Springbots

Experimênto com Algoritmos Genéticos

A última versão deste documento, e outros formatos, podem ser encontrados em http://springbots.sourceforge.net/doc/.

A página oficial do projeto está hospedada em http://springbots.sourceforge.net/.

Rodrigo Setti <rodrigosetti@gmail.com>

Table of Contents

1 Introdução

Um algoritmo genético (AG) é uma técnica de procura utilizada na ciência da computação para achar soluções aproximadas em problemas de otimização e busca. Algoritmos genéticos são uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva como hereditariedade, mutação, seleção natural e recombinação (ou crossing over).

Algoritmos genéticos são implementados como uma simulação de computador em que uma população de representações abstratas de solução é selecionada em busca de soluções melhores. A evolução geralmente se inicia a partir de um conjunto de soluções criado aleatoriamente e é realizada através de gerações. A cada geração, a adaptação de cada solução na população é avaliada, alguns indivíduos são selecionados para a próxima geração, e recombinados ou mutados para formar uma nova população. A nova população então é utilizada como entrada para a próxima iteração do algoritmo.

Algoritmos genéticos diferem dos algoritmos tradicionais de otimização em basicamente quatro aspectos:

- Se baseiam em uma codificação do conjunto das soluções possíveis, e não nos parâmetros da otimização em si;
- os resultados são apresentados como uma população de soluções e não como uma solução única;
- não necessitam de nenhum conhecimento derivado do problema, apenas de uma forma de avaliação do resultado;
- usam transições probabilísticas e não regras determinísticas.

1.1 Algoritmo

Os Algoritmos genéticos são em geral algoritmos simples e fáceis de serem implementados. Segue abaixo um trecho de pseudo-código descrevendo um algoritmo genético:

```
função AlgoritmoGenético(população, função-objetivo) saídas: indivíduo
  entradas: população -> uma lista de indivíduos
            função-objetivo -> recebe um indivíduo e retorna um número real.
  repetir
     lista de pais := seleção(população, função-objetivo)
     população := reprodução(lista de pais)
  enquanto nenhuma condiçao de parada for atingida
  retorna o melhor indivíduo de acordo com a função-objetivo
```

A função objetivo é o objeto de nossa otimização. Pode ser um problema de otimização, um conjunto de teste para identificar os indivíduos mais aptos, ou mesmo uma "caixa preta" onde sabemos apenas o formato das entradas e nos retorna um valor que queremos otimizar. A grande vantagem dos algoritmos genéticos esta no fato de não precisarmos saber como funciona esta função objetivo, apenas tê-la disponível para ser aplicada aos indivíduos e comparar os resultados.

O indivíduo é meramente um portador do seu código genético. O código genético é uma representação do espaço de busca do problema a ser resolvido, em geral na forma de

seqüências de bits. Por exemplo, para otimizações em problemas cujos valores de entrada são inteiros positivos de valor menor que 255 podemos usar 8 bits, com a representação binária normal, ou ainda uma forma de código gray. Problemas com múltiplas entradas podem combinar as entradas em uma única seqüência de bits, ou trabalhar com mais de um "cromossomo", cada um representando uma das entradas. O código genético deve ser uma representação capaz de representar todo o conjunto dos valores no espaço de busca, e precisa ter tamanho finito.

A seleção também é outra parte chave do algoritmo. Em geral, usa-se o algoritmo de seleção por "roleta", onde os indivíduos são ordenados de acordo com a função-objetivo e lhes são atribuídas probabilidades decrescentes de serem escolhidos. A escolha é feita então aleatóriamente de acordo com essas probabilidades. Dessa forma conseguimos escolher como pais os mais bem adaptados, sem deixar de lado a diversidade dos menos adaptados. Outras formas de seleção podem ser aplicadas dependendo do problema a ser tratado.

A reprodução, tradicionalmente, é divididas em três etapas: acasalamento, recombinação e mutação. O acasalamento é a escolha de dois indivíduos para se reproduzirem (geralmente gerando dois descendentes para manter o tamanho populacional). A recombinação, ou crossing-over é um processo que imita o processo biológico homônimo na reprodução sexuada: os descendentes recebem em seu código genético parte do código genético do pai e parte do código da mãe. Esta recombinação garante que os melhores indivíduos sejam capazes de trocar entre si as informações que os levam a ser mais aptos a sobreviver, e assim gerar descendentes ainda mais aptos. Por último vem as mutações, que são feitas com probabilidade a mais baixa possível, e tem como objetivo permitir maior variabilidade genética na população, impedindo que a busca fique estagnada em um mínimo local.

1.2 Programação Genética

Por ser um algoritmo extremamente simples e eficiente, existem diversas variações em cima do algorítmo genético básico para se obter resultados melhores ou mesmo tratar novas classes de problemas. Uma dessas variações é a Programação genética. Na Programação genética os indivíduos representam pequenos programas de computador que serão avaliados de acordo com o resultado de sua execução. Estes programas podem ser expressões simples, como fórmulas aritméticas ou programas complexos, com operações de laço e condicionais, típicas de uma linguagem de programação comum.

2 Projeto

O objetivo deste projeto é realizar uma demonstração do uso de um algoritmo genético para solucionar um problema. Apresentaremos adiante o "genoma", que é o objeto que estaremos otimizando. O "problema", ou seja, aonde se quer chegar com o genoma e que tipos de solução são melhores ou piores e porquê. O "método" utilizado, e finalmente instruções para o usuário configurar e rodar seus próprios experimentos locais ou em rede, salvar e analizar os resultados da otimização e estatísticas produzidas.

2.1 Genoma

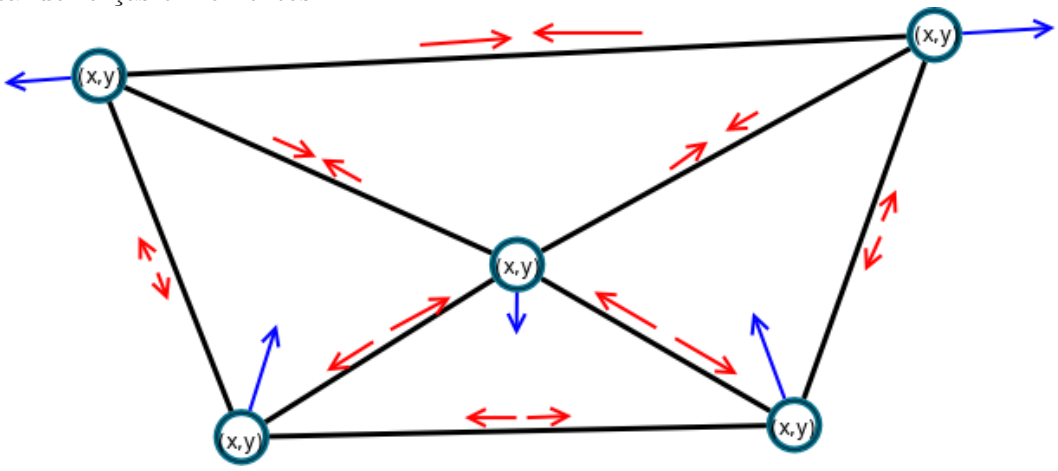
Neste projeto procuramos otimizar o que chamamos de "Springbots". Um springbot pode ser entendido como uma "criatura virtual" que vive em um mundo físico simulado. De maneira mais clara, um springbot é um grafo conexo cujos vértices possuem atributos físicos tais como inércia, posição e momentum linear. As arestas, por sua vez, exercem forças elásticas entre os vértices mantendo a "forma" da criatura, além de oferecerem resistência em ambientes líquidos causando empuxo.

As arestas também podem ter movimento rítmico cuja frequência é senoidal. A amplitude e deslocamento são atributos variáveis nas arestas.

Simulados em um ambiente bidimensional com gravidade e chão um springbot interage de maneira muito similar à objetos físicos reais, respondendo à colisões e à gravidade. Em ambientes líquidos, os springbots ao se moverem causam forças de empuxo em diferentes direções podendo provocar movimentação do centro de massa(nado).

Os springbots são persistidos em formato digital através do uso de um dialeto de XML. Um arquivo XML pode armazenar não somente um, mas uma população de springbots. Os dados salvos correspodem somente ao genoma e não aos atributos físicos correntes(momentum), entretanto, a posição atual da função seno é salva para que a criatura não perca a "sincronia".

A figura abaixo é uma representação gráfica de um springbot com alguns vetores representando forças e momentos:



Este é um exemplo de um arquivo XML descrevendo apenas um genoma:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE springbots SYSTEM
          "http://springbots.sourceforge.net/springbots.dtd">
<springbots>
    <springbot angle="0" name="Swarm-A Abem Ufo" generation="7"
     adapted="swim" bloodline="0:b:8:1:2:5:0">
      <node pos="-225,8" id="S0N1" />
      <node pos="-175,18" id="S0N2" />
      <node pos="-125,8" id="S0N3" />
      <node pos="-75,18" id="S0N4" />
      <node pos="-25,8" id="S0N5" />
      <node pos="25,18" id="S0N6" />
      <node pos="75,8" id="S0N7" />
      <node pos="125,18" id="S0N8" />
      <node pos="175,8" id="S0N9" />
      <node pos="225,18" id="S0N10" />
      <spring from="S0N1" to="S0N2" offset="0.000" amplitude="0.600" />
      <spring from="S0N2" to="S0N3" offset="0.628" amplitude="0.600" />
      <spring from="S0N3" to="S0N4" offset="1.257" amplitude="0.600" />
      <spring from="S0N4" to="S0N5" offset="1.885" amplitude="0.600" />
      <spring from="S0N5" to="S0N6" offset="2.513" amplitude="0.600" />
      <spring from="S0N6" to="S0N7" offset="3.142" amplitude="0.600" />
      <spring from="S0N7" to="S0N8" offset="3.770" amplitude="0.600" />
      <spring from="S0N8" to="S0N9" offset="4.398" amplitude="0.600" />
      <spring from="S0N9" to="S0N10" offset="5.027" amplitude="0.600" />
      <spring from="S0N1" to="S0N3" offset="0.000" amplitude="0.600" />
      <spring from="S0N2" to="S0N4" offset="0.628" amplitude="0.600" />
      <spring from="S0N3" to="S0N5" offset="1.257" amplitude="0.600" />
      <spring from="S0N4" to="S0N6" offset="1.885" amplitude="0.600" />
      <spring from="S0N5" to="S0N7" offset="2.513" amplitude="0.600" />
      <spring from="S0N6" to="S0N8" offset="3.142" amplitude="0.600" />
      <spring from="S0N7" to="S0N9" offset="3.770" amplitude="0.600" />
      <spring from="S0N8" to="S0N10" offset="4.398" amplitude="0.600" />
    </springbot>
</springbots>
```

O descritor DTD oficial pode ser encontrado em springbots.sourceforge.net/springbots.dtd.

É importante discriminar os atributos do elemento “springbot”:

name O nome do springbot. O primeiro nome é a “linhagem”, nunca muda e é usado para identificar a descendência original.

angle Valor do deslocamento do ângulo da função seno, cujo valor é utilizado pelas arestas para calcular o movimento.

generation
Número da geração. o descendênte original inicia em zero, e cada filho herda o valor da geração do pai somado de um.

adapted O springbot pode discriminar o ambiente ou função para o qual está adaptado.

bloodline A linha de descendência é uma sequência de números hexadecimais separados por dois pontos e identifica o rastro de ancestralidade da criatura. Cada irmão tem o mesmo tamanho de linha de descendência e são idênticos exceto pelo último número, e a cada geração um número é adicionado à linha. Este valor serve para recriar árvores de descendência.

Os outros elementos do XML são "node" e "spring" e descrevem "vértices" e "arestas" respectivamente. Basicamente o "node" descreve sua posição e possui um identificador único. As arestas dizem quais vértices ligam, pelos identificadores, e possuem os atributos de movimento "offset" e "amplitude".

O projeto acompanha um programa editor de springbots, em '`editor/editor.py`'. É muito recomendável o usuário utilizar o programa para se acostumar com a construção e comportamento dos springbots. O editor tem uma interface bem simples para manipular as criaturas, baseado em mouse e *drag-and-drop*, com a possibilidade de abrir e salvar genomas.

2.2 Adaptabilidade

A adaptabilidade, ou *fitness*, é a medida de quão otimizado ou adequado é um genoma para resolver um determinado problema, ou meta. O projeto define inicialmente cinco tipos de metas para as criaturas, podendo ser extendido facilmente para mais por alguém com criatividade e conhecimento de programação python. Cada meta ou função *fitness* tem um método de avaliação do genoma que resulta em uma nota, um número. Quanto maior esta nota, mais adaptado ou adequado está o genoma ao problema específico.

Apresentamos os cinco tipos de função *fitness* e o método de operação destas:

walk Simula o springbot em um ambiente com gravidade e chão por um intervalo de tempo. O valor do *fitness* é a diferença horizontal do centro de massa inicial e final, desta maneira, esta função recompensa velocidade horizontal de forma que as criaturas capazes de percorrer a maior distância no intervalo tem uma nota maior.

swim Simula o springbot em um ambiente líquido, sem gravidade e barreiras. O valor do *fitness* é a distância absoluta entre o centro de massa inicial e final, recompensando a velocidade de nado.

jump O springbot é simulado em um ambiente com gravidade e chão. O valor de *fitness* é a diferença vertical entre a menor e maior altura de centro de massa adquirida, recompensado a capacidade de saltar.

height Em um ambiente com gravidade e chão, o valor da função é a proporção entre a altura média pela largura média do springbot. Esta função recompensa a altura relativa em relação a altura, incentivando soluções de estrutura "finas".

equilibrium Também em um ambiente com gravidade e chão, o valor da função é a proporção entre o centro de massa e a altura. Recompensando soluções com o centro de massa mais próximos da altura, logo, incentivando estruturas com massa(vértices) concentrados no topo.

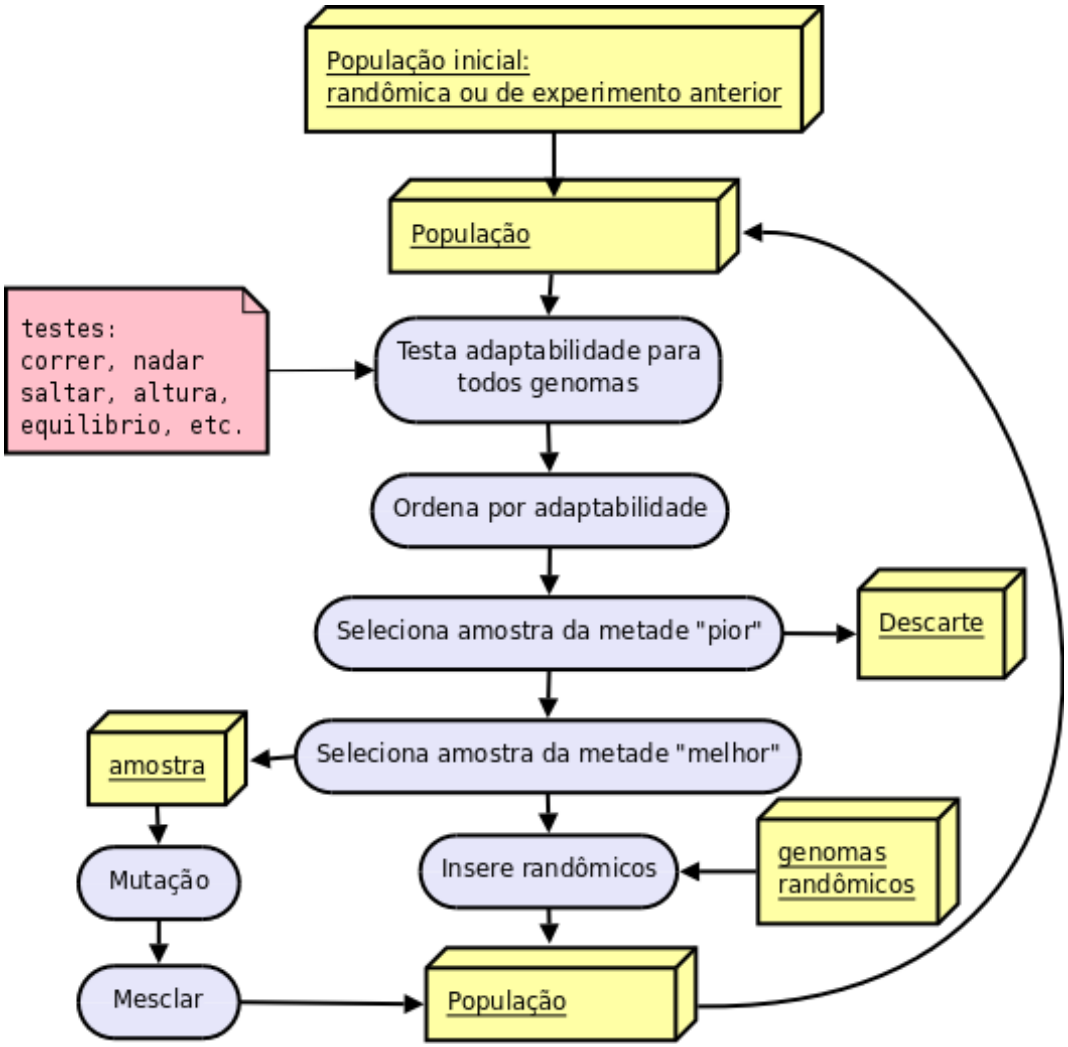
O objetivo do algoritmo genético é otimizar as soluções para a função *fitness* especifica. Cada experimento deve escolher uma destas(ou outra) função, e a população sofrerá seleção baseada na nota de adaptabilidade.

2.3 Modus Operandi

Agora que conhecemos o objeto de otimização, o springbot, e as funções de adaptabilidade, veremos como o sistema emprega um tipo de algoritmo genético para atingir o objetivo proposto. O algoritmo é muito similar ao pseudo-código apresentado na introdução, mas algumas diferenças foram introduzidas para incentivar a diversidade de soluções. De maneira geral, o processo funciona da seguinte maneira:

```
população = população inicial(randômica ou de outro experimento)

repetir:
        para cada indivíduo em população:
             testa fitness e atribui nota

        ordena população pela nota de fitness
        realiza amostragem para descarte(morte) da metade inferior
        filhos = cópia amostragem da metade superior(reprodução)

        para cada indivíduo em filhos:
             realiza mutação em filhos

        insere filhos na população

enquanto nenhuma condição de parada for atingida
```

A figura abaixo mostra o mesmo algoritmo em formato de fluxograma:



Após muitas iterações do *loop* principal é esperado que a nota média da população tenda ao crescimento. Experimentos mostraram que este crescimento se da de maneira logaritmóide.

A mutação é uma operação sobre o genoma que muda algum atributo de algum elemento do grafo ou muda a própria topologia do grafio adicionando ou removendo algum elemento, de maneira geral, a mutação é uma mudaça mínima e randômica no genoma causando uma variação.

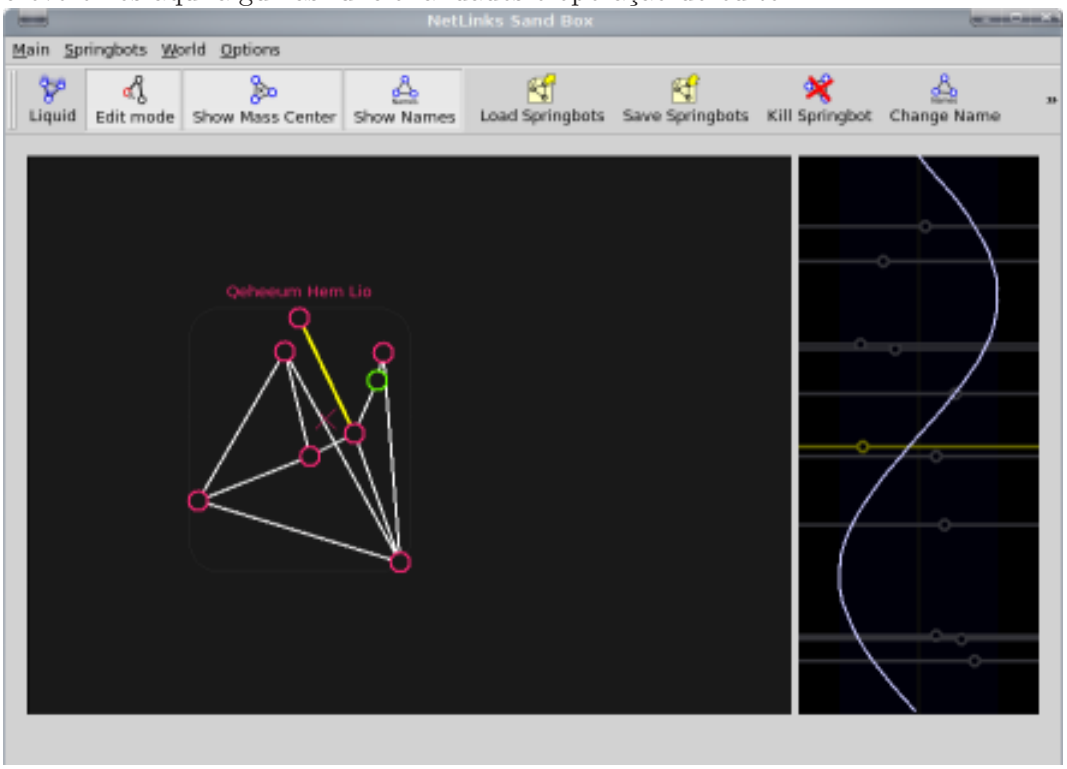
Como existe no processo muita aleatoriedade(mutação e amostragens) é muito improvável, na prática impossível, de uma mesma entrada(população e *fitness*) produzirem após um mesmo número de iterações um mesmo resultado.

2.4 Ferramentas

Nesta seção apresentamos as ferramentas úteis para conduzir experimentos, vizualizar e editar springbots e analisar dados de estatísticas.

2.4.1 Editor

O editor de springbots é o primeiro programa recomendado para o usuário compreender intuitivamente do que se trata o springbot e que tipos de atributos seu genôma contem. Descreveremos aqui algumas funcionalidades e operação do editor.



2.4.1.1 Barra de ferramentas

Liquid Muda para ambiente líquido(sem chão e gravidade)/não líquido(com chão e gravidade).

Edit mode Muda para modo de ediçao/simulação.

Show Mass Center
Mostra centro de massa dos springbots.

Show Names
Mostra nome dos springbots.

Load Springbots
Carrega arquivo XML de genomas.

Save Springbots
Salva springbots em arquivo XML de genomas.

Kill Springbot
Mata(elimina) springbot selecionado.

Change Name
Editar nome do springbot selecionado.

Mutate Efetua uma mutação aleatória no springbot selecionado.

New Random
Insere um springbot gerado randomicamente na área de edição.

2.4.1.2 Área de edição/simulação

Quando no modo de edição(ver barra de ferramentas):

Criar novo vértice
Clicar com o botão direito do mouse em um local vazio(cria novo springbot).

Mover vértice
Arrastar vértice com botão esquerdo do mouse.

Criar aresta para novo vértice
Clicar com o botão direito sobre um vértice e arrastar até uma posição vazia.

Criar aresta para vértice existente da mesma criatura
Clicar com o botão direito sobre um vértice e arrastar até outro do mesmo grafo.

Remover vértice
Clicar com botão do meio sobre o vértice(todas as arestas ligantes também serã eliminadas).

Remover aresta
Clicar com o botão do meio sobre a aresta(o grafo será reduzido se tornar-se desconexo).

Selecionar aresta
Clicar com o botão esquerdo do mouse sobre a aresta.

Selecionas springbot
Clicar com o botão direito sobre qualquer elemento da criatura(vértice ou aresta).

Observe que é impossível ligar por uma aresta dois springbots diferentes.

Quando no modo de simulação não é possível editar a topologia nem criar novos springbots, entretanto, é possível mover os vértices e editar atributos de movimento das arestas(veja no tópico adiante).

2.4.1.3 Área da função senoide

A área da função senoide(a direita da interface) é uma região para editar os atributos de movimentação das arestas. Cada aresta possui dois parâmetros de movimentação: “amplitude” e “deslocamento”. Estes atributos são representados graficamente na área da senoide.

Cada aresta tem uma barra com um nó desenhados na área da senoide. A posição vertical da barra é o deslocamento do movimento aresta em relação a função senoide. A posição horizontal do nó é a amplitude do movimento da aresta, de maneira que quanto mais distânte do centro maior o movimento. se o nó está no centro, a amplitude(e movimento) é nula.

2.4.2 Geração randômica

O programa '`randombot.py`' é utilizado para gerar populações de springbots randômicos. Úteis para servir de entrada para experimentos. O programa recebe alguns parâmetros da linha de comando e escreve na saída padrão(eg. `stdout`) o arquivo XML. Os argumentos da linha de comando são:

-p <número> ou –population=<número>
: População de springbots randômicos a serem gerados

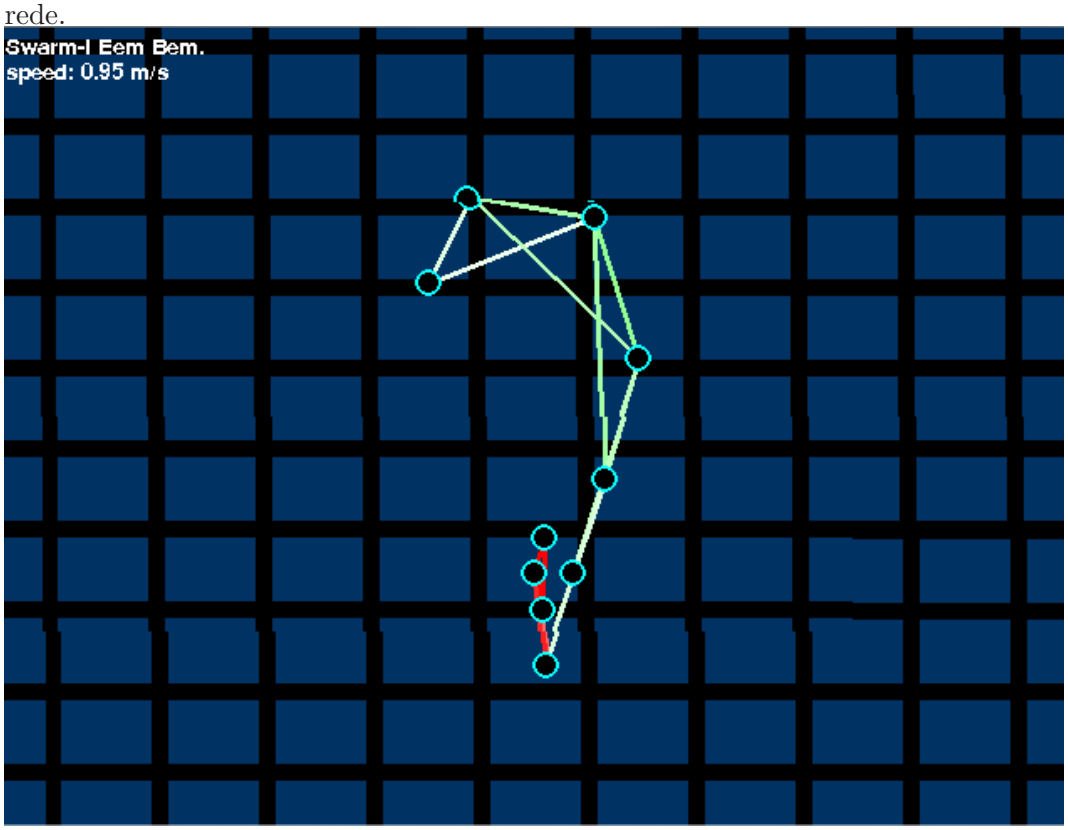
-n <vértices> ou –nodes=<vértices>
: Número de vértices de cada genoma

-s <arestas> ou –springs=<arestas>
: Número de arestas de cada genoma

-r <raio> ou –noderadius=<raio>
: Distância(raio) máximo entre os vértices gerados

2.4.3 Evolve

O programa evolve é o núcleo do projeto, nele está implementado o algoritmo genético que irá "evoluir" a população para uma função especificada. O evolve vem em duas versões: a versão local('`evolve.py`') e a versão *network*('`net-evolve.py`') que pode ser distribuída em rede.



Descreveremos primeiro as características comuns às duas versões para depois entrar em detalhes sobre a operação específica de cada uma.

O evolve recebe via entrada padrão(eg. `stdin`) o XML da população inicial do experimento e escreve opcionalmente(se a opção verbose estiver setada)na saída padrão(eg. `stdout`) os dados estatísticos do que está ocorrendo no experimento. A cada determinado número de iterações(configurável pelo usuário) o programa escreve um arquivo XML com o nome '`<prefixo>-<fitness>-p<tamanho>-i<iterações>.xml`', sendo `<prefixo>` um nome identificador do experimento, `<fitness>` o nome da função *fitness* aplicada, `<tamanho>`, o tamanho da população e `<iterações>`, o número de iterações daquela população. Por exemplo, um arquivo válido pode ser '`joao-walk-p100-i67.xml`'.

Os parâmetros aceitos pelo evolve na linha de comando são:

-v ou –verbose
: Escreve dados estatísticos do experimento na saída padrão.

-f <fitness> ou –fitness=<fitness>
: Nome da função fitness aplicada, por padrão *walk*.

-P <prefixo> ou –prefix=<prefixo>
: Especifica um nome para prefixar nos arquivos de população salvos. Isto é útil para não confundir saídas de diferentes experimentos. Se não for especificado, o programa cria um nome randômico para servir de prefixo.

-b ou –best
: Salva o melhor espécime a cada iteração, em um mesmo arquivo, no formato '`<prefixo>-<fitness>-p<população>-best.xml`', eg. '`exp13-walk-p100-best.xml`'.

-p <arquivo.xml> ou –population=<arquivo.xml>
: Ao invés de ler a população da entrada padrão, lê do arquivo especificado.

-s <valor> ou –save-freq <valor>
: A cada determinado número de iterações(padrão 100), especificado por `<valor>`, salva a população em um arquivo xml no formato '`<prefixo>-<fitness>-p<tamanho>-i<iterações>.xml`', eg. '`maria-swim-p80-i22.xml`'.

-l <valor> ou –limit=<valor>
: Limita em `<valor>` o número máximo de iterações. Por padrão o experimento nunca acaba e é interrompido somente pelo sinal de interrupção de teclado do usuário.

-a <iteracao> ou –start_at<iteracao>
: Inicia o número da primeira iteração com um valor especificado, ao invés de zero. É útil para manter a sequência de arquivos salvos de um experimento interrompido e continuado.

2.4.3.1 Local evolve

O programa '`evolve.py`' aceita todos os parâmetros de linha de comando descrito anteriormente e mais os seguintes:

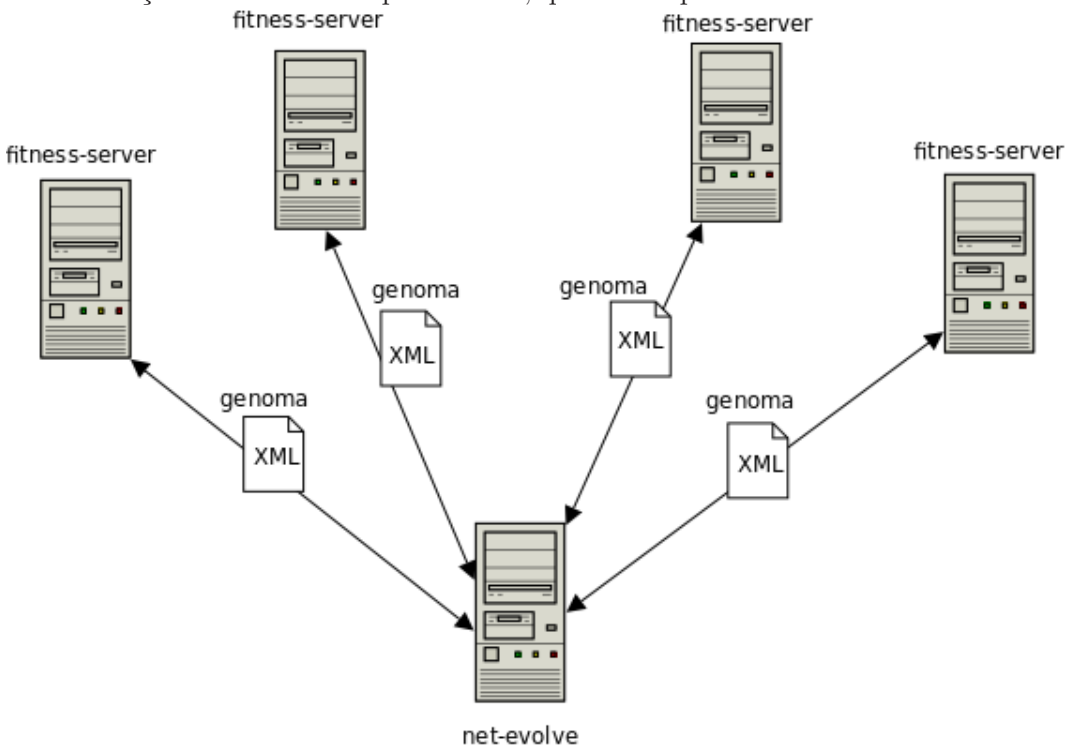
-g ou –graphics
: Ativa gráficos em tempo real das simulações(requer pygame), este parâmetro pode desacelerar o experimento mas é bem interessante.

-F ou –fullscreen
Se os gráficos estão ativos, mostra em tela cheia.

O evolve testa cada indivíduo na população de modo serial e opcionalmente mostra o gráfico em tempo real do que está ocorrendo.

2.4.3.2 Network evolve

O sistema de distribuição em rede do projeto Springbots é baseado na arquitetura cliente-servidor. O cliente é o programa '`net-evolve.py`', e em um experimento existe somente um cliente rodando. Ele é reponsável pelo algoritmo genético, seleção natural, mutação, etc. Com exceção do teste de adaptabilidade, que é feito por um dos servidores conectados.



Cada servidor é uma instância do programa '`fitness-server.py`' rodando na rede, e basicamente o que o servidor faz é receber requisições de testes de adaptabilidade, executar o teste e devolver a nota. Os parâmetros de linha de comando para iniciar o servidor são:

-p <porta> ou –port=<porta>
Porta TCP que o servidor irá escutar por conexões(padrão 8088).

-v ou –verbose
Escreve informações dos testes na saída padrão.

-g ou –graphics
Ativa gráficos em tempo real das simulações(requer pygame), este parâmetro pode desacelerar o experimento mas é bem interessante.

-F ou –fullscreen
Se os gráficos estão ativos, mostra em tela cheia.

O ‘net-evolve.py’ se conecta a um conjunto de servidores e envia simultaneamente pedidos de testes para os servidores disponíveis, entretanto, é necessário que o programa cliente saiba os endereços de rede aonde os servidores estão rodando, isto é feito com o uso de um arquivo de texto de configuração bem simples, no seguinte formato, por exemplo:

```
# Lista de servidores:

http://192.168.0.100:8088
http://192.168.0.101:8088
http://192.168.0.102:8088
http://192.168.0.103:8088
http://192.168.0.104:8088
http://192.168.0.105:8088
```

O programa ignora linhas iniciadas com “#”(comentários) e linhas em branco. Cada linha especificando o servidor deve ser uma URL http válida seguido por “:” e a porta TCP.

As opções de linha de comando do ‘net-evolve.py’ são as genéricas do evolve somadas à mais uma

-n <arquivo> ou –serverslist=<arquivo>
: Especifica o arquivo de texto de configuração contendo o endereço dos servidores de teste(padrão ‘fitness-servers.txt’).

2.4.3.3 Exemplos de Simulações

Os exemplos desta sessão pressupõem um shell unix, entretanto, no Windows é possível executas os scripts chamando o interpretador python(instalado geralmente em ‘C:\Python25\python.exe’).

```
# Evolui uma populacao inicial aleatoria de 100 springbots para nadar,
# com saída grafica em tela-cheia e log salvo
# no arquivo ‘exp90-walk-p100.log’

$ ./randombot.py -p 100 | ./evolve.py -vgFf walk -P exp90 > \
  exp90-walk-p100.log

# Inicia dois servidores locais e evolui uma populacao de um
# experimento anterior à partir da iteração 400 para
# nadar(swim), salvando o log em um arquivo.

$ ./fitness-server.py -p 8089 &
$ ./fitness-server.py -p 8090 &
$ cat 'http://localhost:8089 > servidores.txt
$ cat 'http://localhost:8090 >> servidores.txt
$ ./net-evolve.py -vn servidores.txt -f swim -a 400 -P exp88 \
  < exp88-swim-i400-p100.xml > exp88-swim-p100.log
```

Em sistemas que não suportam pipe(|) uma alternativa é direcionar a saída para um arquivo para em seguida em outro comando direcionar esse arquivo para a entrada.

2.4.4 Estatísticas

Tanto o 'evolve.py' quanto o 'net-evolve.py', com a opção -v ou --verbose escrevem na saída padrão estatísticas acerca do experimento envolvido. Um exemplo desta saída é:

```
# exp_003 experiment
# Initiating simulation with a population of 300 specimens.
# Evolving for walk:
# At each iteration 60 will be discarded, 45 of the remaining will be selected cloned
Iteration 0:
        1/300: "Aulaumnum"(1) 137.619
        2/300: "Prioppegrum"(1) 65.862
        3/300: "Proittodu"(1) 24.481
        4/300: "Dioprialia"(1) 72.372
        5/300: "Ujiamno"(1) 21.781
        6/300: "Foicizae"(1) 115.870
        7/300: "Phiaphittum"(1) 55.026
        8/300: "Uhiapha"(1) 340.431
        9/300: "Faettiavum"(1) 75.965
        10/300: "Iolupae"(1) 59.812
        11/300: "Ilaevio"(1) 218.874
        12/300: "Coidiaee"(1) 0.165
        13/300: "Zaraesta"(1) 0.482
        14/300: "Droimmuppia"(1) 16.799
        15/300: "Ahagru"(1) 58.451
        16/300: "Friqistau"(1) 53.140
        17/300: "Mieammem"(1) 59.089
        18/300: "Riapparum"(1) 27.444
...
        294/300: "Jaedisia"(1) 42.492
        295/300: "Ioxuzo"(1) 142.339
        296/300: "Loibiacci"(1) 137.999
        297/300: "Egiase"(1) 220.619
        298/300: "Audroivem"(1) 114.466
        299/300: "Oijauxu"(1) 34.296
        300/300: "Aulupia"(1) 29.118
Bloodline lenght average: 1.0000
Fitness average: 77.2136
Iteration 1:
        1/300: "Uhiapha"(1) 0.969
        2/300: "Hummallo"(1) 273.360
        3/300: "Rursebre"(1) 278.580
        4/300: "Aegumnem"(1) 197.639
        5/300: "Qodiahum"(1) 357.095
        6/300: "Loigrolum"(1) 31.281
        7/300: "Ivoigoi"(1) 111.583
        8/300: "Ifroppem"(1) 329.433
```

```
...
        294/300: "Phaummaumu"(1) 39.388
        295/300: "Ojorsoi"(1) 11.525
        296/300: "Uttoifrem"(1) 28.324
        297/300: "Ugraumma"(1) 138.344
        298/300: "Lopriobum"(1) 67.797
        299/300: "Auroidri"(1) 50.479
        300/300: "Uxiocem"(1) 70.993
Bloodline lenght average: 4.4267
Fitness average: 283.4820
Iteration 34:
        1/300: "Hummallo Aefroi Qe"(4) 415.008
        2/300: "Hummallo Iabrem Dum"(5) 619.966
        3/300: "Hujaegrem Aphem"(2) 393.573
        4/300: "Friqistau Oiphia Sti"(4) 598.703
        5/300: "Hummallo Hum Aefroi"(3) 544.045
        6/300: "Driapiolia Oirsem Ra"(10) 582.507
        7/300: "Ciosaemnoi Appa Aeeem"(5) 477.941
        8/300: "Hummallo So Rem"(5) 327.234
        9/300: "Hummallo Hum"(2) 485.291
        10/300: "Ciosaemnoi Uta Pa"(6) 520.220
        11/300: "Hummallo Aefroi Mu"(4) 570.085
...
```

Todas as linhas iniciadas por "#" são consideradas comentários e não contém dados estatísitos, apenas informações para o usuário. A cada iteração do algoritmo genético ele imprime uma linha '`Iteration <N>:`'(aonde '`<N>`' é o número da iteração), em seguida imprime uma linha para cada teste realizado com cada genoma, esta linha tem a seguinte sintaxe:

```
<N>/<TOTAL> "<NOME DA LINHAGEM> [SOBRENOME]"(<GERACAO>) <FITNESS>
```

Os Parâmetros são auto-explicativos. Ao final da geração o programa imprime a linha '`Bloodline lenght average: <N>`' informando o tamanho médio da linha de descendência(*bloodline*) e '`Fitness average: <N>`', o valor médio dos *fitness*.

Todos estes dados podem alimentar o programa '`log-plot.py`' que gera gráficos baseado nos dados. O '`log-plot.py`' pode gerar três tipos de gráfico: média de *fitness*, *fitness* individuais e histograma de *fitness*. É possível filtrar os dados para mostrar gráficos de linhagens específicas, a média geral(todos) e todos exceto uma linhagem específica. Tudo isso é especificado via linha de comando:

-v ou –verbose
: Escreve algumas informações na saída padrão sobre o processo de geração do gráfico.

-l <lista> ou –lineages=<lista>

Especifica uma lista de linhagens separadas por vírgula para criar o gráfico. Podem ser nomes de linhagens, '`all`', para a média geral ou então '`-<linhagem>`'(nome da linhagem precedido por um sinal de menos) que processa os dados de todos com exceção desta linhagem. As linhagens adicionalmente podem ser combinadas com '`"+"`', como em '`<linhagem1>+<linhagem2>`' aonde um único grupo de dados será processado para as duas linhagens.

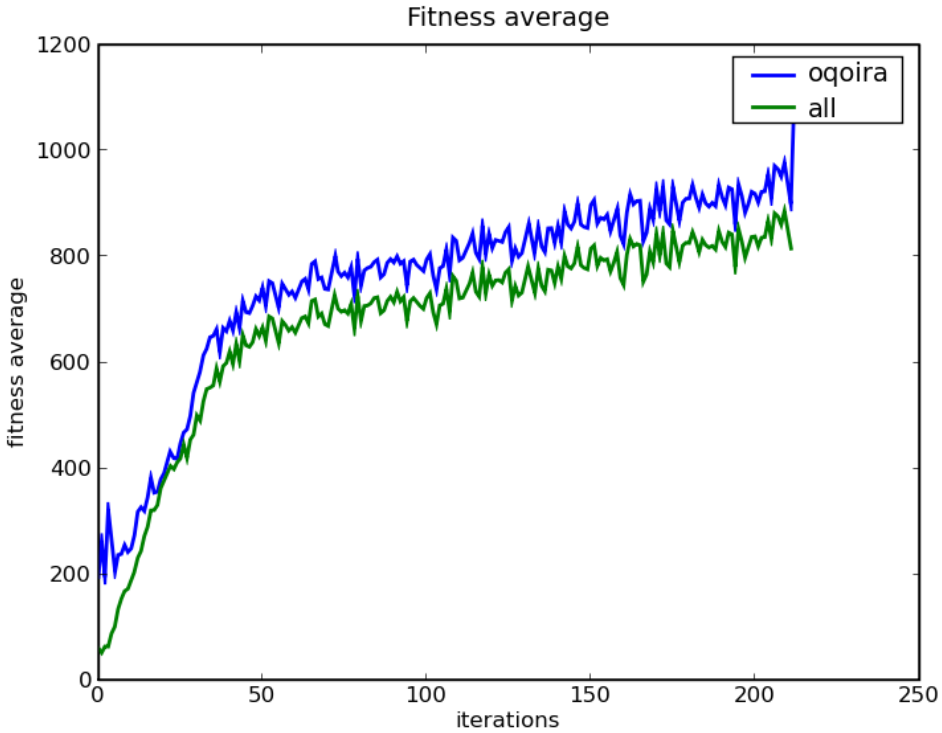
-g <tipo> ou –graph=<tipo>

Especifica o tipo de gráfico, podendo ter três valores: '`averages`'(média de *fitness*), '`individual`'(*fitness* individuais) ou '`histogram`'(histograma de *fitness*).

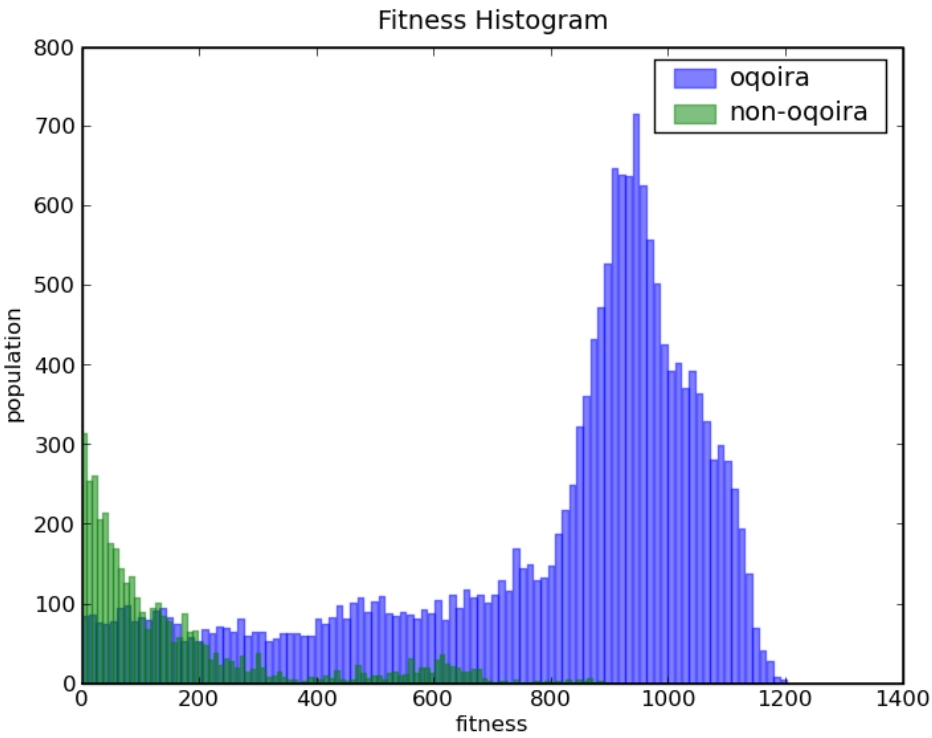
Se os gráficos gerados possuem mais de um objeto de dados é gerado uma legenda automaticamente.

2.4.4.1 Exemplos Gráficos

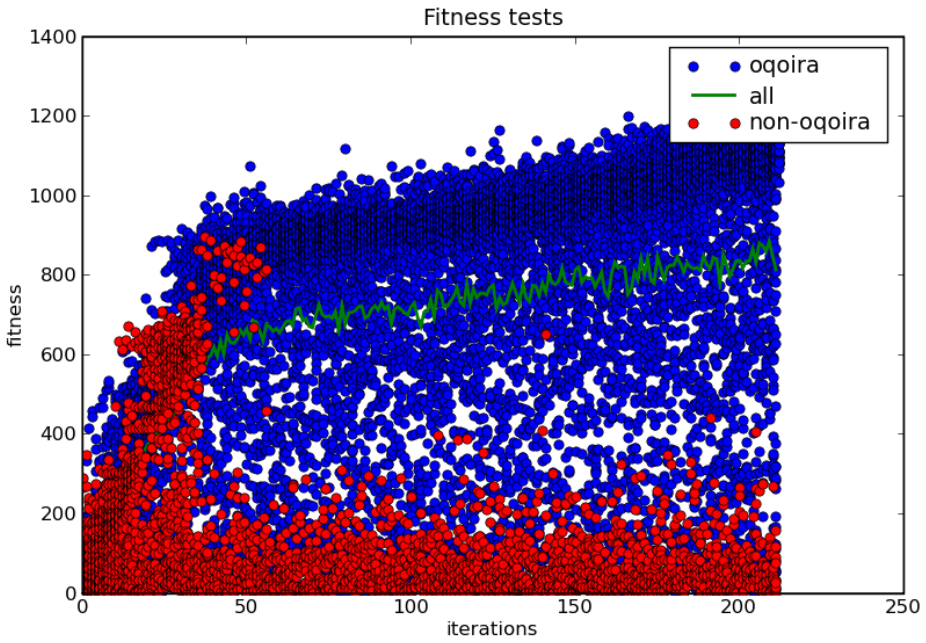
Todos os exemplos gráficos a seguir usam o mesmo conjunto de dados de um experimento, salvo no arquivo '`abc-walk-p100.log`'. Neste experimento uma população de 100 springbots evoluiu para andar durante 213 iterações. Ocorreu que a linhagem "Oqoria" se tornou dominante a partir da iteração 50, mesmo com infusões randômicas após a extinção das linhagens competidoras, nenhuma outra consegui se estabelecer. Estes gráficos mostram alguns pontos de vista.



```
$ ./log-plot.py -g av -l all,oqoira < abc-walk-p100.log
Média de fitness geral comparada com a
média da linhagem Oqoira: A média geral acompanha
o crescimento da linhagem dominante mas se mantém
sempre abaixo, devido às infusões randômicas.
```




```
$ ./log-plot.py -g hist -l oqoira,-oqoira < abc-walk-p100.log
Histograma de fitness comparando a linhagem Oqoira
com todas as não-Oqoiras: A linhagem dominante, também de
população dominante, está concentrada em torno do fitness
900, existe uma distribuição uniforme nesse meio devido às
mutações deletérias. O  restante da população não-Oqoira
concentra-se principalmente em fitness zero(infusões
randômicas) e um pouco em torno de 600(linhagens que competiram
mas foram extintas).
```



`$ ./log-plot.py -g ind -l all,oqoira,-oqoira < abc-walk-p100.log`
Fitness individual comparando a média geral, a linhagem Oqoira e os não-Oqoira: Observa-se a concentração da linhagem principal em uma linha evolutiva logaritmóide, os não-Oqoira evolutiram até se extinguirem em torno da iteração 50, o restante, são infusões randômicas concentrados no *fitness* zero, nota-se que é muito improvável que um espécime aleatório venha a competir com a linhagem dominante, que já se encontra bem evoluída após monopolizar o experimento.