initial ancestor, parasites, super-parasites, cheaters, symbionts, and others.

In later years, Ray has started working on extensions of TIERRA based on the evolution of multi-threaded (multi-cellular) organisms coexisting in

different machines [7]. The "reproductive cells" are more or less equivalent to the old ancestors, while the basic instruction set has been extended to 64 instructions to allow for the "sensorial" capabilities of a new cell type.

A Simplified Artificial Life Environment

The system presented here allows for the evolution of one-cell virtual organisms in a way similar but simpler than the TIERRA environment. Each organism is defined by a very small set of 13 "instructions", has an associated value (which depends on its performance) and tries to reproduce itself on some available fixed-size space (for 10 instructions.) The set of instructions is:

```
  0: Stop
  1: No operation
  2: Load index register 1 (ir1) with the
starting address of the organism in
control
  3: Allocate space for a child organism.
Address is returned at index register 2
(ir2)
  4: Load next instruction into the
counter register (cr)
  5: Load index register 1 (ir1) with the
starting address of the organism being
executed
  6: Jump to the start of the previous
organism in memory
  7: Jump to the start of the next
organism in memory
  8: Increment the value of the organism
in control
  9: Decrement the value of the organism
in control
 10: Load the contents of the memory
address pointed by ir1 into the
accumulator (ac) and increment ir1
 11: Save the accumulator at the memory
address pointed by ir2 and increment ir2
 12: Decrement the counter register. If
it is not zero, jump back as many
instructions as indicated by the next
instruction. Otherwise, skip the next
instruction
```

The ancestor organism is defined as the following program:

```
4 cr = 10
 10
  2 load initial address of organism
    in control into ir1
  3 set ir2 = space for a child
    organism
 10 ac = [ir1]. ir1++
 11 [ir2] = ac. ir2++
 12 cr--. If 0, jump back 2
    instructions; otherwise skip 1
  2
  0 stop
  0 stop
```

There are two unnecessary instructions in the code of the ancestor organism: the initial address of the organism in control is pre-loaded by the operating system into ir1 before an organism is given control. The second stop instruction is unreachable. This means that the ancestor organism could have been programmed in just 8 instructions and is capable of copying itself into the space allocated by instruction 3. These two instructions have been put there on purpose, to see how they are used by evolution.

APL2 implementation

The implementation consists of a single function with 51 lines:

```
[0]  NT ALIFE N;A;⎕IO;S0;S1;S2;SZ;SZ
0;NA;Q;COP
[1]    ⎕IO←0
[2]    →((0≠⎕NC 'T')∧0=⎕NC 'NT')/L
[3]    ⍎(0=⎕NC 'NT')/'NT←1000'
[4]    I←TOTAL←G←0
[5]    T←NT⍴4 10 2 3 10 11 12 2 0 0
[6]    AB←10×⍳⌊NT÷10
[7]    VB←(⍴AB)?⍴AB
[8]  L:→(I≠0)/L00
[9]    VB←VB-⌊/VB ◊ (20⍴'-') ◊ OUT 'G
ENERATION' G ◊ CLASSIFY
[10]   →(N=0)/L00
[11]   G←G+1
[12]   A←(⍴AB)?⍴AB
[13]   T←,(T←(((⍴T)÷10),10)⍴T)[A;]
[14]   VB←VB[A]
[15] L00:→(0>N←N+NA←¯1)/0
[16]   TOTAL←TOTAL+1
[17]   A←AB[I]
[18]   S1←AB[I]
[19]   S2←AB[VB⍳⌊/VB]
[20]   S0←SZ0←SZ←COP←0
[21]   Q←50×S0←1
[22] L1:→(0=Q←Q-1)/I00
[23]   →(T[A]>12)/I1
[24]   →(I0,I1,I2,I3,I4,I5,I6,I7,I8,I
9,I10,I11,I12)[T[A]]
[25] I00:VB[I]←0
[26] I0:⍎((NA=¯1)∨COP=0)/'→I0B,VB[I]
←0'
[27]   COP←COP⌊10 ◊
       ⍎(SZ0≠0)/'VB[AB⍳NA]←⌊.1×(COP×V
B[I])+(10-COP)×VB[AB⍳NA]'
[28]   VB[I]←VB[I]+COP-+/T[AB[I]+⍳COP
]≠T[NA+⍳COP]
[29] I0B:I←I+1
```

```
[30]   ⍎(I=⍴AB)/'I←0'
[31]   →L
[32]  I1:→I4B,A←A+1
[33]  I2:→I1,S1←AB[I]
[34]  I3:S2←AB[VB⍳⌊/VB]
[35]   T[S2+9]←0
[36]   →I1,NA←S2
[37]  I4:SZ←SZ0←T[A+1]
[38]  I4A:A←A+2
[39]  I4B:⍎(A≥⍴T)/'A←A-⍴T' ◊ →L1
[40]  I5:→I1,S1←10×⌊A+10
[41]  I6:→L1,A←(¯1⌽AB)[I]
[42]  I7:→L1,A←(1⌽AB)[I]
[43]  I8:→I1,VB[I]←VB[I]+1
[44]  I9:→I1,VB[I]←VB[I]-1
[45]  I10:S0←T[S1] ◊ S1←S1+1 ◊ ⍎(S1=⍴
T)/'S1←0' ◊ →I1
[46]  I11:→(NA<0)/I1
[47]   COP←COP+1 ◊ T[S2]←⍎(5>?1000)⊃'
S0' '?13' ◊ S2←S2+1 ◊
       ⍎(S2=⍴T)/'S2←0' ◊ →I1
[48]  I12:→(SZ=0)/I4A
[49]   SZ←SZ-1
[50]   →(SZ=0)/I4A
[51]   →L1,A←0⌈A-T[A+1]
```

Listing 1: An APL program implementing an environment for artificial life

The left argument of this function is the size of the circular memory space where the simulated organisms will evolve. If not given, 1000 is taken as the default, or the previous emulation will continue until N organisms are given control.

Lines 3-7 generate the environment, which essentially is made of three APL variables: T (the memory), which is initially filled with copies of the ancestor organism (100 copies in the default case); AB (the address set), a vector of the initial addresses of all the organisms; and VB (the value set), a vector of the values associated to all the organisms. These variables, as well as three scalar counters (I, G, TOTAL), are global, making it possible to continue the emulation in several successive executions of the ALIFE function.

Lines 8-51 make up the emulation system. In applications of this type, where the result of a step depends on the results of all the previous steps, loops are unavoidable. The emulator is made of two embedded loops: the inner one (lines 22-51) gives control to an organism, which executes its instructions and tries to generate a copy of itself; it will maintain control until instruction "stop" is executed, or until the operating system recovers control after a certain number of executed instructions (arbitrarily set to 50) to prevent endless execution. Labels I0-I12 execute the thirteen different virtual machine instructions, many of which can be implemented by means of a single APL instruction. Variables S0, S1, S2, SZ represent the internal registers of the virtual machine: the accumulator (ac), index registers ir1 and ir2, and the counter (cr), respectively.

Instruction 3 (allocate space for a daughter cell) selects the space associated to the organism with the minimum current value, in that way eliminating the less adapted organisms. When a new organism is generated, it is assigned an initial value which is a function of the value of its mother, the value of the organism previously occupying the space, and the number of instructions copied by the mother. The mother's value is also changed depending on the number of instructions it was able to copy to the daughter organism. In this way, the ability to copy itself is positively selected.

Instruction 11, which copies an instruction from the mother to the daughter organism, is subject to "mutations", in the sense that, with a certain probability, the instruction being copied is replaced by a random instruction. The mutation rate has been set to the low value of five instructions changed out of 1000 instructions copied.

Lines 8-21 make up the outer loop, the operating system of the artificial life environment, which counts organisms and generations and produces output snaps of the situation by means of auxiliary functions OUT (which writes on the screen or to a file) and CLASSIFY, which generates statistics of the currently existing organisms, such as the following:

```
GENERATION 5
2 10 2 3 10 11 12 2 8 0 :  1
1  1 1 1  1  1  1 1 1 1 :  3
4 10 2 3 10 11 12 2 0 0 : 13
4 10 2 3 10 11 12 2 8 0 : 83
 MAXIMUM VALUE:  16
 BELONGS TO  1 COPY OF
4 10 2 3 10 11 12 2 8 0
```

After each generation, the positions of the organisms in memory are shuffled to prevent localized effects, in this way making them mobile.

Results

The results presented here have been obtained by executing the artificial life environment for 500 generations (which corresponds to 50000 organism reproductions, or about 2 million virtual machine instructions). The total time taken by the execution (including file output) was about ten minutes in a Pentium-2 at 400 MHz.

Different emulations are obtained by changing the random seed, which affects when the mutations

happen, the instruction inserted at a mutation, and the shuffling of the organisms after each generation.

It may be seen that, after just 5 generations, the ancestor organism has been replaced by a mutant that has substituted the first "stop" instruction by an "increase-my-value" instruction, which, evidently, provides it with a clear advantage. There are also three organisms which can be considered parasitic, as they have replaced there whole instruction set by no operations, in this way passing the control (and the copying effort) to the next organism in memory. Remember that instruction 2 loads ir1 with the address of the organism in control, which means that the organism under control of a parasite will copy the parasite instructions into the allocated daughter space. However, when the parasitized organism next receives control, it will copy itself with the same instructions previously used to copy the parasite. This means that parasites are actually comensals, as the other organism is not damaged by the use the parasite makes of it.

Looking at the result after 15 generations, it may be seen that dominance has passed again to a new mutant that has replaced the two unnecessary instructions by two "increase-my-value" instructions, a predictable development. The no-operation parasite has also increased its population to 17 individuals, two of which happen to have reached the maximum value (remember that parasites also use the "increase-my-value" instructions of the organisms they control).

```
 GENERATION 15
4  1 8  3 10 11 12 2 8 0 :  1
4 10 1  3 10 11 12 2 8 0 :  1
4 10 8 12 10 11 12 2 8 0 :  2
1  1 1  1  1  1  1 1 1 1 : 17
4 10 8  3 10 11 12 2 8 0 : 79
 MAXIMUM VALUE:  24
 BELONGS TO  2
1 1 1 1 1 1 1 1 1 1
```

After 25 generations, the no-operation parasite has disappeared, but a new type of parasite has taken its place: one whose first instruction is a jump to the preceding organism, which takes much less execution time to attain the same results.

```
GENERATION 25
 4  0  8  3 10 11 12 2 8 0 :  1
 4 10  8  3 10 11 12 1 8 0 :  1
 4 10 10  3 10 11 12 2 8 0 :  1
 1 10  8  3 10 11 12 2 8 0 :  1
10 10  3 10 11 12  2 8 0 1 :  1
 6 10  8  3 10 11 12 2 8 0 :  2
 4 10  8  3 10 11 12 2 8 0 : 93
 MAXIMUM VALUE:  20
 BELONGS TO  1
4 10 8 3 10 11 12 2 8 0
```

After 35 generations, a third parasite has replaced the two previous ones. In this case, the jump to the preceding organism has been delayed until after an "increase-my-value" instruction has been executed by the parasite which, added to the two normally added by the controlled organism, provide the parasite with an additional selective advantage.

```
GENERATION 35
4  5 8 6 10 11 12 2 8 0 :  4
4 10 8 6 10 11 12 2 8 0 :  5
4 10 8 3 10 11 12 2 8 0 : 91
 MAXIMUM VALUE:  16
 BELONGS TO  2
4 10 8 3 10 11 12 2 8 0
```

After 45 generations, this type of parasite has increased its number to 34 individuals, 27 of which have reduced their instructions to two: "increase-my-value" and "jump-to-the-previous-organism". This is, however, about the maximum number of parasites the environment may support. If parasites became dominant, they would not be able to find the normal organism they need to control, and their numbers would decrease immediately. This gives rise to Volterra-like[9,2] fluctuations between the populations of the parasites and the dominant normal organisms.

```
GENERATION 45
2  6 10 11  7  2  8 0 10 0 :  1
4 10  8  3 10 11  8 2  8 0 :  1
8  6 10 11 11  2  8 0 10 0 :  4
4 10  8  6 10 11 12 2  8 0 :  7
8  6 10 11  7  2  8 0 10 0 : 23
4 10  8  3 10 11 12 2  8 0 : 64
 MAXIMUM VALUE:  41
 BELONGS TO  2
8 6 10 11 7 2 8 0 10 0
```

Thus, after 65 generations, the number of parasites has gone down to three, two of which belong to a new class, similar to the first "no-operation" one. In this case, instruction 4 is executed 5 times to load a value of 4 in the counter register. Actually, these instructions are another form of no-operation, as the counter register will normally be reloaded with the appropriate value of 10 by the first instruction of the controlled organism.

```
 GENERATION 65
6 10 11 7  2 10  0 4 0 4 :  1
4 10  8 3 10 11 12 1 8 0 :  1
4  4  4 4  4  4  4 4 4 4 :  2
4 10  8 3 10 11 12 2 8 0 : 96
```

After 95 generations, we find a new effect: a new kind of organism that start like a normal mother cell, but in their first back jump in the copy loop, jump into

the same position of the preceding organism, in this way sharing the copy work between two different organisms. This may be considered as still another form of comensalism.

```
 GENERATION 95
4 10 8 3  6 11 12  2 8 0 :  1
4 10 8 3  3 11 12  2 8 0 :  1
1 10 8 3 10 11 12  2 8 0 :  1
4 10 8 3 10 11 12 12 8 0 :  6
4 10 8 3 10 11 12  2 8 0 : 91
 MAXIMUM VALUE:  18
 BELONGS TO  2
4 10 8 3 10 11 12 2 8 0
```

After 125 generations we find another new development: an organism that copies only its first eight instructions, leaving the remaining two of the organism that previously occupied the space allocated to the daughter cell. In this way, a form of genetic recombination (sexual reproduction) is achieved.

```
 GENERATION 125
4  8 8 3 10 11 12 2 8 0 :  1
4 10 8 3 10 11 12 1 8 0 :  1
4  4 4 4  4  4  4 4 4 4 :  3
4 10 8 3 10 11 12 2 8 0 : 95
 MAXIMUM VALUE:  20
 BELONGS TO  1
4 10 8 3 10 11 12 2 8 0
```

After 150 generations, the situation has changed: the species dominant since generation 15 has achieved total dominance, although new mutations will soon break the equilibrium again.

```
 GENERATION 150
4 10 8 3 10 11 12 2 8 0 : 100
 MAXIMUM VALUE:  20
 BELONGS TO  2
4 10 8 3 10 11 12 2 8 0
```

At generation 179, the parasites have appeared again in their most advantageous form, making a new maximum of the Volterra equations. This situation is maintained for a long time.

```
 GENERATION 179
8  6 10 11 11  2  8 0 4 4 : 25
4 10  8  3 10 11 12 2 8 0 : 75
 MAXIMUM VALUE:  34
 BELONGS TO  1
8 6 10 11 11 2 8 0 4 4
```

At generation 287, a new interesting species appears to break the equilibrium, which acts as a parasite to the parasites. This is achieved by replacing one of the two "increase-my-value" instructions by instruction 5 ("load-ir1-with-my-own-address".) In this way, if a parasite tries to use this organism to generate its own copy, the controlled organism copies itself instead, and will copy itself again when it receives control directly. This super-parasite thus copies itself twice in every generation, in this way achieving a large advantage that more than compensates the slight loss due to the elimination of one of the "increase-my-value" instructions. At generation 288, the super-parasite has reached a population of 22 individuals, larger than that of the parasites. At generation 289, it becomes the dominant species.

```
 GENERATION 288
4  0 8 3  6 11 12 2 2 0 :  1
4 10 8 3 11 11 12 2 2 0 :  1
1  1 1 1  1  1  1 1 1 1 :  1
4 10 8 3  6 11  3 2 2 2 :  2
4 10 8 3  6 11  2 2 2 0 :  4
4 10 8 3  6 11 12 2 2 0 : 17
4 10 5 3 10 11 12 2 8 0 : 22
4 10 8 3 10 11 12 2 8 0 : 52
 MAXIMUM VALUE:  24
 BELONGS TO  2
4 10 8 3  6 11 12 2 2 0
4 10 5 3 10 11 12 2 8 0
 GENERATION 289
1  1 1 1  1  1  1 1 1 1 :  1
4 10 8 3  6 11  2 2 2 0 :  3
4 10 8 3  6 11 12 2 2 0 :  8
4 10 8 3 10 11 12 2 8 0 : 34
4 10 5 3 10 11 12 2 8 0 : 54
 MAXIMUM VALUE:  18
 BELONGS TO  1
4 10 5 3 10 11 12 2 8 0
```

At generation 291, the parasites are practically extinct. However, when this happens, the super-parasite automatically loses its advantage (it cannot reproduce twice unless it finds an adjacent parasite), and the previous dominant species becomes again advantageous (with its two "increase-my-value" instructions.) This effect is visible in the next few generations. At 294, the super-parasite has lost dominance, and in the next generation it disappears. A few generations later, the parasites reappear.

```
 GENERATION 291
4 10 9 3 10 11 12 2 8 0 :  1
4 10 8 3  6 11 12 2 2 0 :  5
4 10 8 3 10 11 12 2 8 0 : 13
4 10 5 3 10 11 12 2 8 0 : 81
 MAXIMUM VALUE:  14
 BELONGS TO  1
4 10 5 3 10 11 12 2 8 0
. . .
 GENERATION 294
4 12 8 3 10 11 12 2 8 0 :  2
4 10 5 3 10 11 12 2 8 0 : 15
4 10 8 3 10 11 12 2 8 0 : 83
 MAXIMUM VALUE:  21
 BELONGS TO  1
4 10 8 3 10 11 12 2 8 0
```

```
 GENERATION 295
4 10 8 3 10 11 12 2 8 12 :  1
4 10 8 3 10 11 12 8 8  0 :  1
4 10 8 3 10 11 12 2 8  0 : 98
 MAXIMUM VALUE:  17
 BELONGS TO  1
4 10 8 3 10 11 12 2 8 0
. . .
 GENERATION 298
4 10 8 3 10 11 12 5 8 0 :  1
4 10 3 3 10 11 12 2 8 0 :  4
1  1 1 1  1  1  1 1 1 1 : 13
4 10 8 3 10 11 12 2 8 0 : 82
 MAXIMUM VALUE:  18
 BELONGS TO  1
4 10 8 3 10 11 12 2 8 0
```

The last interesting effect takes place at generation 340, when a fifth dominant species appears, which increases the number of bytes copied to 12. In this way, after copying itself on the space allocated to its daughter cell, it copies two extra instructions (with the mutation) into the contiguous space, thus infecting the next organism with its own mutation. In this way, it reproduces very quickly, at generation 346 becomes dominant, at 350 the old dominant species is extinct, while at 491 the new species achieves total dominance.

```
 GENERATION 340
4 10 8 3  7 11 12 2 8 0 :  1
4 10 8 3  6 11 10 2 8 0 :  4
4 10 8 3  7 11 12 2 0 0 :  4
1  1 1 1  1  1  1 1 1 1 :  5
4 12 8 3 10 11 12 2 8 0 : 11
4 10 8 3  6 11 12 2 8 0 : 16
4 10 8 3 10 11 12 2 8 0 : 59
 MAXIMUM VALUE:  22
 BELONGS TO  1
4 10 8 3 7 11 12 2 0 0
```

A new Volterra fluctuation is established between the new dominant and the different parasitic species, with sporadic recurrences of the super-parasite.

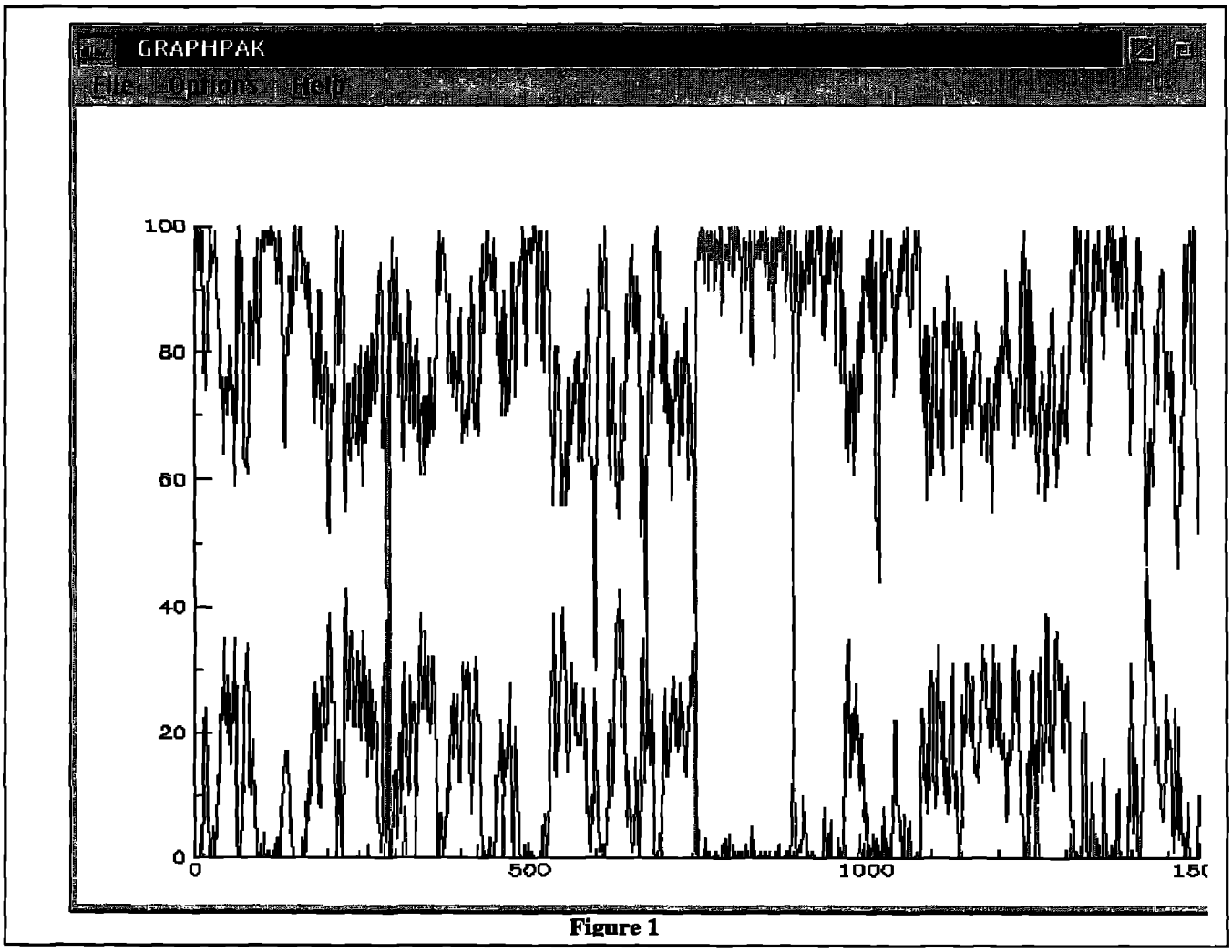

Figure 1

```
 GENERATION 500
4 12 5 3 10 11 12 2 8 0 :  1 AV
4  0 8 3 10 11 12 2 8 0 :  2
4 12 8 3 10 11 12 2 8 0 : 97 D4
 MAXIMUM VALUE:  20
 BELONGS TO  1
4 12 8 3 10 11 12 2 8 0
```

Figure 1 shows the fluctuations of the dominant versus the parasitic and super-parasitic species. The black upper curve shows the sum of all the dominant normal species; the black lower curve is the sum of all the parasites; the gray curve shows the single super-parasitic species.

Conclusion

APL2 has proved to be an appropriate language to implement artificial life applications. The one described here, based on TIERRA (by T.S.Ray), displays in a simpler environment most of the features of the older application, including the replacing of dominant species and the emergence of different parasites and super-parasites. Our APL2 function is simple enough to make adjustments and experiments easy and affordable.

References

[1] Alfonseca, M. Genetic algorithms. APL Quote Quad (ACM SIGAPL), Vol. 21:4, 1-6, Jul. 1991.

[2] M.Alfonseca, de Lara, J., and Pulido, E. Educational simulation of complex ecosystems in the World-Wide Web. Proc. ESS'98, SCS Int., .248-252, 1998.

[3] Booker, L.B., Goldberg, D.E., and Holland, J.H. Classifier Systems and Genetic Algorithms. The Univ. of Michigan Cognitive Science and Machine Intelligence Laboratory Technical Report 8, 1987.

[4] Fogel, L.J., Owens, A.J., and Walsh, M.J. Artificial intelligence through a simulation of evolution, in biophysics and cybernetics systems. Proceedings of the Second Cybernetics Sciences Symposium, ed. by M. Maxfield, A. Callahan, and L.J. Fogel, Spartan Books, Washington, 1965.

[5] Goldberg, D.E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Mass., 1989. ISBN: 0-201-15767-5.

[6] Ray, T.S. Evolution, ecology and optimization of digital organisms. Santa Fe Institute Working Paper 92-08-042, 1992.

[7] Ray, T.S. and Hart, J. Evolution of differentiated multi-threaded digital organisms. Artificial Life VI Proceedings, ed. C.Adami et al., pp. 295-304, The MIT Press, Cambridge, 1998.

[8] Skomorokhov, A. O. Genetic algorithms: APL2 implementation and a real life application. APL Quote Quad (ACM SIGAPL), Vol. 26:4, 97-106, 1996.

[9] Volterra, V. Leçons sur la Théorie Mathématique de la Lutte pour la Vie. Gauthier-Villars, Paris, 1931.