UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba (PR)

LABIC
Laboratório de Bioinformática e Inteligência Computacional - UTFPR
http://labic.utfpr.edu.br

Computação Evolucionária

Prof. Heitor Silvério Lopes
hslopes@utfpr.edu.br
http://silverio.net.br/heitor

class7a (2020)

Programação Genética

- Parte I - Fundamentos:
  - Origens, literatura, contextualização, resolução de problemas, indução de programas
  - Conjuntos de funções e terminais, geração da população inicial, medidas de *fitness*, operadores principais e secundários, critério de parada
- Parte IIa – Aplicações:
  - Exemplos de aplicação clássicos (cap. 7):
    - Regressão simbólica
    - Formiga artificial
- Parte IIb – Aplicações:
  - Evolução de comportamentos emergentes (cap. 12):
    - Busca de alimento
    - Priorização de tarefas
  - Evolução da classificação (cap. 17):
    - Mineração de dados
    - Reconhecimento de padrões
  - Evolução de estratégias (cap. 15):
    - Estratégia de jogo perseguidor-evasor
  - Software Lil-GP

Evolução de Estratégias

(cap. 15)

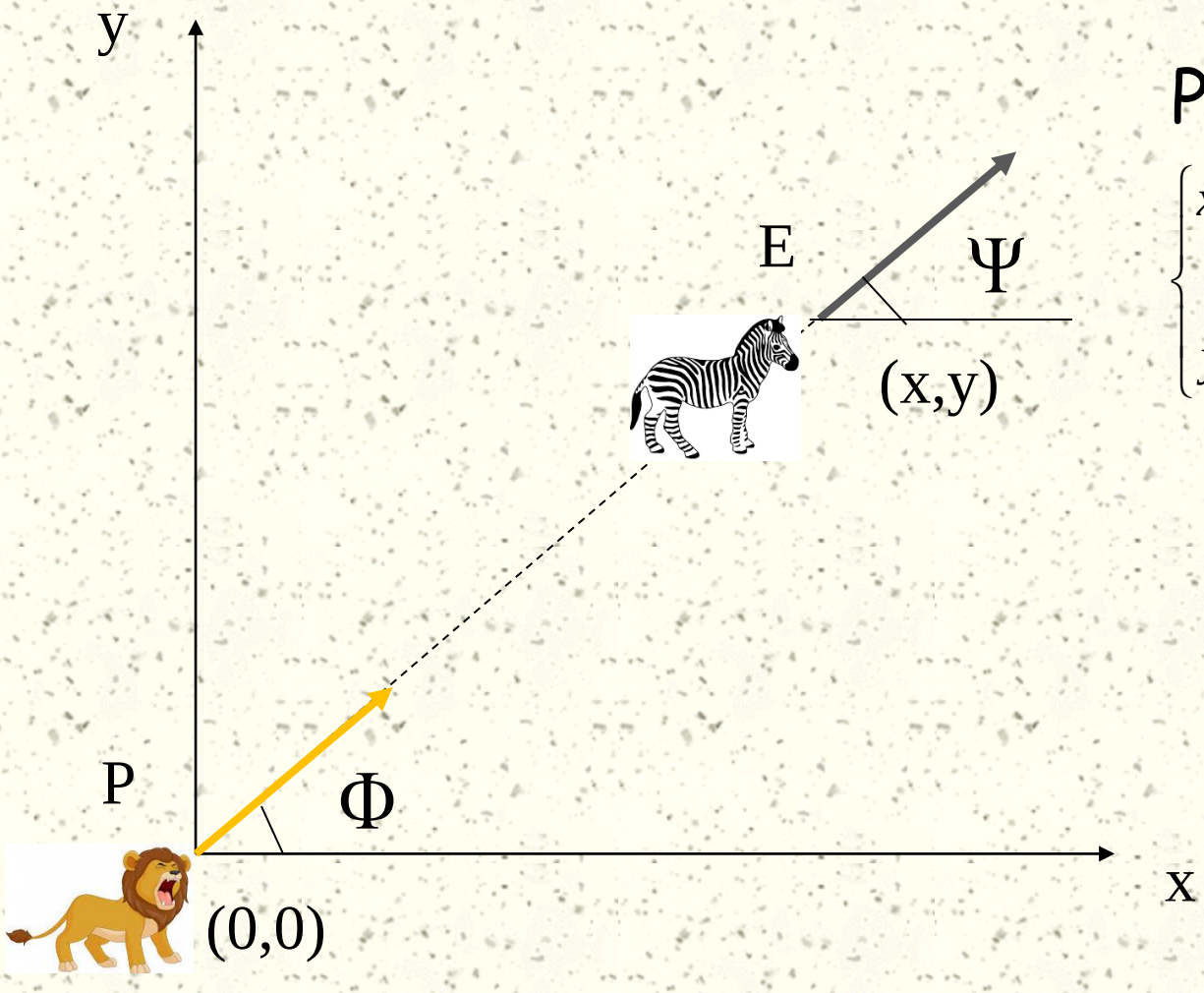
Evolução de Estratégias

- Problema : Encontrar uma estratégia para se jogar um jogo. Há poucas técnicas para a descoberta de estratégias de jogo
- Uma estratégia para um determinado jogador é uma maneira de especificar cada escolha que o jogador pode fazer a cada momento do jogo
- Descobrir uma estratégia para se jogar um jogo pode ser visto como "Encontrar um Programa", onde as entradas do programa podem ser:
  - A história completa dos movimentos do jogo
  - O estado atual do jogo

Estratégia de jogo perseguidor-evasor

- Inspiração na natureza: encontro predador-presa
  - Objetivo do perseguidor (predador):
    - Alcançar o evasor (presa)
  - Objetivo do evasor (presa):
    - Fugir do perseguidor (predador)
- Os objetivos são sempre conflitantes
- Perseguidor e evasor podem ter vantagens diferenciais que podem decidir o embate
- O cenário é dinâmico

Modelo matemático simples



Posição do evasor:

$$\begin{cases} x(t+1) = x(t) + w_e.\cos\psi - w_p.\cos\phi \\ y(t+1) = y(t) + w_e.sen\psi - w_p.sen\phi \end{cases}$$

Jogo diferencial Perseguidor-Evasor

- Um jogador perseguidor (P), mais rápido, tenta capturar um jogador evasor (E) mais lento
- Ambos se movem e tomam decisões simultaneamente
- A cada momento os jogadores podem escolher qual direção (ângulo) vão tomar
- As velocidades de P e E são: $w_p=1,0$ e $w_e=0,67$
- Variáveis de estado: (x_p, y_p) e (x_e, y_e)
- Variáveis de controle: ϕ e ψ $(0..2\pi)$
- Simplificação: o perseguidor está sempre na origem do sistema de coordenadas

Parâmetros da PG

- Objetivo: encontrar a melhor estratégia para o Perseguidor
- Conjunto de terminais: T = {x, y, R} .
  - onde R é uma constante aleatória efêmera, variando entre -1000 e + 1000.
- Conjunto de funções: F = {+, -, *, %, EXP, IFLTZ}
  - onde IFLTZ é uma operação condicional de três argumentos (IF a <O THEN b ELSE c)
- *Fitness cases*: 20 pontos iniciais diferentes para o Evasor.

Parâmetros da PG

- *Raw Fitness* : tempo requerido para capturar o Evasor, média dos 20 casos;
- Parâmetros padrão: M = 500 e G = 51.
- O perseguidor tem 100 espaços de tempo para capturar o Evasor

Resultados

- Melhor solução encontrada na geração 48.
- Expressão-S:

(% (+ IFLTZ (* X 0.6370001) (+ X X) (IFLTZ -0.674 YY)) (IFLTZ X (+ X Y) (*(IFLTZ (* X 0.6370001) (IFLTZ (* X X) (- X (EXP(- (%Y Y) (IFLTZ(* Y Y)) (* (- X 0.12900007) - 0.029999971) (+ -0.796 X))))) Y) IFLTZ (EXP(-(% (IFLTZ (* X 0.6370001) (+ X X) (- Y 0.129000007)) (- 0.992 Y)) (IFLTZ (IFLTZ Y Y X) Y X))) (+ % Y Y) (IFLTZ X (+ X Y) (+ (IFLTZ (* X 0.6370001) ((* (- X 0.12900007) - 0.029999971) (+ -0.796 X))))) Y) IFLTZ (EXP(-(% (IFLTZ (* X 0.6370001) (+ X X)))))

Conclusão

- É possível criar um programa de computador que funcione como uma estratégia para um determinado jogo.
- É muito importante o número de casos de *fitness* utilizados para se encontrar uma estratégia.

Possíveis aplicações práticas

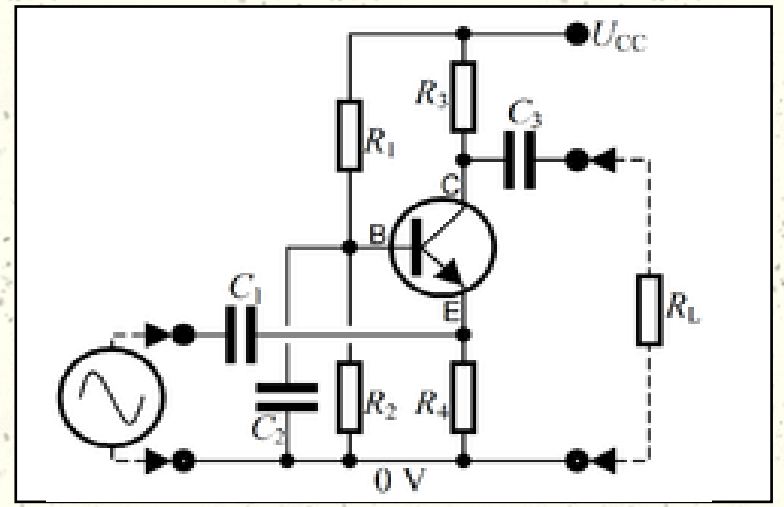
- Evolução de estratégias para:
  - Perseguição de alvos móveis
  - Evasão de mísseis
  - Futebol de robôs
  - Simulação de ecossistemas

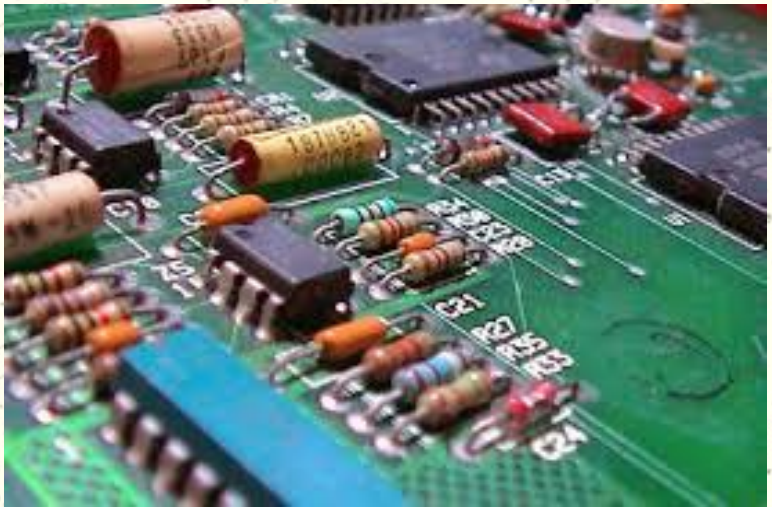
Extra

(Livro 2 Koza)

Projeto de circuitos eletrônicos

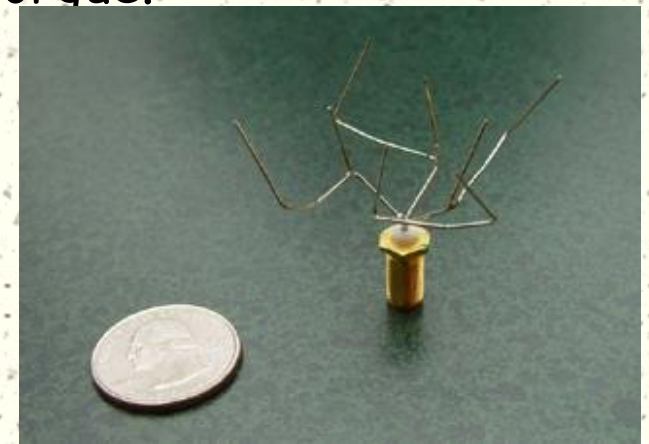
- O projeto de circuitos requer algumas escolhas pelo projetista:
  - Os componentes eletrônicos, suas especificações e seus valores
  - Um meio de combinar os compoentes obedecendo as restrições físicas e os princípios da eletricidade
  - Um meio de avaliar a capacidade do circuito de executar a tarefa para a qual foi projetado, respeitando as limitações técnicas estabelecidas no projeto: p.ex. consumo de energia, distorção harmônica, ganho, resposta em frequência, etc

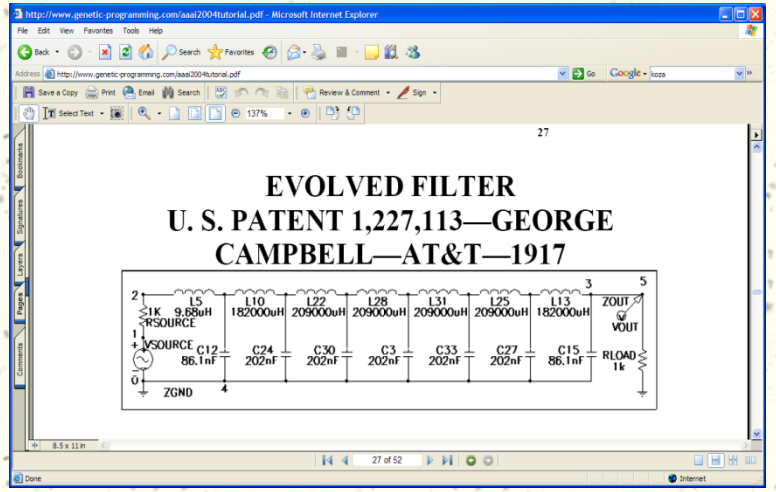


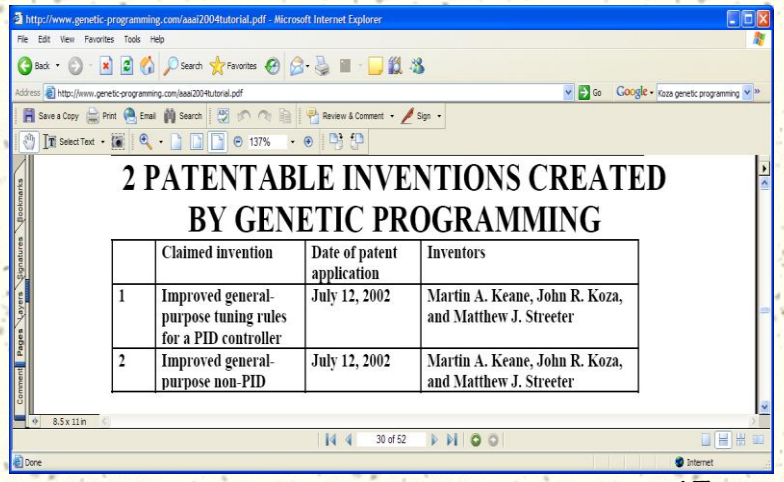


Projeto de circuitos eletrônicos

- Koza e colaboradores registraram patentes usando PG para evoluir circuitos eletrônicos.
- Alguns circuitos foram "reinventados"
- Pesquisadores da NASA projetaram uma antena para comunicação espacial utilizando PG. Ela funcionou melhor do que uma projetada por engenheiros, mas ninguém conseguiu explicar o porquê.







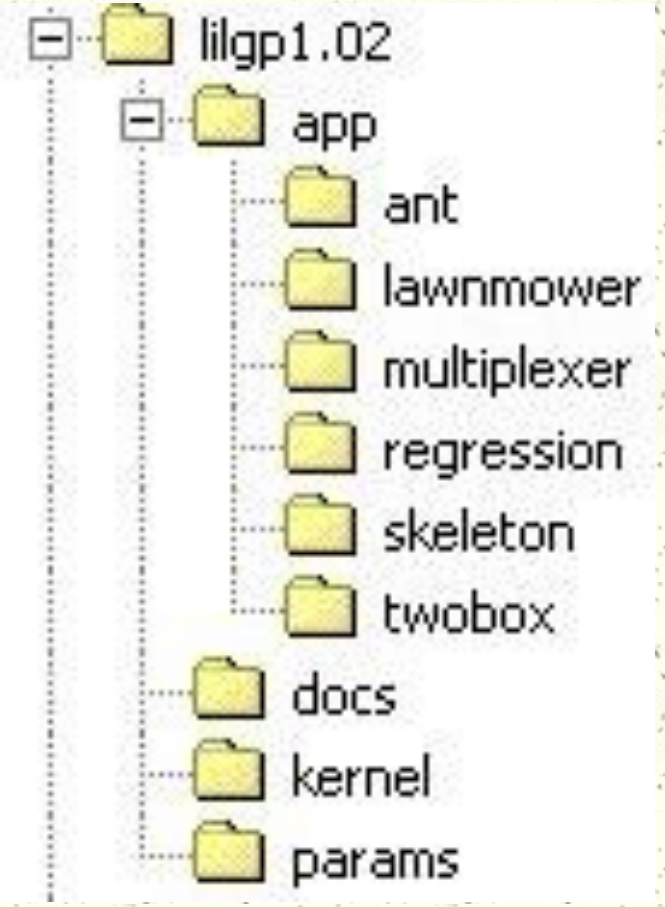
	Claimed invention	Date of patent application	Inventors
1	Improved general-purpose tuning rules for a PID controller	July 12, 2002	Martin A. Keane, John R. Koza, and Matthew J. Streeter
2	Improved general-purpose non-PID	July 12, 2002	Martin A. Keane, John R. Koza, and Matthew J. Streeter

Software Lil-GP

Características principais

- Implementado em ANSI-C com código aberto, versão atual 1.1 (set/98)
- 3 métodos de inicialização propostos por Koza: *full*, *grow* e *ramped-half-and-half*
- 7 métodos de seleção: *fitness*, *fitness* inverso, torneio, sobreseleção intensa, aleatório, melhor e pior
- 3 operadores genéticos: *crossover*, mutação e reprodução
- Limita o tamanho das árvores por números de nós e/ou por profundidade
- Permite o uso de constantes como terminais e a especificação de conjuntos de funções criadas pelo próprio usuário
- Permite múltiplas populações com diferentes parâmetros, topologia e política migratória arbitrárias
- Tem uma interface em Java para inicialização.
- Gera inúmeros arquivos com dados e estatísticas

Diretórios do Lil-GP



app: exemplos de aplicações

ant: formiga artificial

lawnmower: cortador de grama (usa ADFs)

multiplexer: multiplexador booleano de 11 entradas

regression: regressão simbólica

skeleton: "esqueleto" para implementar sua aplicação

twobox: problemas das duas caixas (usa ADFs)

docs: manual em pdf e txt

kernel: código fonte (.c e .h)

params: parâmetros utilizados no livro do Koza (94)

Arquivos de saída

- **.sys**: informações gerais de cada rodada
- **.gen**: estatísticas sobre o tamanho e profundidade das árvores
- **.prg**: estatísticas sobre o fitness e o número de acertos
- **.bst**: informações sobre o melhor indivíduo de cada rodads (best-of-run)
- **.his**: histórico do arquivo .bst
- **.stt**: versão condensada de todos os dados estatísticos (20) em cada rodada, num formato próprio para plotar gráficos

Principais parâmetros

```
pop_size = 500
max_generations = 50
random_seed = 1
output.basename = regress
output.detail = 50
init.method = half_and_half
init.depth = 2-6
max_nodes =
max_depth = 17
breed_phases = 2
breed[1].operator = crossover, select=fitness
breed[1].rate = 0.9
breed[2].operator = reproduction,select=fitness
breed[2].rate = 0.1
```

Roteiro para aplicação a novos problemas

- Escrever o código em C para cada função ou terminal, garantindo a condição do fechamento
- Escrever a função de *fitness* cru, podendo-se ou não utilizar os demais tipos de fitness definidos por Koza
- Definir os arquivos *event.c* e *event.h* a partir das opções disponíveis e do S.O. utilizado

Parâmetros específicos dos problemas-exemplo

- Regressão simbólica:
  - ***app.use_ercs***: utiliza ou não constantes [-1..1] como terminais
  - ***app.fitness_cases***: define o número de pontos a serem ajustados
  - ***app.value_cuttoff***: tolerância máxima entre cada valor ajustado e o valor real
- Formiga artificial:
  - ***app.maxtime***: número de passos permitidos
  - ***app.trail***: nome do arquivo que define a trilha (labirinto)
  - ***app.use_progrn4***: utiliza ou não a função PROGRN4

Versão para Windows (95/98 !!)

- *GPE – Genetic Programming Studio*, por Andres del Campo Novales (Univ. Córdoba, Espanha):
  - *http://www.uco.es/~i52canoa/www3/index_en.htm*
  - Baseia-se na versão 1.02
  - Interface gráfica para entrada dos parâmetros
  - Permite pausas, reinícios e salvamento de contexto
  - Permite a visualização e a exploração das características de cada indivíduo
  - Fornece gráficos *on-line* de *fitness* ajustado, complexidade estrutural e profundidade máxima
  - Versões em espanhol e em inglês

Estudio de Programación Genética - antstafe.epg

Archivo Ver Evolución Parámetros Individuos Ayuda

```
=== generación 27.
    evaluación completa.     (0s)
    producción completa.     (0s)
=== generación 28.
    evaluación completa.     (0s)
    producción completa.     (0s)
=== generación 29.
    evaluación completa.     (1s)
    producción completa.     (0s)
=== generación 30.
    evaluación completa.     (0s)
```

Resultado de la Evolución

Estadísticas aplicables a la subpoblación: 0 (0 = población completa)

Resultados

Máximo de nodos de un individuo:	77	Máximo número de aciertos:	69
Mínimo de nodos de un individuo:	1	Mínimo número de aciertos:	0
Número de nodos medio:	26.60	Media de aciertos:	43.71
Número de nodos del mejor individuo:	40	Mejor aptitud ajustada:	0.0476
Número de nodos del peor individuo:	24	Peor aptitud ajustada:	0.0111
Máxima profundidad de un individuo:	10	Aptitud ajustada media:	0.0278
Mínima profundidad de un individuo:	0	Mejor generación:	31
Profundidad media:	4.65	Peor generación:	31
Profundidad del mejor individuo:	7	Mejor subpoblación:	1
Profundidad del peor individuo:	4	Peor subpoblación:	1

Resultados aplicables a la: (•) Generación ( ) Ejecución Ayuda

Características del Individuo

Subpoblación: 1 Individuo: 1 Retardo de evaluación (mseg): 100

Características del individuo

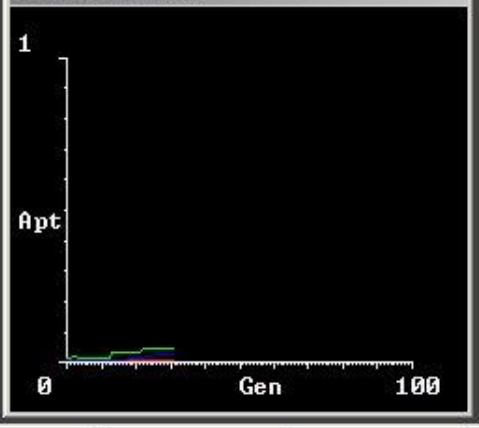
Profundidad:	7	Aptitud cruda:	69.00
Número de nodos:	40	Aptitud estandarizada:	20.00
Número de aciertos:	69	Aptitud ajustada:	0.0476

Composición del individuo

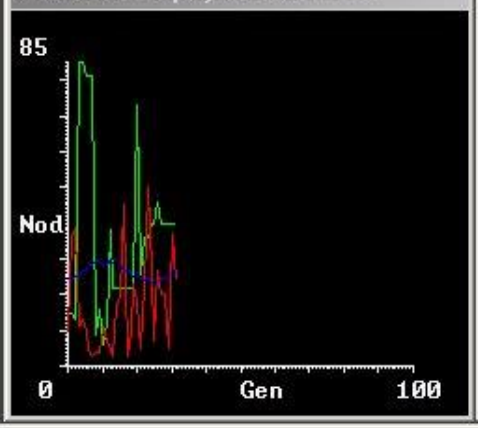
```
TREE:
 (progn3 (if-food-ahead (progn2 (if-food-ahead (if-food-ahea
                                                (progn2 left
                                                        (if-f
                                (if-food-ahead (if-food-ahea
                                                (progn2 left
                                                        (progn
                        (if-food-ahead move left))
         (if-food-ahead (if-food-ahead right left)
                        (if-food-ahead move left))
         (progn2 left
                 (if-food-ahead move
                                (progn2 left move))))
```

Evaluar Ordenar Copiar Ayuda Cerrar

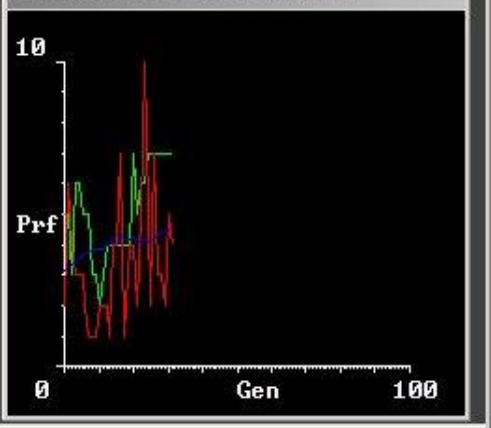
Gráfica de Aptitud



Gráfica de Complejidad Estructural



Gráfica de Profundidad del Árbol



Subpobl: 1 | Indiv/Subp: 500 | Gen: 31/100 | Semilla: 21685 | PAUSA / 1 GEN

Genetic Programming in Python

- https://github.com/trevorstephens/gplearn

Genetic Programming in Python, with a scikit-learn inspired API:

gplearn

- http://pystep.sourceforge.net/

pySTEP

Python Strongly Typed gEnetic Programming

- https://deap.readthedocs.io/en/master/

DISTRIBUTED EVOLUTIONARY ALGORITHMS IN PYTHON