



Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Nome do Aluno

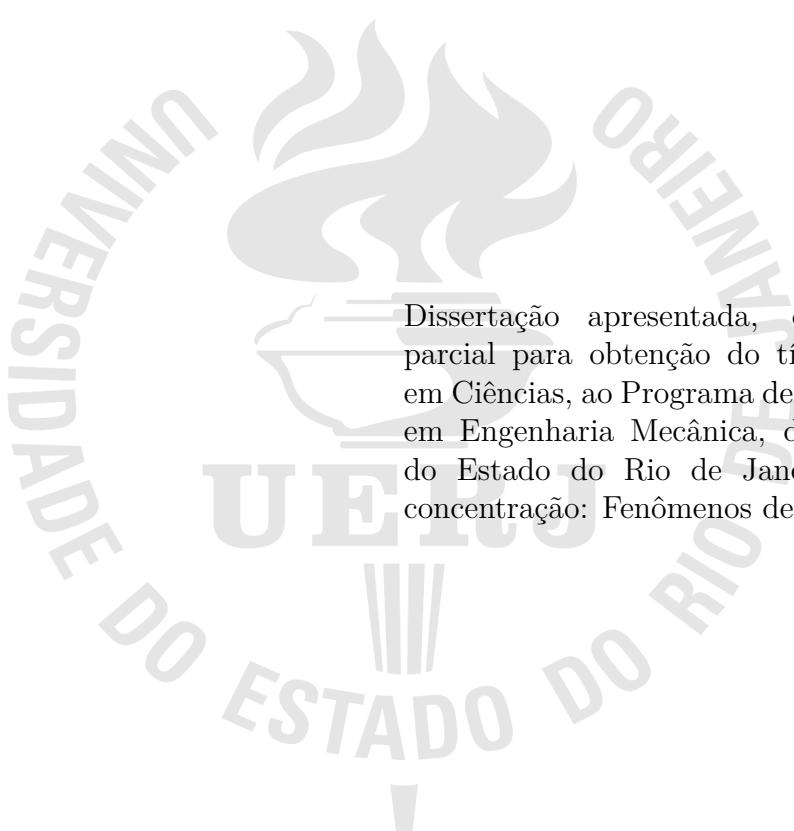
Título do Trabalho

Rio de Janeiro

2012

Nome do Aluno

Título do Trabalho



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Orientador: Prof. Dr. Nome do Professor

Rio de Janeiro

2012

CATALOGAÇÃO NA FONTE

S237

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

Sobrenome, Nome do Autor

Título do trabalho / Nome completo do autor. – 2012.
105 f.

Orientadores: Nome do orientador1;
Nome do orientador1.

Dissertação(Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

Texto a ser informado pela biblioteca

CDU 621:528.8

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial desta dissertação, desde que citada a fonte.

Assinatura

Data

Nome do Aluno

Título do Trabalho

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Aprovado em: 29 de Maio de 2012

Banca Examinadora:

Prof. Dr. Nome do Professor 1 (Orientador)
Instituto de Matemática e Estatística da UERJ

Prof. Dr. Nome do Professor 2
Faculdade de Engenharia da UERJ

Prof. Dr. Nome do Professor 3
Universidade Federal do Rio de Janeiro - UFRJ - COPPE

Prof. Dr. Nome do Professor 4
Instituto de Geociências da UFF

Prof. Dr. Nome do Professor 5
Universidade Federal do Rio de Janeiro - UFRJ - COPPE

Rio de Janeiro

2012

DEDICATÓRIA

Aqui entra sua dedicatória.

AGRADECIMENTO

Aqui entra seu agradecimento.

É importante sempre lembrar do agradecimento à instituição que financiou sua bolsa, se for o caso...

Agradeço à FAPERJ pela bolsa de Mestrado concedida.

RESUMO

SOBRENOME, Nome *Título do Trabalho*. 105 f. Dissertação (Mestrado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2012.

Aqui entra o seu resumo organizado em um parágrafo apenas.

Palavras-chave: Palavra1, Palavra2, Palavra3, Palavra 4.

ABSTRACT

Aqui entra seu resumo em inglês também organizado em apenas um parágrafo.

Keywords: Word1, Word2, Word3, Word4.

Listas de Figuras

FIGURA 1 – Um exemplo de indivíduo representado como uma sequência de caracteres com tamanho fixo.	17
FIGURA 2 – Representação de um indivíduo (programa) na <i>programação genética</i> . As letras 'abcdef' podem representar operações matemáticas, constantes ou até tomadas de decisão.	17
FIGURA 3 – Pêndulo invertido.	18
FIGURA 4 – A biblioteca <i>Gym</i> fornece alguns problemas de controle clássico como o pêndulo invertido (esquerda) e o pêndulo <i>swingup</i> , cujo objetivo é mantê-lo verticalmente para cima.	18
FIGURA 5 – Ciclo evolucionário artificial aplicado a uma população de soluções.	20
FIGURA 6 – Um indivíduo aleatório representando uma lei de controle.	21
FIGURA 7 – Um exemplo de inicialização de uma população com três indivíduos (aleatórios), com as características descritas em (1).	22
FIGURA 8 – Representação 2D da busca pelo mínimo da função custo. A operação de mutação permite uma busca em larga escala (em vermelho), enquanto que, em azul, a operação de cruzamento converge para o mínimo local.	24
FIGURA 9 – Mutação aplicada à um ponto correspondente a uma variável terminal gerando um <i>ramo</i> .	25
FIGURA 10 – Mutação aplicada em um ponto correspondente a um <i>ramo</i> . A nova sub-árvore possui apenas uma <i>variável terminal</i> .	25
FIGURA 11 – Exemplo de cruzamento atuando em dois indivíduos selecionados.	26
FIGURA 12 – Operação de <i>replicação</i> aplicada em um indivíduo selecionado.	26
FIGURA 13 – Ciclo evolucionário para programação genética.	28
FIGURA 14 – Sistema do pêndulo invertido (<i>Cart Pole</i>).	29
FIGURA 15 – Processo de decisão de Markov.	31
FIGURA 16 – Diagrama de classes para o ambiente de simulação do pêndulo invertido, gerado com o utilitário <i>pyreverse</i> .	33
FIGURA 17 – Forma geral de um algoritmo para realização de uma simulação no <i>OpenAi Gym</i> .	35

FIGURA 18 – <i>Box</i> é uma classe que permite a criação de conjuntos multidimensionais, no ambiente do pêndulo invertido é apenas uma lista (unidimensional).	36
FIGURA 19 – Recompensa recebida pelo agente.	36
FIGURA 20 – Valor lógico que indica se o episódio terminou ou não.	36
FIGURA 21 – Módulos da biblioteca DEAP e suas funcionalidades.	38
FIGURA 22 – A saída depende do sinal da expressão que resulta da avaliação do indivíduo.	40
FIGURA 23 – As linhas pontilhadas indicam a maior distância entre a <i>raiz</i> e uma variável <i>terminal</i> . Este exemplo utiliza o conjunto primitivo \mathcal{P} definido anteriormente.	43
FIGURA 24 – Média da aptidão de todos os indivíduos por geração (verde). Valor máximo de aptidão observado em cada geração (vermelho). Menor aptidão observada em cada geração (azul).	58
FIGURA 25 – Número de indivíduos, em cada geração, que obtiveram cada faixa de aptidão. Verifica-se que já na 12º geração, a maior parte dos indivíduos possuem a aptidão máxima.	59
FIGURA 26 – Comprimento dos indivíduos por geração.	60
FIGURA 27 – As medidas relacionadas à complexidade dos indivíduos em cada geração. Essa estatística tem uma alta correlação com a aridez das operações que ocorrem nos indivíduos.	60
FIGURA 28 – Ocorrência de cada operador e variável nos indivíduos da última geração..	61
FIGURA 29 – Primeiro indivíduo do hall da fama na primeira execução.	62
FIGURA 30 – Vigésimo terceiro indivíduo do hall da fama, também na primeira execução do algoritmo.	63
FIGURA 31 – Controle do pêndulo pelo indivíduo da Figura 29. O pêndulo é levado rapidamente à um estado de baixa oscilação nos valores angulares.. .	64
FIGURA 32 – Recompensa média acumulada em função do número de passos de simulação. A linha azul representa a média móvel.	65
FIGURA 33 – Recompensa média ao longo dos passos de simulação e a média móvel. .	65
FIGURA 34 – Posição do carrinho (topo) e ângulo do bastão em função do tempo. .	66

FIGURA 61 – Complexidade média dos indivíduos da população ao longo das gerações.	89
FIGURA 62 – Ocorrências dos operadores e variáveis terminais na última geração.	90
FIGURA 63 – Melhor indivíduo da primeira execução.	91
FIGURA 64 – Atuação do indivíduo da Figura 63 em um episódio aleatório.	92
FIGURA 65 – Agente DQN e a recompensa acumulada ao longo dos passos de tempo.	92

Lista de Tabelas

TABELA 1 – Variáveis de estado fornecidas pela biblioteca.	32
TABELA 2 – Variáveis terminais.	39
TABELA 3 – Conjunto de operadores lógicos e matemáticos.	39
TABELA 4 – Parâmetros utilizados para o problema do pêndulo invertido.	57
TABELA 5 – Comparaçāo entre a programação genética e DQN para o pêndulo invertido.	66
TABELA 6 – Variáveis de estado para o pêndulo swing-up.	68
TABELA 7 – Parâmetros da programação genética aplicada ao pêndulo swing-up. .	69
TABELA 8 – Comparaçāo entre a programação genética e DDPG para o pêndulo swing-up.	77
TABELA 9 – Variáveis de estado para o pêndulo duplo invertido.	79
TABELA 10 – Comparaçāo entre PG e DDPG para o pêndulo duplo invertido. . . .	85
TABELA 11 – Variáveis de estado para o problema do carro na ladeira.	86
TABELA 12 – Parâmetros utilizados para o problema do carro na ladeira.	87
TABELA 13 – Comparaçāo entre PG e DQN para o problema do carro na ladeira. .	92
TABELA 14 – Condições iniciais do ambiente <i>CartPole-v1</i>	99
TABELA 15 – Condições iniciais do ambiente <i>Pendulum-v0</i>	99
TABELA 16 – Condições iniciais do ambiente <i>Pendulum-v0</i>	100

Sumário

<u>INTRODUÇÃO</u>	13
1 A PROGRAMAÇÃO GENÉTICA	20
2 OPENAI GYM: PÊNDULO INVERTIDO	29
2.1 SISTEMA DINÂMICO	29
2.2 OPENAI GYM	30
3 DEAP: IMPLEMENTAÇÃO DOS AGENTES	37
3.1 REPRESENTAÇÃO E INICIALIZAÇÃO	38
3.1.1 Representação	38
3.1.2 Aptidão	41
3.1.3 Inicialização	42
3.1.4 População	44
3.2 AVALIAÇÃO DE INDIVÍDUOS	46
3.3 SELEÇÃO DE INDIVÍDUOS	49
3.4 OPERADORES GENÉTICOS	50
3.4.1 Replicação	50
3.4.2 Mutação	50
3.4.3 Cruzamento	51
3.4.4 Seleção Aleatória do Operador	52
3.5 CICLO ITERATIVO	53
4 DEAP: ESTUDOS DE CASO	54
4.1 Pêndulo Invertido	56
4.2 Pêndulo <i>Swing-up</i>	66
4.3 Pêndulo Duplo Invertido	78
4.4 Carro na Ladeira	85
<u>CONCLUSÃO</u>	94
REFERÊNCIAS	97

APÊNDICES	99
A Códigos	99
B Condições Iniciais	99
B.1 Pêndulo Invertido	99
B.2 Pêndulo Swing-up	99
B.3 Carro na Ladeira	100

INTRODUÇÃO

Há anos criamos máquinas que podem ser consideradas inteligentes, dada a complexidade de suas tarefas. Entretanto, uma atenção especial vem sendo direcionada à máquinas que possuem a capacidade de aprender, ou evoluir com as experiências, isto é, não necessariamente precisamos ensina-la como realizar essas tarefas complexas. Arthur Daniel sumarizou essa ideia em uma frase atribuída a ele [1]:

”Como computadores podem aprender a resolver problemas sem serem explicitamente programados? Em outras palavras, como computadores podem ser feitos de modo que façam uma tarefa sem que digamos exatamente como?”

Esse campo, conhecido nos dias de hoje por *aprendizado de máquinas*, ou *Machine Learning*, se baseia principalmente na ideia mencionada por Arthur Daniel. Nesse paradigma, a tarefa do humano é criar uma máquina complexa o suficiente que a permita o aprendizado por experiência.

Nos últimos anos, foram criados máquinas, ou agentes, que aprendem e por consequências alguns feitos marcantes foram atingidos. *AlphaGo Zero*, uma máquina que joga xadrez, é capaz de atingir performances melhores que qualquer humano ou máquina explicitamente programada. Quando criado *AlphaGo Zero* não possuía informações básicas sobre o funcionamento do jogo mas foi capaz de melhorar sua performance com a experiência.

A realização dessas máquinas só foi possível graças aos avanços na capacidade de processamento de dados dos computadores e a quantidade de dados disponíveis. Por consequência, mesmo que a taxa de aprendizado da máquina seja baixa, conseguimos expor o agente a um alto numero de experiencias.

Os algoritmos de aprendizado de máquinas podem ser vistos como uma busca, dentro de um enorme espaço de possíveis soluções candidatas. De certa forma, essa busca é guiada pelas experiências passadas de forma a maximizar uma medida de performance [2].

Em geral, podemos dividir os algoritmos de aprendizado de máquinas em quatro categorias [3]:

- a) **Aprendizado Supervisionado:** Um conjunto de exemplos de treinamento (com a resposta correta) é fornecido ao algoritmo, baseado nesses exemplos as respostas são generalizadas para qualquer entrada possível. Em outras palavras, fornecemos

os dados ao algoritmo e dizemos o que esperamos de resposta, cabe ao programa aprender com a experiência fornecida e deduzir corretamente a resposta para outras entradas.

- b) **Aprendizado Não-Supervisionado:** Não é fornecida a resposta correta nos exemplos de treinamento, cabe ao algoritmo categorizar as entradas de acordo com suas semelhanças.
- c) **Aprendizado por Reforço:** O algoritmo recebe a informação de que a resposta está incorreta, mas não é dito como consertá-la. A máquina busca no espaço de possíveis soluções até que encontre a solução correta.
- d) **Aprendizagem Evolucionária:** Baseia-se na visão de que a evolução biológica é um processo de aprendizado, já que os organismos devem se adaptar ao ambiente para aumentar suas chances de sobrevivência e reprodução. São empregadas funções de medição de performance (*fitness*) para avaliar a solução.

A principal diferença das duas últimas abordagens em relação às primeiras é o papel da máquina no ambiente em que ocorre o aprendizado: geralmente, para os algoritmos de aprendizagem supervisionada ou não-supervisionada, o agente não influencia o sistema, isto é, suas ações no decorrer do aprendizado (ou treinamento) não afetam a dinâmica do ambiente.

Por exemplo, algoritmos de aprendizado supervisionado podem ser utilizados para previsões no mercado de ações, e algoritmos de aprendizado não-supervisionado podem ser utilizados para agrupar clientes de acordo com suas preferências de compras. A aprendizagem por reforço possui aplicações em diversas áreas, incluindo robótica e controle. Muitos problemas em que seja natural o uso de aprendizagem por reforço podem ser abordados também por algoritmos evolucionários que utilizam programação genética.

Vimos que os algoritmos de aprendizado podem ser vistos como uma busca, em um espaço enorme de possibilidades, por máquinas capazes de resolver problemas com um determinado grau de acurácia. Já que o espaço de busca por soluções é vasto, precisamos realizar a procura de forma adaptativa ou inteligente [1].

Os algoritmos evolucionários buscam soluções melhores através de um processo iterativo inteligente em que:

- a) Mede-se a performance (*fitness*) de diversos indivíduos inicializados com características aleatórias.
- b) Os agentes que apresentarem boa performance são selecionados e têm suas características modificadas, gerando novos indivíduos com aspectos semelhantes e possivelmente mais aptos à resolução do problema.
- c) A nova população de indivíduos é avaliada e os agentes que apresentam melhor performance são novamente selecionados e modificados.
- d) O processo se repete até um critério de término pré-estabelecido.

Verificamos a semelhança da escolha de soluções em virtude de sua performance com a seleção natural biológica, isto é, indivíduos mais aptos sobrevivem e reproduzem. As modificações das soluções selecionadas são inspiradas nos fenômenos biológicos de mutação e cruzamento genético dos genitores. A essas modificações damos o nome de operações genéticas. Por consequência, a mutação e o cruzamento genético (*também conhecido como crossover*) são chamados de *operadores genéticos*. Seguindo a mesma lógica, o programa que utiliza a metodologia descrita acima também é chamado de *algoritmo genético*.

Vale ressaltar que as características dos indivíduos podem ser representadas de diversas formas. Além disso, as operações genéticas podem ser aplicadas de diversas maneiras, de acordo com a representação de indivíduo escolhida.

A *programação genética* se refere a uma categoria de algoritmo genético introduzida principalmente por John R. Koza [1] em que *programas de computador* são evoluídos. Nesse sentido, diversos problemas podem ser reformulados como uma demanda por um programa de computador que produza uma saída desejada quando uma determinada entrada seja apresentada.

A principal distinção do método introduzido por Koza se refere ao modo de **representação** dos indivíduos. Grande parte dos trabalhos realizados até a década de 90 na área de algoritmos genéticos utilizavam representações de tamanho fixo com uma sequência de caracteres.

Para muitos problemas a representação natural de uma solução é um programa hierárquico de tamanho variável ao invés de uma representação de tamanho fixo. Isso ocorre porque frequentemente não sabemos de antemão os tamanhos e dimensões da solução procurada.

a	b	c	d	e	f
---	---	---	---	---	---

Figura 1: Um exemplo de indivíduo representado como uma sequência de caracteres com tamanho fixo.

Programas de computador hierárquicos, particularmente escritos em linguagens funcionais e/ou recursivas, podem ser facilmente representados por árvores. Por essa razão, a linguagem LISP é uma das escolha naturais para abordar a programação genética [4].

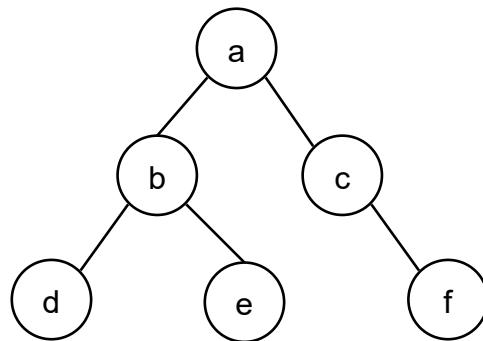


Figura 2: Representação de um indivíduo (programa) na *programação genética*. As letras 'abcdef' podem representar operações matemáticas, constantes ou até tomadas de decisão.

A estrutura da ?? representa um programa de computador. De fato, podemos escrever funções em LISP utilizando uma representação recursiva semelhante. Programas de computador são entidades capazes de resolver inúmeros problemas em diversos campos, por esse motivo, a programação genética pode ser utilizada em uma gama variada de problemas.

Buscaremos aplicar a *programação genética* (PG) em problemas clássicos de controle. O sistema do pêndulo invertido da ?? é um dos problemas mais importantes na teoria de controle, principalmente pela sua natureza não-linear e instável, portanto, servirá como estudo de caso básico para este trabalho.

O objetivo consiste em balancear o pêndulo utilizando apenas F como variável de controle, que pode ser aplicada na direção positiva ou negativa do eixo x .

Utilizaremos como ambiente de teste um conjunto de ferramentas na linguagem *Python* chamado de *Gym* [5]. O principal propósito dessa biblioteca é criar um ambiente padronizado para testar algoritmos de aprendizagem por reforço, veremos como utilizá-la em um problema a ser solucionado por programação genética.

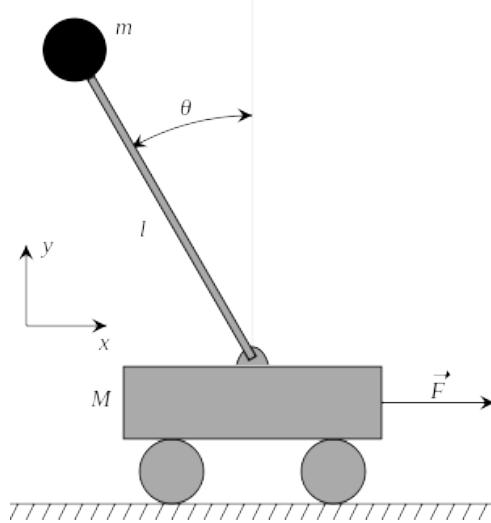


Figura 3: Pêndulo invertido.

Fonte: https://en.wikipedia.org/wiki/Inverted_pendulum

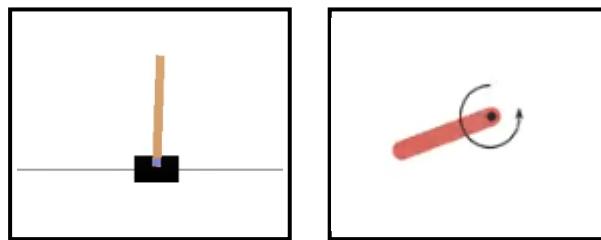


Figura 4: A biblioteca *Gym* fornece alguns problemas de controle clássico como o pêndulo invertido (esquerda) e o pêndulo *swingup*, cujo objetivo é mantê-lo verticalmente para cima.

Fonte: [5]

A biblioteca DEAP (*Distributed Evolutionary Algorithms in Python*) é um framework de computação evolucionária designada para implementação eficiente de algoritmos evolucionários [6]. Nossa objetivo é implementar um algoritmo de programação genética utilizando a biblioteca DEAP e evoluir a solução utilizando o ambiente proporcionado pela biblioteca Gym.

Buscamos nesse trabalho prover uma abordagem didática para a implementação da *programação genética* utilizando ferramentas atuais de teste. Por ser direcionado principalmente a algoritmos de *aprendizagem por reforço*, existem poucas implementações de PG nos ambientes de aprendizado providos pela biblioteca Gym.

Inicialmente, iremos realizar uma análise mais detalhada do ciclo evolutivo artificial no contexto da programação genética. Além disso, iremos verificar com mais detalhes a representação da ?? e seus componentes. Então, será feito estudo do sistema pêndulo

invertido e como poderemos incluir na representação informações importantes para a resolução do sistema.

Com o modo de representação dos indivíduos definido, podemos pensar em como inicializá-los. O conjunto de ferramentas DEAP proporciona diversos métodos que iram simplificar esse processo. O próximo problema a ser abordado é a medida de desempenho (*fitness*), isto é, como iremos medir a aptidão de um indivíduo ou solução.

Com a medida de performance para cada indivíduo, podemos implementar um método para selecionar indivíduos. Serão discutidas as possíveis abordagens para essa etapa. Finalmente, iremos implementar cada uma das operações genéticas a serem aplicadas nos indivíduos selecionados.

Todas as etapas serão realizadas na linguagem Python utilizando as bibliotecas Gym, DEAP e outras para recursos adicionais, como por exemplo, produção de gráficos.

1 A PROGRAMAÇÃO GENÉTICA

Um breve sumário dos conceitos da programação genética foram incluídos na introdução. Entretanto, precisamos aprofundar um pouco mais os conceitos abordados, de modo que possamos aplicá-la com eficiência.

Nesse capítulo lidaremos com aspectos mais específicos da representação da 2, isto é, o que exatamente são os caracteres *abcdef* em cada problema. Uma visão geral do ciclo evolutivo artificial será incluída e servirá de base para os próximos capítulos, já que o queremos implementar de fato é o ciclo completo que permitirá a evolução de uma solução apropriada para o problema.

Um algoritmo evolucionário em geral pode ser representado graficamente de acordo com a 5.

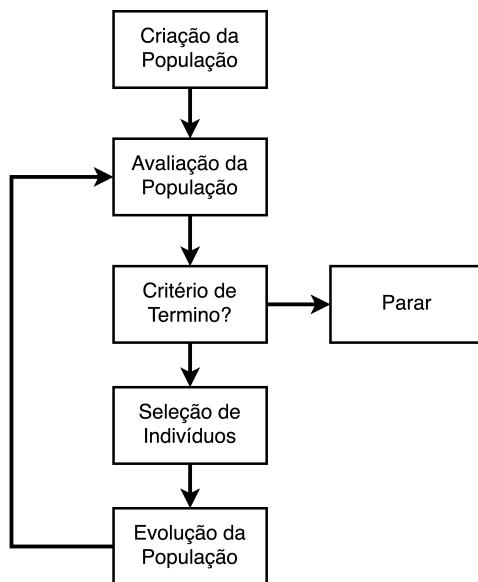


Figura 5: Ciclo evolucionário artificial aplicado a uma população de soluções.

O primeiro passo é a criação da população de soluções de acordo com a representação escolhida. A nossa abordagem utiliza o conceito proposto por Koza [1], geralmente cada indivíduo será representado como uma árvore que representa em sua essência um programa de computador.

Será abordado um problema relacionado ao controle de um sistema (ou planta), em suma, o objetivo é encontrar uma lei de controle. Por exemplo, um indivíduo aleatório inicializado poderia ter as características da 6. A lei de controle seria dada pela expressão matemática que resulta da avaliação recursiva da árvore.

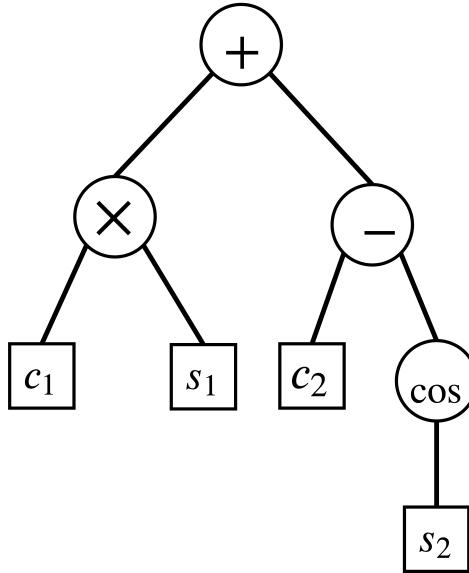


Figura 6: Um indivíduo aleatório representando uma lei de controle.

Na 6 representamos **operadores** através de círculos e **variáveis terminais** com quadrados. Como estamos utilizando uma representação de árvore podemos chamar de **raiz** o operador no topo da árvore, **ramo**, os operadores subsequentes à raiz e **folhas** as variáveis terminais. Na 6

Os operadores podem ser matemáticos, lógicos, entre outros. Quanto à aridade pode-se utilizar operadores que atuam em uma, duas ou mais variáveis. As variáveis terminais podem ser constantes aleatórias como 1, 0 ou até π . Em problemas de controle é comum admitir como variáveis terminais uma leitura de sensor, por exemplo.

Em LISP, esse indivíduo poderia ser representado da seguinte forma:

$$(+(*(s_1)(c_1))(-(s_1)(\cos(c_2))))$$

Para um problema em que cada indivíduo representa um agente (como um robô), podemos ter como variáveis terminais *ESQ* ou *DIR* representando as ações de movimento para a esquerda ou direita, respectivamente.

Geralmente, a representação em forma de árvore é mais intuitiva. Porém do ponto de vista computacional, costuma ser mais eficiente trabalhar com *strings* representando uma lista.

É possível estipular, por exemplo, um tamanho mínimo e máximo de cada árvore, eliminando assim soluções muito simples em que sabemos não servir como solução para o problema, ou soluções demasiadamente complexas.

Existem diversas variações da estrutura em forma de árvore. Na programação genética cartesiana [7], por exemplo, cada variável de entrada pode servir de operando para mais de um operador. Na programação genética linear [8], um indivíduo é um programa de computador com instruções que recebem um ou dois operandos.

Após a escolha da representação desejada, a inicialização de cada indivíduo é feita de forma aleatória, respeitando um dado conjunto de operadores \mathcal{O} , variáveis terminais \mathcal{V} e nível (ou número de camadas) $\mathcal{D} \in (D_{min}, D_{max})$.

Por exemplo, se definirmos:

$$\begin{aligned}\mathcal{O} &= \{+, -, *\} \\ \mathcal{V} &= \{1, 0, 3\} \\ \mathcal{D} &= \{2, 3\}\end{aligned}\tag{1}$$

Indivíduos como os da Figura 7 seriam possíveis elementos dessa população.

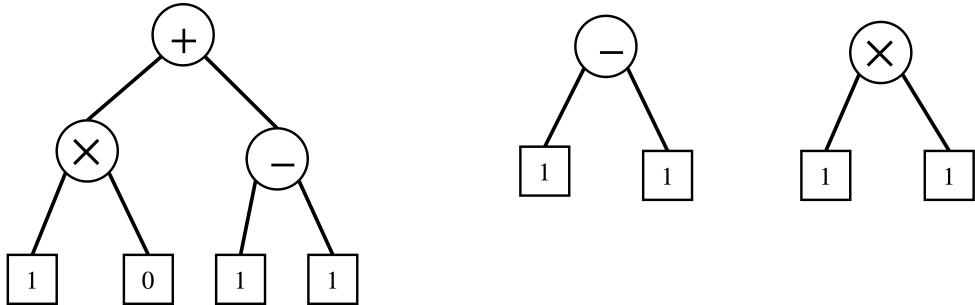


Figura 7: Um exemplo de inicialização de uma população com três indivíduos (aleatórios), com as características descritas em (1).

Após a inicialização da população é feita a avaliação de cada indivíduo. O objetivo dessa etapa é verificar a aptidão de todos os indivíduos. Essa medida de performance do indivíduo deve refletir o que buscamos em termos de soluções efetivas para o problema, ou seja, a função que determina a medida de performance da solução é específica a cada problema.

Em problemas de controle, geralmente o objetivo é manter a posição de um objeto o mais próximo possível de um valor de referência. Logo, a função de avaliação pode ser representada como uma função custo \mathbf{J} . O objetivo do algoritmo é, portanto, achar uma lei de controle que minimiza \mathbf{J} .

Uma função custo pode ser implementada de modo que penalize desvios de um estado de referência.

$$J = \frac{1}{T} \int_0^T J_t dt = \frac{1}{T} \int_0^T (X_{ref} - X_{med})^2 dt \quad (2)$$

Onde X_{med} é uma variável de saída medida e X_{ref} é o valor de referência dessa mesma variável.

Com uma medida de aptidão associada a cada indivíduo, é possível aplicar um método de seleção na população, ou seja, os indivíduos mais aptos serão selecionados. Um método básico e eficaz de seleção de soluções é comparar dois indivíduos aleatórios na população (utilizando a avaliação realizada anteriormente) e selecionar o mais apto.

Esse método conhecido como *campeonato* pode ser feito em dois ou mais níveis, isto é, escolhemos quatro ou mais indivíduos aleatórios na população, realizando comparações dois a dois, de modo que apenas um indivíduo (o mais apto) seja selecionado.

É aplicado no indivíduo selecionado um dos operadores genéticos e a nova solução gerada é inserida em uma nova população. O processo de seleção, aplicação de operadores genéticos e eventual inserção do indivíduo se repete até que a nova população possua a mesma quantidade de indivíduos da população antiga.

Para cada indivíduo selecionado, apenas uma operação genética é aplicada de acordo com uma probabilidade específica para cada operação. Podemos definir:

- P_c : probabilidade de cruzamento.
- P_m : probabilidade de mutação.
- P_r : probabilidade de replicação.

De modo que $P_c + P_m + P_r = 1$.

É interessante lembrar que o ciclo evolucionário é uma **busca** por soluções. Nesse contexto, são comuns na literatura os termos **exploration** e **exploitation**, que se referem a exploração, entretanto *exploration* se refere à uma busca em larga escala, enquanto que *exploitation* refere-se à uma busca local.

De acordo com [9], a busca pela função custo pode ser representada em gráficos semelhantes à 8, com a operação de cruzamento (*crossover*) sendo responsável pela busca

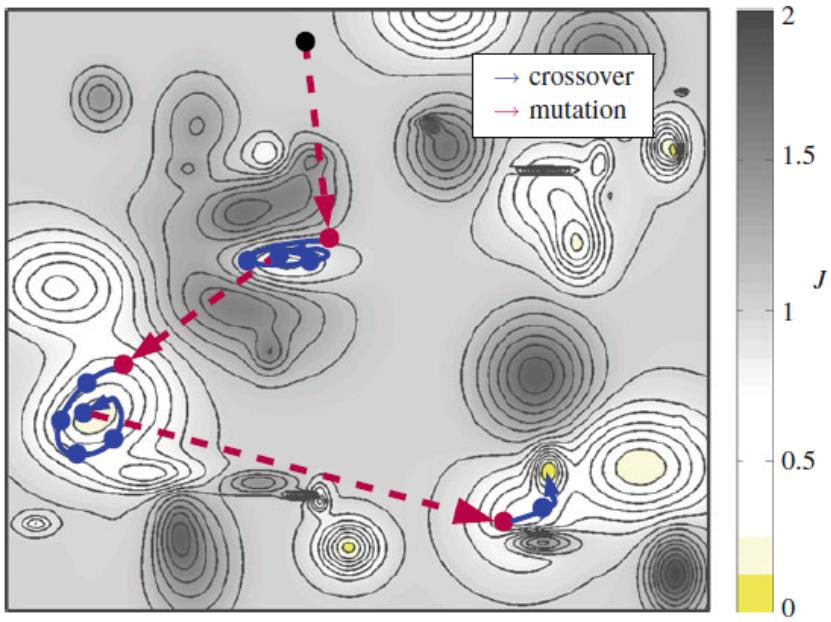


Figura 8: Representação 2D da busca pelo mínimo da função custo. A operação de mutação permite uma busca em larga escala (em vermelho), enquanto que, em azul, a operação de cruzamento converge para o mínimo local.

Fonte: [9]

local do mínimo (*exploitation*), enquanto que a operação de mutação realiza uma busca mais ampla (*exploration*).

A operação de cruzamento é realizada em dois indivíduos selecionados, isto é, aptos de acordo com o critério de desempenho escolhido. É razoável, portanto, que seja responsável pela convergência local, pois os novos indivíduo gerados possuem toda sua estrutura formada por partes de soluções consideradas boas. Já no caso da mutação o novo indivíduo possuirá uma parcela de sua estrutura sendo gerada de forma aleatória, o que não garante convergência, porém é útil para explorar o espaço de possíveis soluções.

A operação genética de mutação na PG começar ao selecionar um ponto aleatório na árvore, podendo este ser um ponto interno ou externo (operadores ou variáveis terminais). A mutação então remove esse ponto da árvore junto à qualquer sub-árvore proveniente desse ponto. Na localização do ponto removido uma nova sub-árvore é gerada aleatoriamente.

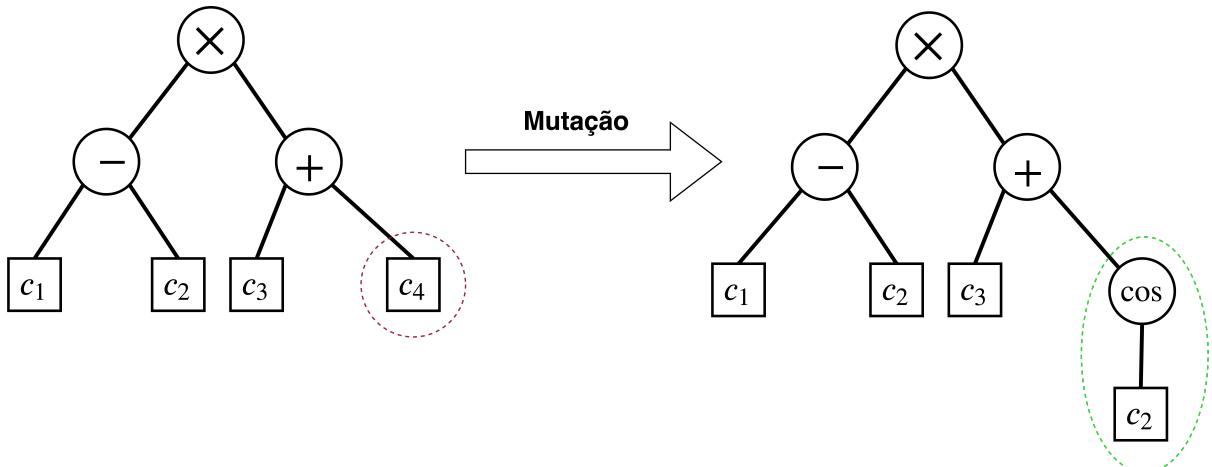


Figura 9: Mutação aplicada à um ponto correspondente a uma variável terminal gerando um *ramo*.

A nova sub-árvore é gerada aleatoriamente e pode conter um *ramo* ou possivelmente apenas uma variável terminal, como na 9.

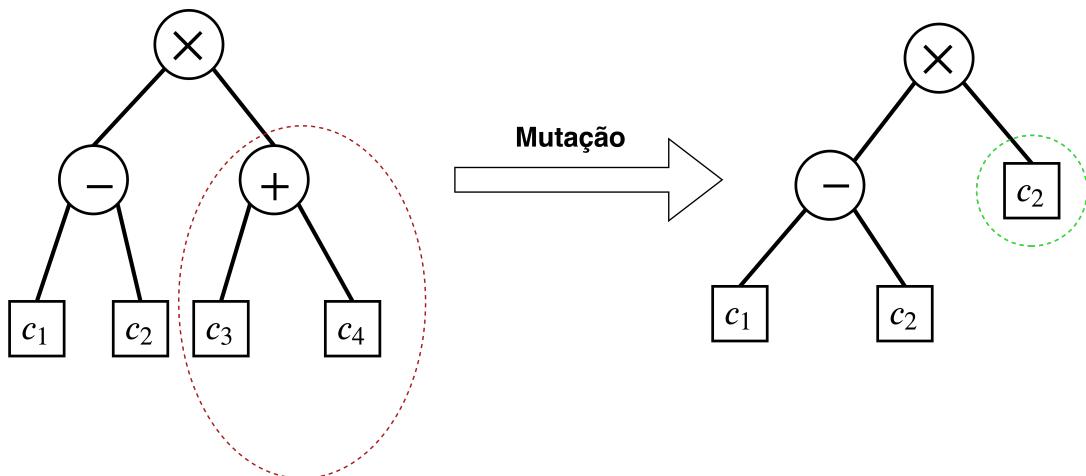


Figura 10: Mutação aplicada em um ponto correspondente a um *ramo*. A nova sub-árvore possui apenas uma *variável terminal*.

A operação de *cruzamento* atua a partir de dois indivíduos selecionados, novamente, soluções consideradas aptas pelo método de seleção. Assim como na operação de *mutação*, um ponto aleatório na árvore de cada indivíduo é selecionado e a sub-árvore é trocada entre as soluções.

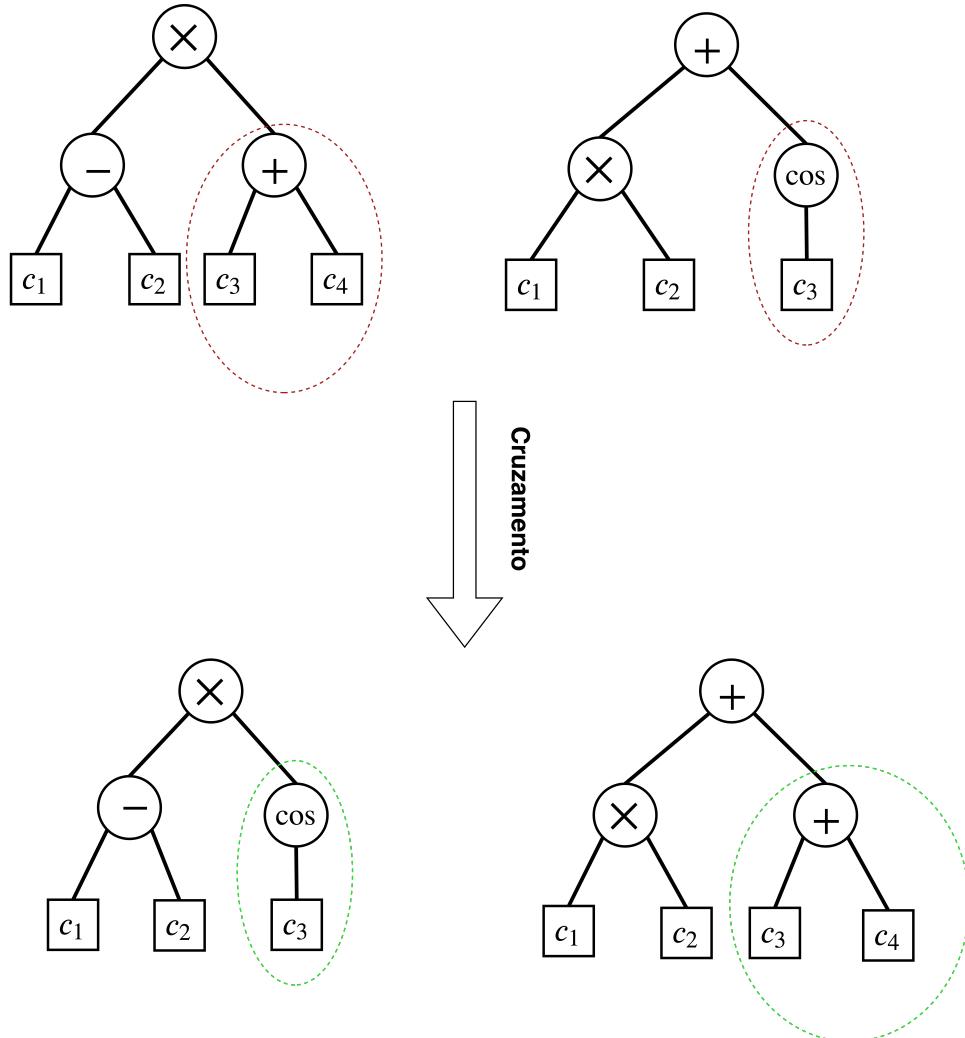


Figura 11: Exemplo de cruzamento atuando em dois indivíduos selecionados.

A *replicação* é um operador genético simples em que um indivíduo selecionado é copiado diretamente para a próxima geração, sem qualquer alteração em sua estrutura.

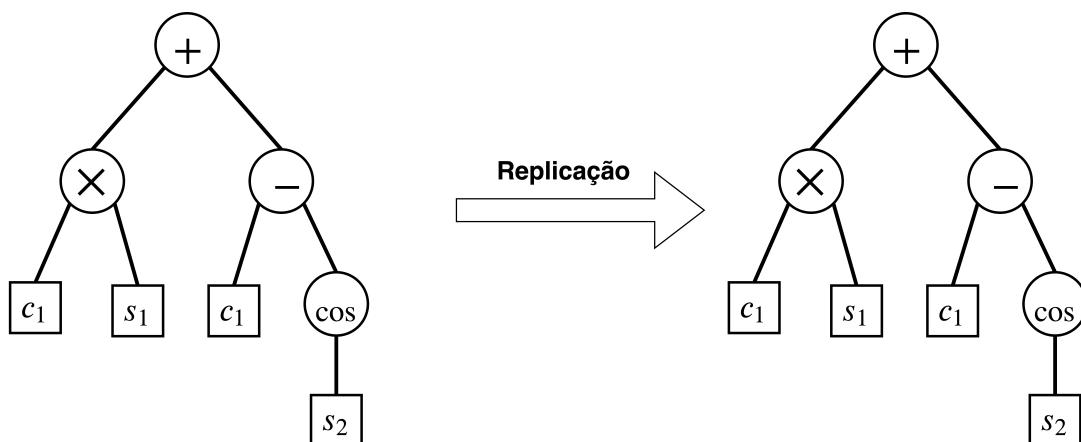


Figura 12: Operação de *replicação* aplicada em um indivíduo selecionado.

Koza [1] menciona outras operações genéticas como a *edição*, cuja função é editar

e simplificar as expressões das árvores, *permutação*, uma generalização do operador de inversão aplicado em algoritmos genéticos de representações fixas e *encapsulamento*, um método de identificação de ramos potencialmente úteis que pode ser referenciado posteriormente. Possivelmente, uma operação de *elitismo* pode ser empregada, que copia os melhores indivíduos diretamente para a próxima geração.

O critério de término do ciclo evolutivo geralmente empregado é o pre-estabelecimento de um número limite de gerações. Alguns problemas permitem a criação de um critério de desempenho, isto é, o ciclo evolutivo se repete até que um indivíduo obtenha um determinado valor de aptidão (medida pela função de avaliação).

Uma implementação de programação genética não se torna completa sem o estabelecimento de alguns parâmetros de controle, como por exemplo, número de indivíduos da população ou a probabilidade de cada operação genética. Uma lista dos principais parâmetros de controle para um algoritmo de PG é fornecida abaixo.

1. Em relação a população:

- a) Número de indivíduos da população: M
- b) Número máximo de gerações: G
- c) Método de inicialização
- d) Método de seleção de indivíduos

2. Em relação à representação do indivíduo:

- a) Conjunto de operações em ramos: \mathcal{O}
- b) Conjunto de variáveis terminais: \mathcal{V}
- c) Mínima profundidade inicial da árvore: D_{min}
- d) Máxima profundidade inicial da árvore: D_{max}

3. Em relação às operações genéticas:

- a) Probabilidade de cruzamento: P_c
- b) Probabilidade de mutação: P_m
- c) Probabilidade de replicação: P_r

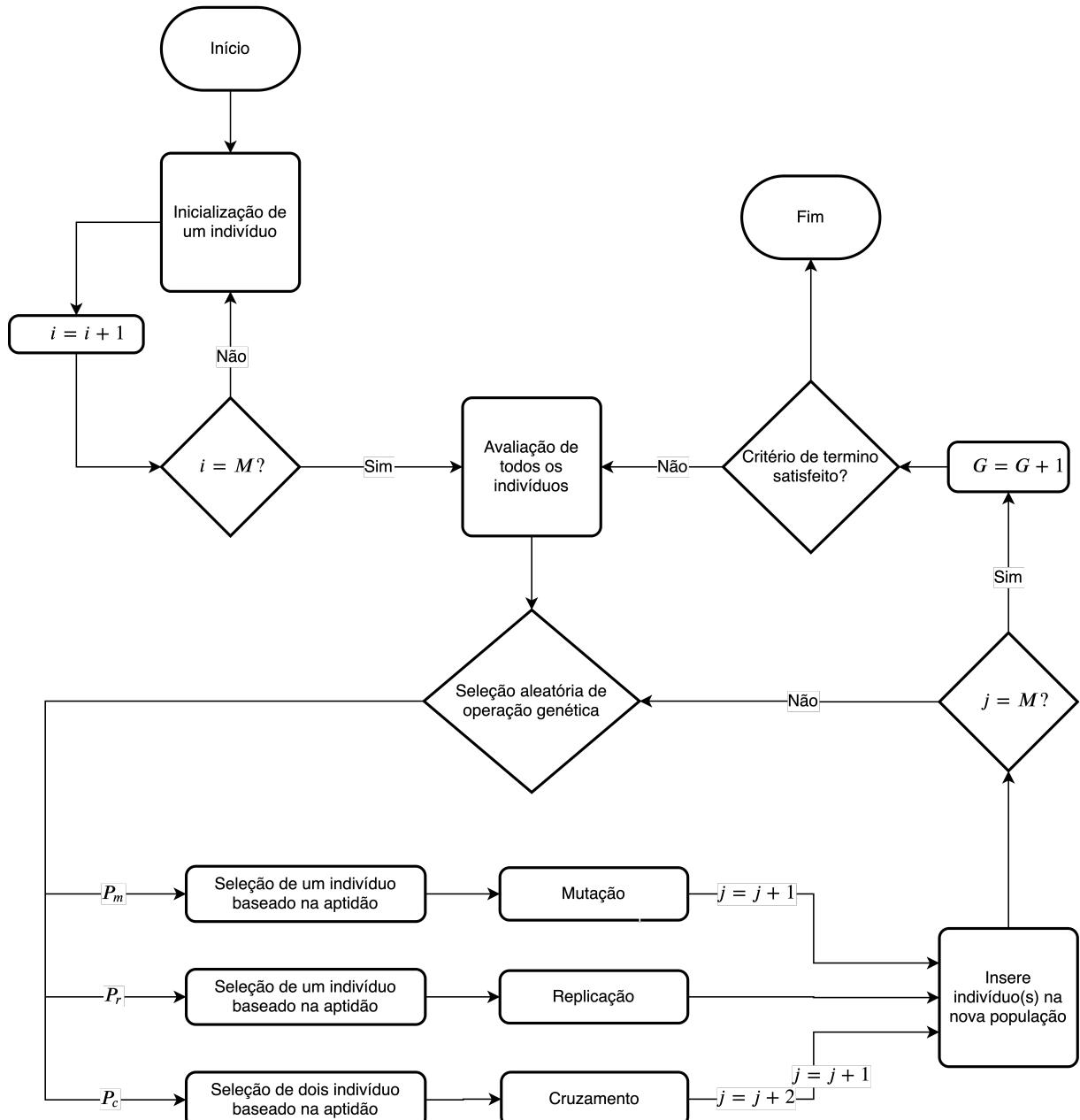


Figura 13: Ciclo evolucionário para programação genética.

A partir dos parâmetros estabelecidos e dos métodos esclarecidos neste capítulo, podemos reformular o ciclo da 5 para a inclusão de mais detalhes.

A partir do fluxograma da 13 vemos como implementar a programação genética à um problema genérico, resta-nos, portanto, particularizar o ciclo evolutivo para o problema do pêndulo invertido, o que nos obriga a verificar uma função de aptidão específica (*fitness*), entre outros detalhes.

2 OPENAI GYM: PÊNDULO INVERTIDO

Foi mencionada a importância de analisar o problema em que desejamos aplicar a programação genética. Essa análise permitirá a escolha da função de avaliação, das variáveis terminais e dos operadores matemáticos ou lógicos. Além disso, teremos como objetivo verificar a implementação do sistema na biblioteca que será utilizada, *OpenAI Gym*.

2.1 SISTEMA DINÂMICO

O pêndulo invertido pode ser considerado um dos sistemas robóticos mais simples, composto apenas por um corpo rígido e um ponto de rotação. Apesar da sua simplicidade, muitas técnicas padrões derivadas da teoria de controle são ineficientes, por isso, o sistema é considerado um dos teste mais importantes na área de controle não-linear [10].

Existem diversas variações desse sistema [11], lidaremos com o caso de um bastão apoiado sobre um carro que se movimenta com apenas um grau de liberdade e sua posição é dada pela variável s . O ângulo θ representa a inclinação do bastão em relação à uma reta normal à superfície. O sistema deve ser controlado por uma força F de magnitude fixa aplicada paralelamente à superfície em qualquer sentido.

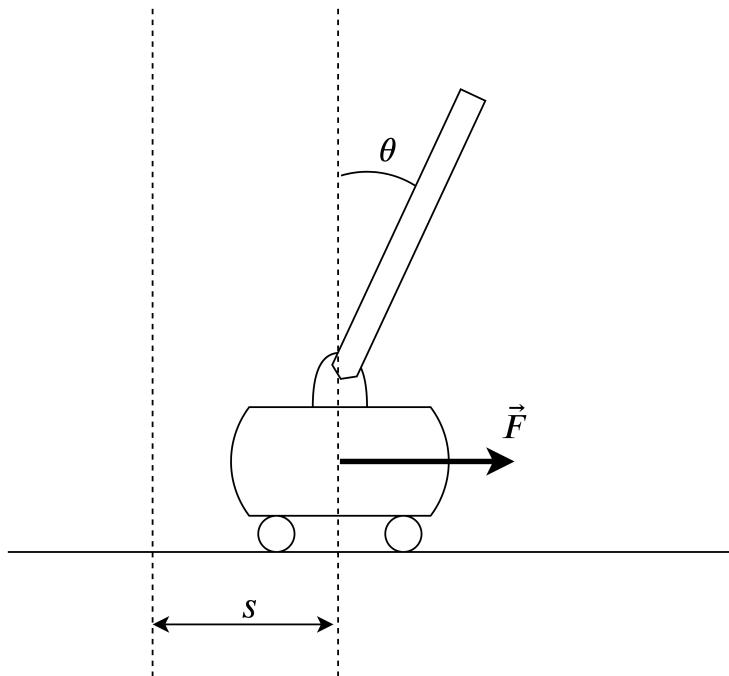


Figura 14: Sistema do pêndulo invertido (*Cart Pole*).

A dinâmica do sistema é modelada pelas equações diferenciais não-lineares a seguir [12]:

$$\ddot{\theta} = \frac{g \sin \theta + \cos \theta \left[\frac{-F - m_c l \dot{\theta}^2 \sin \theta + \mu_c \operatorname{sgn}(\dot{s})}{m_c + m_b} \right] - \frac{\mu_p \theta}{m_b l}}{l \left[\frac{4}{3} - \frac{m_b \cos^2 \theta}{m_c + m_b} \right]} \quad (3)$$

$$\ddot{s} = \frac{F + m_b l \left[\dot{\theta}^2 \sin \theta - \ddot{\theta} \cos \theta \right] - \mu_c \operatorname{sgn}(\dot{s})}{m_c + m_b}$$

Onde:

- $g = -9.8 \text{ m/s}^2$, aceleração causada pela gravidade,
- $m_c = 1.0 \text{ kg}$, massa do carro,
- $m_b = 0.1 \text{ kg}$, massa do bastão,
- $l = 0.5 \text{ m}$, metade do comprimento do bastão,
- $\mu_c = 0.0005$, coeficiente de atrito do carro na superfície,
- $\mu_p = 0.000002$, coeficiente de atrito do bastão no carro,
- $F = \pm 10.0 \text{ N}$, força aplicada no centro de massa do carro.

Barto et al. [12] utilizam subscrito t (omitido) para algumas variáveis do sistema diferencial (3) para explicitar a dependência do tempo (discreto). As equações em (??) serviram de base para implementação do ambiente simulado na biblioteca *Gym* [13].

2.2 OPENAI GYM

Antes de discutir a implementação do sistema na biblioteca *Gym* faremos uma análise das suas principais funcionalidades. *Gym* é uma biblioteca voltada principalmente para o teste de algoritmos de aprendizagem por reforço. Frequentemente, os ambientes de aprendizado para esses algoritmos são descritos por meio de *processos de decisão Markovianos*.

A principal ideia dos sistemas Markovianos é a interação de um *agente* com um *ambiente* externo (fora de seu controle). Essa interação se da por meio de ações, estados e recompensas, denotados por A_t , S_t e R_t , respectivamente. A cada passo de tempo, o ambiente fornece ao agente o seu estado atual S_t e uma recompensa R_t , o agente então seleciona uma ação A_t que servirá de entrada para o ambiente e influenciará sua dinâmica no próximo instante de tempo, com o objetivo de maximizar a recompensa acumulada. Esse processo é descrito graficamente na 15.

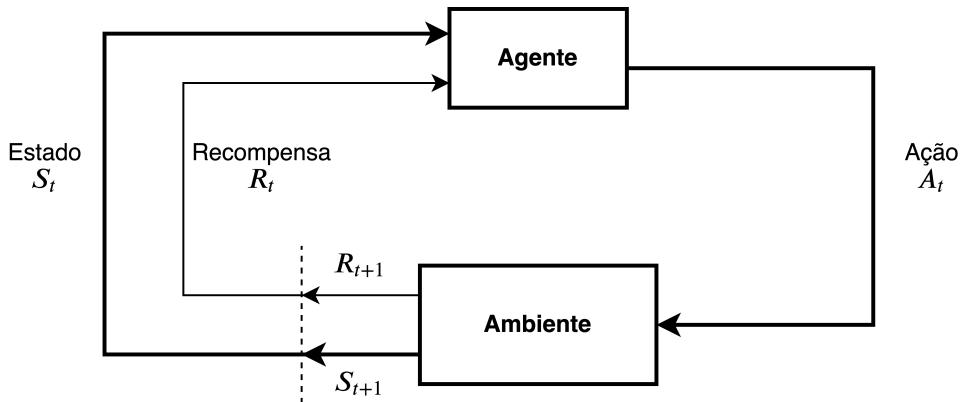


Figura 15: Processo de decisão de Markov.

Fonte: Adaptado de [14].

Para processos Markovianos, um estado S_t caracteriza completamente a dinâmica de um sistema, no instante t . Isto significa que para ambientes estocásticos Markovianos, dado um estado S_t e uma ação A_t , sabemos a probabilidade associada a cada possível próximo estado, S_{t+1} e recompensa futura R_{t+1} . Não há probabilidades associadas ao sistema do pêndulo invertido, logo, para o nosso problema, se há um estado Markoviano S_t observável, saberemos com precisão qual será o próximo estado do sistema e a recompensa futura recebida pelo agente.

As definições dos parágrafos anteriores são úteis para entendermos o funcionamento do ambiente que iremos utilizar. Os sistemas providos pela biblioteca são baseados em processos de decisão Markovianos, isto significa que a cada instante existe uma variável acessível ao agente que indica o estado atual do ambiente e sua recompensa, respectivamente, S_t e R_t . O agente será a máquina que aprende com a experiência, mais especificamente, com informações de estados e recompensas obtidos ao longo do tempo. Além disso, vimos que o agente influencia o sistema através de ações A_t , que também são proporcionadas pela biblioteca.

Vimos que estados Markovianos descrevem completamente o próximo estado do

ambiente. Por esse motivo, S_t deve conter informações suficientes para que possamos prever o próximo estado. Essa restrição impõe aos estados explica o motivo da observação fornecida ao agente pela biblioteca. Mais especificamente, para o problema do pêndulo invertido, a observação fornecida ao agente inclui as variáveis de estado:

Tabela 1: Variáveis de estado fornecidas pela biblioteca.

Variável	Significado	Valor Mínimo	Valor Máximo
s	Posição do carro	-4.8	4.8
\dot{s}	Velocidade do carro	$-\infty$	∞
θ	Ângulo do bastão	-24°	24°
$\dot{\theta}$	Velocidade angular do bastão	$-\infty$	∞

As variáveis de estado fornecidas são precisamente as necessárias para caracterização da dinâmica do sistema, conforme as equações (3). Isto significa que o estado observado obedece a propriedade de Markov, o que já era esperado. Vale notar que a modelagem do sistema na biblioteca é ligeiramente diferente pois não inclui os atritos.

A simulação ocorre em episódios de duração limitada, o término acontece quando as variáveis de estado ou o tempo decorrido excedem um certo valor. Caso o carro se distancie do centro em 2.4 m, o ângulo do bastão seja maior que 12° ou a duração seja maior que 200, a simulação termina.

As recompensas são fornecidas ao agente pelo ambiente, conforme a 15, em cada instante de tempo discreto. A implementação de recompensas R_t no ambiente do pêndulo invertido é simples: a cada instante de tempo, o agente recebe 1 de recompensa. Controladores com baixo desempenho irão forçar o término do episódio rapidamente, recebendo pouca recompensa. A ação que o agente pode tomar está restrita a aplicar uma força de 10 N para esquerda ou direita.

Obviamente, cada componente da simulação está implementado na linguagem de programação *Python* utilizando classes e tipos de dados específicos.

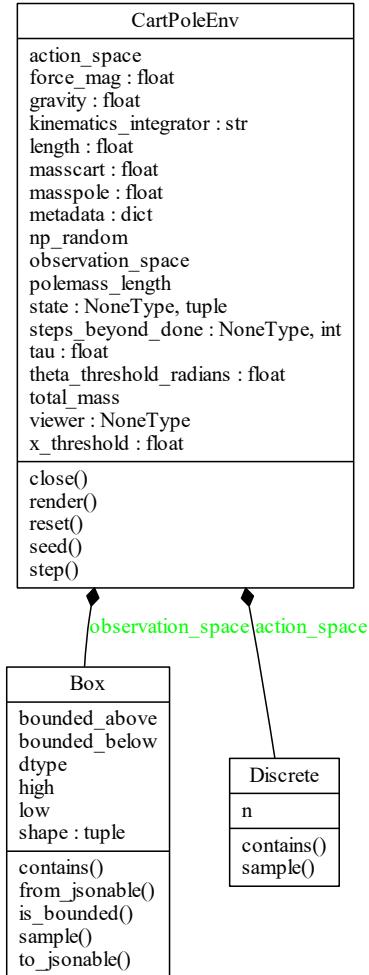


Figura 16: Diagrama de classes para o ambiente de simulação do pêndulo invertido, gerado com o utilitário *pyreverse*.

Na 16, verificamos que os objetos `observation_space` e `action_space`, pertencem às classes `Box` e `Discrete`, respectivamente, e compõe o ambiente de simulação. Essas classes determinam quais as observações e ações possíveis na simulação, isto é, restringem o tipo de ação que o agente pode tomar e a observação que pode ser recebida por ele. Na simulação do pêndulo invertido, as ações estão restritas aos valores inteiros 0 e 1, por exemplo. Os métodos implementados são utilizados para realizar a interação do agente com o ambiente. As principais funções utilizadas serão detalhadas a seguir:

a) `make(nome)`:

- **Descrição:** Cria um objeto em que ocorre a simulação.
- **Argumento:** (*String*) Nome do ambiente de simulação. Ex: 'CartPole-v1'

- **Retorno:** (*Env*) Objeto ambiente de simulação criado.

b) **reset(*ambiente*)**:

- **Descrição:** Inicia um episódio de simulação com valores iniciais aleatórios dentro de uma faixa específica.
- **Argumento:** (*Env*) O objeto (ambiente de simulação) que será inicializado.
- **Retorno:** (*Box*) Observação (objeto) que contém o valor das variáveis de estado.

c) **step(*ambiente, ação*)**:

- **Descrição:** Realiza a *ação* no sistema.
- **Argumento:** (*Env, Int*) Objeto com o ambiente de simulação e ação discreta do agente.
- **Retorno:** (*Tupla*) Retorna uma lista que contém: Observação (após a ação do agente), recompensa, indicação de término do episódio e informações adicionais.

d) **render(*ambiente, modo*)**:

- **Descrição:** Renderiza o ambiente de simulação.
- **Argumento:** (*Env, String*) Ambiente a ser simulado e o modo de exibição.
- **Retorno:** Objeto de renderização.

As funções descritas são métodos que atuam no ambiente de simulação. A utilização dessas funções podem ser exemplificadas no fluxograma da 17.

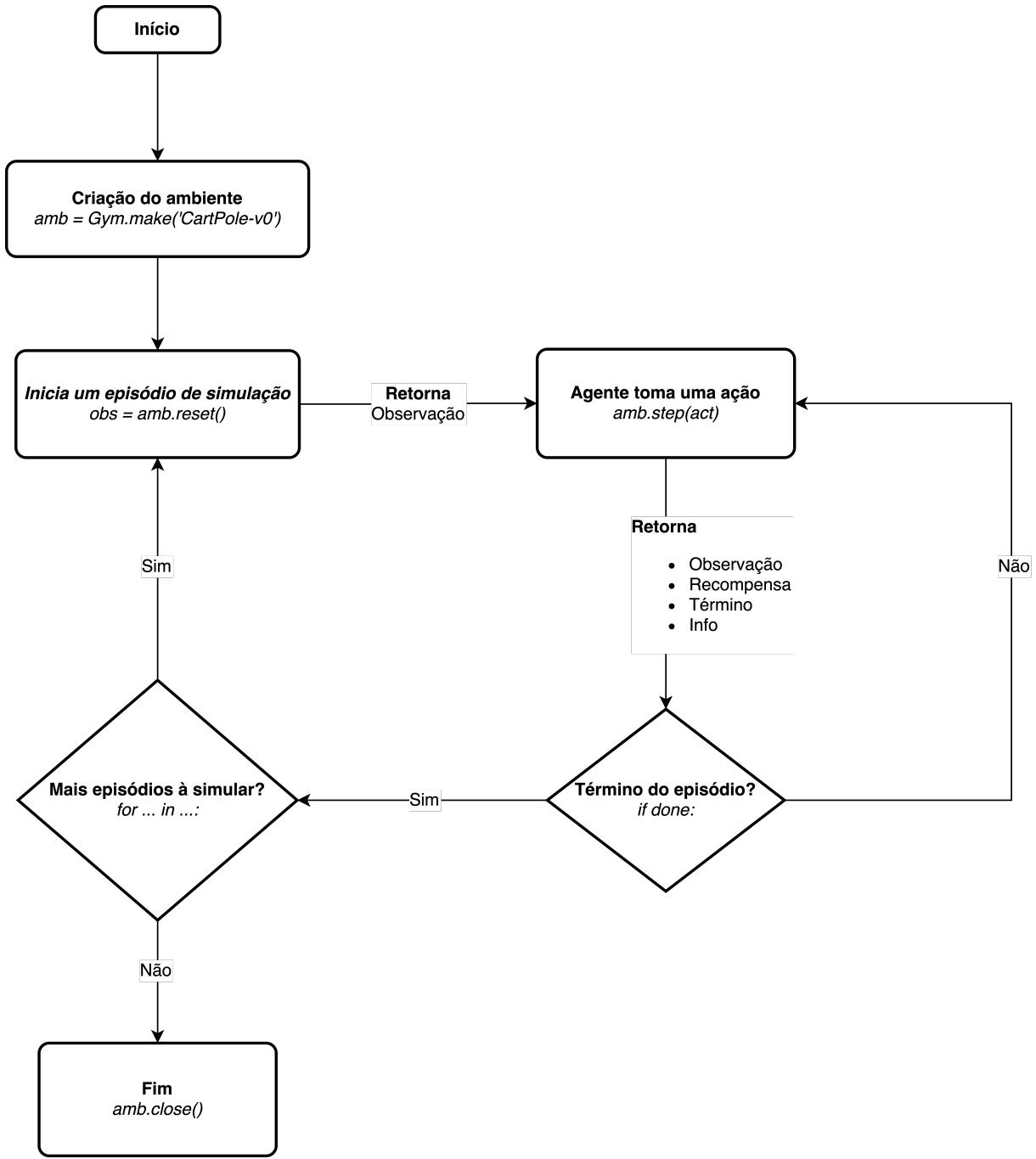


Figura 17: Forma geral de um algoritmo para realização de uma simulação no *OpenAi Gym*.

Será feita a seguir uma análise breve de cada objeto da interação demonstrada na 17, com o objetivo de facilitar a criação do algoritmo.

O principal objeto é o retornado pela função `step()`, de acordo com o item c anterior ou a 17. É uma tupla (lista imutável) que retorna quatro objetos que o agente utilizará para obter informações sobre o sistema, são eles:

- *Observação*: lista indexável que contém o valor das variáveis de estado do sistema

logo após a *ação* do agente.

Observação

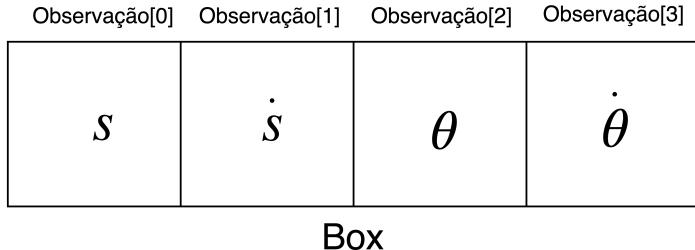


Figura 18: *Box* é uma classe que permite a criação de conjuntos multidimensionais, no ambiente do pêndulo invertido é apenas uma lista (unidimensional).

- *Recompensa*: valor inteiro retornado pelo sistema após ter sofrido uma *ação* do agente. Pode ter valor 0 ou 1.

Recompensa

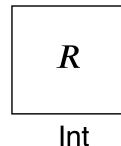


Figura 19: Recompensa recebida pelo agente.

- *Término*: valor lógico (verdadeiro ou falso) que indica se o episódio de simulação terminou ou não, de acordo com os critérios de término estabelecidos.

Término

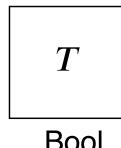


Figura 20: Valor lógico que indica se o episódio terminou ou não.

- *Info*: essa variável fornece informações adicionais úteis para verificar o funcionamento do algoritmo, geralmente não é utilizada.

Com isso, finalizamos a análise do problema do pêndulo invertido e como esta implementada a sua simulação. Possuímos as ferramentas necessárias para evoluir um agente utilizando a programação genética. A seguir veremos como podemos criar esses agentes e moldá-los para que possam ser utilizados no ambiente de simulação analisado.

3 DEAP: IMPLEMENTAÇÃO DOS AGENTES

No capítulo anterior, verificamos o funcionamento do ambiente de simulação proporcionado pela biblioteca Gym, ao analisar a interação do ambiente com o agente. Neste capítulo, teremos como objetivo verificar as funcionalidades providas pela biblioteca DEAP e como podemos utilizar essas ferramentas para representar indivíduos que serão evoluídos.

DEAP é uma biblioteca em *Python* que promove a criação de protótipos e ideias na área da computação evolutiva, com mecanismos que permitem a paralelização de tarefas. Suas áreas de aplicação, além da programação genética, incluem: algoritmos genéticos, estratégias evolucionárias e optimização por enxame de partículas [15].

Nosso trabalho, obviamente, será voltado para a sua aplicação em PG. O objetivo será implementar uma população de indivíduos, representados como árvores ou listas de tamanho variável, que representarão leis de controle para um sistema dinâmico.

Neste capítulo será feita a implementação da população e dos indivíduos que a compõe utilizando as ferramentas proporcionadas pela biblioteca. Analisaremos os parâmetros essenciais para a representação da solução, isto é, o tamanho da população, o número máximo de níveis das árvores, entre outros aspectos. Posteriormente, será implementada a seleção e avaliação de indivíduos e finalizaremos com a definição dos operadores genéticos e o critério de término.

Já que iremos trabalhar com a biblioteca DEAP ao longo deste capítulo, é proveitoso que tenhamos uma visão geral das funcionalidades de cada módulo da biblioteca, isso esta demonstrado na Figura 21.

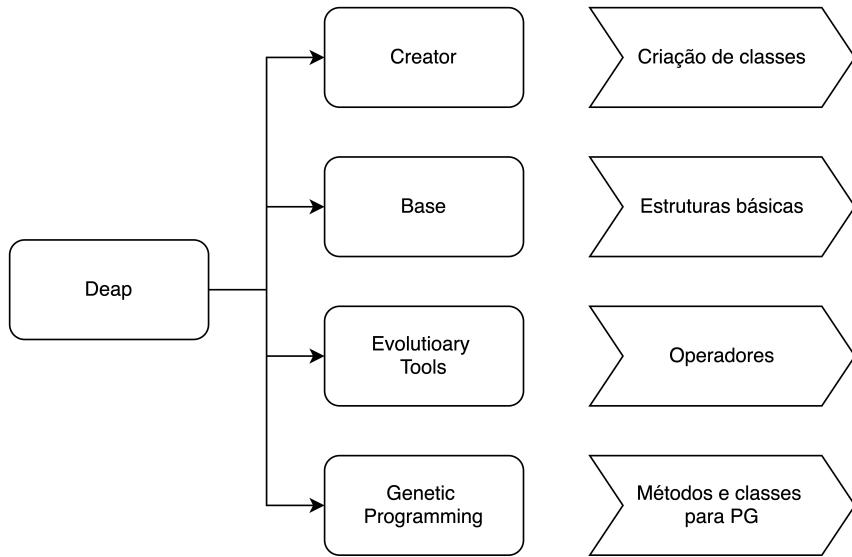


Figura 21: Módulos da biblioteca DEAP e suas funcionalidades.

3.1 REPRESENTAÇÃO E INICIALIZAÇÃO

Na Seção 1 fornecemos uma lista dos principais parâmetros que definem uma implementação de PG. Nessa seção lidaremos com os aspectos referentes à representação de cada indivíduo e sua respectiva inicialização, além disso serão definidos parâmetros adicionais da população como um todo. Ao fim desta seção, seremos capazes de criar uma população inicial aleatória pronta para sofrer transformações ao longo do ciclo evolucionário.

3.1.1 Representação

Vimos no Seção 1 que para representar um indivíduo na PG devemos definir: o conjuntos de operações matemáticas/lógicas e o conjunto de variáveis terminais, denominados por \mathcal{O} e \mathcal{V} , respectivamente. Geralmente essas duas entidades são agregadas em único conjunto denominado primitivo. Logo, pode-se definir como *conjunto primitivo* \mathcal{P} o grupo que contém todas as funções (de diferentes aridades) e variáveis terminais que irão compor os indivíduos da população.

As variáveis terminais podem ser constantes ou entradas vindas do sistema. As entradas serão simplesmente as variáveis de estado do sistema. Geralmente, as constantes são valores aleatórios, dentro de uma faixa, gerados na inicialização dos indivíduos. A existência de constantes aleatórias permite uma exploração maior do espaço de buscas, uma vez que as funções que atuam sobre esses números podem possivelmente modelar

coeficientes de uma lei de controle ótima para o problema.

Tabela 2: Variáveis terminais.

Variável Terminal	Significado	Tipo
s	Posição do carro	Entrada
s_p	Velocidade do carro	Entrada
t	Ângulo do bastão	Entrada
t_p	Velocidade angular do bastão	Entrada
R	número gerado aleatoriamente ($-1.0 < R < 1.0$)	Constante

Com base na literatura [9] [1] [16] usaremos as funções matemáticas e lógicas, conforme a Tabela 3.

Tabela 3: Conjunto de operadores lógicos e matemáticos.

Operador	Símbolo	Resultado	Aridade
Soma	add	$a + b$	2
Subtração	sub	$a - b$	2
Multiplicação	mul	$a \cdot b$	2
Divisão	div	a/b	2
Raiz Quadrada	sr	\sqrt{a}	1
Raiz Cúbica	cr	$\sqrt[3]{a}$	1
Cosseno	cos	$\cos(a)$	1
Comparação	gt	$a, \text{ se } a \geq b$ $b, \text{ se } a < b$	2
Sinal	sgn	$1, \text{ se } a \geq 0$ $-1, \text{ se } a < 0$	1

A operação de divisão geralmente é implementada de modo que o denominador possa assumir o valor zero, para evitar possíveis erros durante a execução do programa.

Um problema se apresenta quando pensamos em uma possível lei de controle gerada através das funções da Tabela 3 com as variáveis terminais da Tabela 2: a ação do

agente não se dá no espaço contínuo, isto é, a saída do agente deve ser 0 ou 1, conforme especificado no espaço de ações do problema (*action space*), para o pêndulo invertido.

Já que os indivíduos são expressões matemáticas que produzem resultados numéricos variados, uma possível solução seria definir que a saída produzida será 1, caso o número produzido na avaliação do indivíduo seja positivo e 0 caso seja negativo. Esse processo está exemplificado na Figura 22.

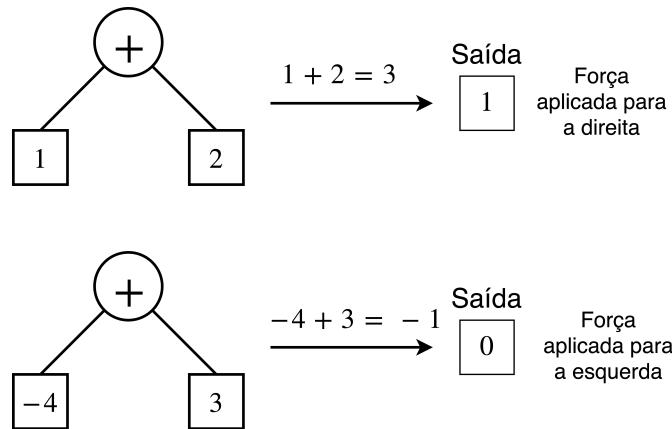


Figura 22: A saída depende do sinal da expressão que resulta da avaliação do indivíduo.

Com isso, garantimos que a ação realizada pelo indivíduo está contida no espaço de ações possíveis determinada pelo ambiente de simulação.

Resta-nos, no momento, agrupar as operações e variáveis terminais em um conjunto único. A biblioteca que será utilizada provê funcionalidades que permitem a criação de um conjunto primitivo. A partir da classe *PrimitiveSet* é possível criar tal grupo e incluir funções e variáveis terminais.

Classe PrimitiveSet (*name*, *arity*, *prefix*)

Cria um conjunto primitivo de nome *name* com *arity* entradas. É possível adicionar um prefixo *prefix* à nomeação de cada entrada.

Será dado o nome de *main* para o conjunto primitivo a ser criado por convenção. Como temos quatro variáveis de estado que usaremos como entrada, a aridade do conjunto primitivo será quatro. É conveniente renomear as entradas alterando o prefixo para facilitar o entendimento (em condições normais, as variáveis da Tabela 2 seriam nomeadas ARG0, ARG1, ARG2 e ARG3, por exemplo).

Após a criação do conjunto primitivo, a partir da classe descrita, é possível adicionar a esse objeto as funções da Tabela 3 e a constante efêmera da Tabela 2 (as outras variáveis terminais são consideradas entradas e são adicionadas na criação do conjunto \mathcal{P}).

Método `addPrimitive (op, arity, name)` de `PrimitiveSet`

Adiciona a operação lógica/matemática *op* ao conjunto primitivo, sob o nome *name*. O parâmetro *arity* define o número de entradas do conjunto.

Método `addEphemeral (name, constant)` de `PrimitiveSet`

Adiciona a *constante* ao conjunto primitivo sob o nome *name*.

A seguir, implementaremos um atributo de aptidão que será adicionado ao indivíduo em sua inicialização.

3.1.2 Aptidão

Vimos ao longo da Seção 1 que existem diversas maneiras de representar a aptidão de um indivíduo. Em problemas de controle, é comum expressar esse critério de desempenho como funções custo 2. Um indivíduo apto, portanto, seria caracterizado por um valor baixo de aptidão, ou seja, desejaríamos minimizar esse valor. Entretanto, poderíamos elaborar uma função de aptidão a ser maximizada, basta que essa função represente o tempo total do episódio de simulação atingido por um indivíduo, para o problema do pêndulo invertido. Dessa forma, uma solução com desempenho insatisfatório causaria rapidamente o término da simulação e estaria associada à um valor de aptidão baixo.

Já que o ambiente de simulação proporcionado pela biblioteca Gym provê uma recompensa unitária a cada intervalo de tempo discreto em que a simulação não termina, podemos calcular a aptidão de um indivíduo como a soma de recompensas acumulada ao longo do tempo. Como buscamos indivíduos com o máximo de aptidão, podemos nomear essa classe como *AptidaoMax*.

O DEAP proporciona uma classe base denominada *Fitness* com um atributo *weight*

(peso). O peso indica se a aptidão deve ser maximizada ou minimizada. Um peso positivo, por exemplo, indica que o objetivo do ciclo evolucionário é gerar indivíduos com aptidão maior possível.

A biblioteca conta também com uma função de criação de classes, denominada *create* da biblioteca *creator*, que permite criar novas classes a partir de outras primitivas.

Função *create* (*name*, *base*, *attributes*)

Cria a classe com o nome *name* a partir da classe *base* com os atributos adicionais *attributes*.

O atributo peso permite facilmente comparar dois indivíduos pois é possível ignorar a implementação da função de avaliação, ou seja, o maior valor numérico de aptidão indica o indivíduo mais apto, seja o objetivo a maximização ou minimização.

O peso é um atributo representado em forma de *tupla* o que permite a utilização de funções de aptidão com diferentes pesos, o que é útil para aplicações com múltiplos objetivos.

3.1.3 Inicialização

Os indivíduos de uma população são árvores cujo conteúdo pertence a um conjunto primitivo e que possuem um atributo de aptidão, que caracteriza o desempenho da solução no sistema. Tais características de um indivíduo foram implementadas nas seções 3.1.1 e 3.1.2, resta-nos, portanto, criar uma estrutura de dados que possa ser inicializada de forma aleatória utilizando o conjunto primitivo e que possua um atributo de aptidão.

As principais escolhas associadas a esta etapa estão relacionadas a profundidade das árvores representando os indivíduos e o método de inicialização, ou seja, como essas árvores seriam geradas.

Na Seção 1 definimos como D_{\min} a profundidade mínima da árvore que representa um indivíduo inicializado, isto é, que faz parte da população inicial. D_{\max} foi estabelecido como a profundidade máxima de um indivíduo inicializado, ou seja, a maior distância da raiz a uma variável terminal.

Serão escolhidos os valores 2 e 5 para D_{\min} e D_{\max} , respectivamente, para o problema do pêndulo invertido. Na 23 esta um exemplo de indivíduo inicializado com valores extremos de profundidade da árvore.

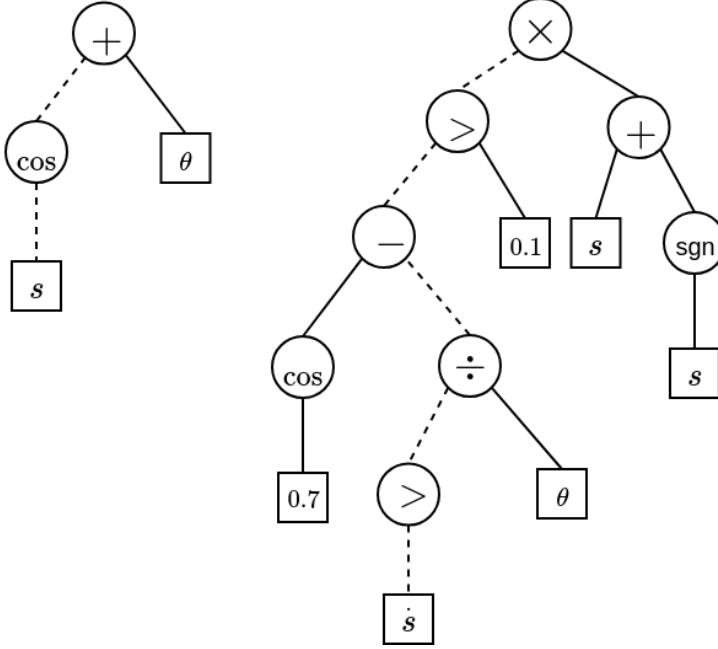


Figura 23: As linhas pontilhadas indicam a maior distância entre a *raiz* e uma *variável terminal*. Este exemplo utiliza o conjunto primitivo \mathcal{P} definido anteriormente.

Pode-se notar que uma árvore pode conter diferentes profundidades, com o valor entre os extremos pré-estabelecidos. Por exemplo, na 23, a árvore da direita possui caminhos com comprimento mínimo apesar de conter um caminho de profundidade máxima. Se partirmos do princípio que a criação dos indivíduos inicia a partir da raiz, o que determina a profundidade máxima do indivíduo a ser criado é a escolha: o próximo nó é um operador ou uma variável terminal?

No processo recursivo de criação de árvores a partir da raiz é possível fazer com que todos os caminhos possíveis da árvore sejam o maior possível, D_{\max} . Esse método de inicialização é chamado de *full* [1]. Os indivíduos poderiam ser gerados com caminhos de comprimento aleatório, isto é: a única restrição é que cada caminho deveria ter um comprimento l_i tal que $D_{\min} < l_i < D_{\max}$. Este método é conhecido como *grow*.

Koza [1] sugere um método híbrido, frequentemente empregado, denominado *ramped half-and-half*. Nessa abordagem são gerados N indivíduos para cada nível máximo de profundidade desde D_{\min} a D_{\max} , de forma que, para cada nível, metade dos indivíduos são gerados através do método full e a outra metade é gerada pela rotina grow.

A biblioteca DEAP provê os três métodos de inicialização, porém utilizaremos apenas a rotina *ramped half-and-half*, implementada pela função descrita abaixo, do módulo *gp*.

Função genHalfAndHalf (*pset*, *min*, *max*)

Gera uma expressão (árvore) através do conjunto primitivo *pset* pelo método grow ou full, utilizando os valores *min* e *max* de comprimento mínimo e máximo, respectivamente.

As expressões geradas pela função descrita acima possuem estrutura recursiva semelhante à forma de árvore vista, construída a partir do conjunto primitivo, porém não há associação dessa estrutura com uma aptidão própria.

Representaremos indivíduos, de fato, utilizando a expressão gerada pela função acima, com o atributo *AptidaoMax* criado anteriormente.

A classe *PrimitiveTree* proporciona diversos métodos para essas expressões: é possível verificar a profundidade da árvore, o operador correspondente a raiz ou até procurar funções dentro da própria árvore. Portanto, é proveitoso que utilizemos essa classe como base para a representação de indivíduos.

Utilizando novamente a função *create* é possível criar uma nova classe *Individuo* baseado na classe *PrimitiveTree* com os atributo *AptidaoMax* e *ConjPrim* (conjunto primitivo), isto é, fornecemos como características adicionais uma aptidão a ser maximizada e um conjunto primitivo, a cada indivíduo.

Com isso, já é possível criar indivíduos representados como árvores recursivas, construídas a partir de um conjunto primitivo, e com um atributo de aptidão que pode ser calculado durante a execução da simulação.

A seguir trataremos da criação de múltiplas instâncias de indivíduos que irão compor uma população. A representação da população em si também será abordada.

3.1.4 População

Em relação à população trataremos de apenas um parâmetro no momento: o número de indivíduos. Esta bem estabelecido na literatura a importância desse parâmetro para a evolução de programas [1].

Em geral, a determinação do tamanho ótimo da população aumenta com a complexidade do problema. Em contrapartida, a eficiência do programa genético é mais sensível ao tamanho populacional para problemas de menor complexidade [17].

A PG se mostra robusta em uma variedade de problemas e a determinação precisa dos parâmetros não é necessária, pois tipicamente obterá resultados satisfatórios. É prática comum o uso de populações com no mínimo 500 indivíduos [16].

A inicialização de uma população envolve a criação de instâncias da classe *Individuo* criada anteriormente. Para auxiliar na criação da população inicial, podemos utilizar o módulo *Evolutionary Tools* que provê uma função denominada *register* que auxilia na criação de funções com argumentos pré-determinados.

Mais especificamente, a classe *Toolbox* permite agrupar diversas funções em um único objeto. É possível, então, fornecer um objeto dessa classe (que efetivamente significa "caixa de ferramentas") a um algoritmo que implementa o ciclo genético completamente ou parcialmente.

Função register (*alias, fun, args*)

Registra uma função de nome *alias* que funciona como a função *fun* chamada com os argumentos *args*.

Se o *alias* de uma função registrada corresponder a **mate** ou **mutate**, estamos indicando que essa função implementa um cruzamento ou uma mutação, respectivamente. Alguns algoritmos que realizam um ciclo genético irão buscar dentro do objeto *Toolbox* as funções chamadas *mate* ou *mutate* ao realizar as operações de cruzamento ou mutação.

Vimos que as expressões recursivas geradas com a função *genHalfAndHalf* acima se assemelham muito aos objetos da classe *PrimitiveTree*. De fato, a representação interna dos dois objetos são listas ordenadas de operações e variáveis terminais do conjunto primitivo.

Para atribuir os elementos retornados de uma função geradora de lista (como a função *genHalfAndHalf*) à outro objeto representado por uma lista (como a classe criada *Individuo*), utilizamos a função *initIterate*.

Função initIterate (*container, generator*)

Armazena o conteúdo gerado pela função *generator* no objeto de tipo ou classe *container*.

A criação da população consiste no processo iterativo de criação de indivíduos

colocando-os em um objeto de armazenamento. Tal processo iterativo de armazenamento pode ser obtido através da função *initRepeat*:

Função *initRepeat* (*container*, *fun*, *n*)

Chama a função de geração de objetos *fun*, *n* vezes,
preenchendo um objeto de tipo/classe *container*.

É possível, então armazenar a população de indivíduos em uma lista a partir do tipo básico *list*, chamando a função de geração de indivíduos 500 vezes.

Com isso temos todos as funções necessárias para inicializar uma população completa. Verificaremos a seguir como podemos avaliar indivíduos de acordo com seu desempenho no ambiente de simulação.

3.2 AVALIAÇÃO DE INDIVÍDUOS

Com a criação da população inicial abordada ao longo da Seção 3.1, é necessário avaliar cada indivíduo de acordo com uma função de aptidão (ou custo). Essa avaliação deve ocorrer logo após a criação da população e depois de cada aplicação de operadores genéticos, isto é, cada geração necessita de uma avaliação completa de todos os seus membros.

A função de avaliação utilizará os resultados da interação do indivíduo com o sistema dinâmico, através da simulação. De maneira geral, a partir dos conceitos apresentados na Figura 17, a função de avaliação utiliza como argumentos uma observação das variáveis de estado. Por exemplo, no sistema do pêndulo invertido, uma função de avaliação possível seria a soma das diferenças, ao longo do tempo, do ângulo atual $\theta(t)$ em relação ao alvo $\theta = 0$.

Uma das vantagens da biblioteca Gym é a implementação das recompensas, que auxiliam no cálculo da função de aptidão. De certa forma, a recompensa é uma função de aptidão, e pode, portanto, ser utilizada para o cálculo do desempenho do indivíduo. Para generalizar a implementação de aptidão para os ambientes da biblioteca Gym, é possível definir:

$$A(t) = f_a(O(t), r(t)) \quad (4)$$

$$A_{tot} = \sum_{t=0}^T A(t)$$

Onde T representa o instante de término da simulação, $O(t)$ é a observação no instante t e $r(t)$ é a recompensa. Para o problema do pêndulo invertido, serão utilizadas as seguintes expressões para o cálculo da aptidão de um indivíduo:

$$A(t) = f_a(r(t)) = 1, \forall t < T \quad (5)$$

$$A_{tot} = \sum_{t=0}^T 1$$

Basta, portanto, iniciar um episódio com $A_{tot} = 0$ e somar 1 a cada instante de tempo em que não ocorre o término da simulação.

Vimos na Seção 2.2 que os valores iniciais das variáveis de estado são aleatórios e restritos à uma faixa específica. Caso o cálculo da aptidão de um indivíduo seja realizado com base em apenas uma simulação, é possível que a aptidão atribuída não reflita a real capacidade da solução em relação à todas situações possíveis. Realizar a simulação em diversos episódios diferentes, auxilia na convergência da busca do indivíduo que resolva o problema para qualquer condição inicial.

Em geral, a amostragem estocástica que utiliza apenas um episódio para a avaliação do indivíduo não gera resultados confiáveis. Essa afirmação pode ser provada através de conceitos estatísticos [18]. Para o problema do pêndulo invertido, podemos considerar, por exemplo, a utilização de 10 episódios para a avaliação de um indivíduo. Nesse caso, a aptidão de uma solução será dada por:

$$\bar{A} = \frac{1}{nep} \sum_{ep=1}^{nep} A_{tot}^{ep}$$

$$\bar{A} = \frac{1}{10} \sum_{ep=1}^{10} A_{tot}^{ep}$$

(6)

Onde A_{tot}^{ep} representa a aptidão do indivíduo no episódio ep e nep é o número de episódios ou simulações.

Avaliar um indivíduo, em suma, significa utilizar a lei de controle (representada em forma de árvore) no sistema dinâmico, em diversos episódios. Como o número que resulta de uma árvore, em um instante de tempo t , depende do valor atual de cada variável de estado, fica claro que a função de controle f_c tem como argumento $O(t)$.

$$f_c(t) = f_c(O(t)) = f_c(s, \dot{s}, \theta, \dot{\theta}) \quad (7)$$

$O(t)$ representa precisamente as entradas do sistema, utilizadas como variáveis terminais, de acordo com a Tabela 2. É necessário, portanto, transformar a representação de um indivíduo em forma de árvore, em uma função de controle que receba como argumentos as variáveis terminais de entrada. A função *compile* do módulo *Genetic Programming* realiza esta tarefa.

Função *compile* (*expr*, *pset*)

Retorna uma função de n argumentos ao avaliar a expressão *expr* no contexto do conjunto primitivo *pset* ou um valor numérico caso o número de entradas seja nulo.

No início da avaliação de cada indivíduo, o mesmo é compilado, e a função de controle gerada é utilizada em cada episódio de simulação, a cada instante de tempo.

A função de avaliação será registrada no objeto da classe Toolbox, sob o nome de *evaluate* (avalia), dessa forma a função de avaliação de uma população pode ser chamada

automaticamente em cada iteração do ciclo evolucionário.

3.3 SELEÇÃO DE INDIVÍDUOS

Ao longo da Seção 1 mencionamos o papel da seleção realizada na população. De forma geral, esse processo busca inspiração na seleção natural descrita por Darwin. Ao longo desta seção, teremos como objetivo analisar as principais formas de implementar essa seleção na população da PG.

Um dos principais conceitos quando analisamos métodos de seleção é a *pressão de seleção*, que indica o quanto favorecido os indivíduos mais aptos serão. Isto é, quanto maior a pressão de seleção mais discriminante será o ambiente, o que pode levar à uma rápida perda de diversidade da população [16].

O método conhecido como *roulette wheel* associa a cada indivíduo uma probabilidade de ser selecionado, de modo que, quanto maior sua aptidão, maior a probabilidade. Por este motivo, esse método também é chamado de *fitness proportionate* (proporcional à aptidão). Esse processo exerce uma pressão de seleção grande à população e possui um custo computacional maior se comparado ao próximo.

O método chamado de *campeonato* foi descrito brevemente no Capítulo 1. Relembando, selecionamos aleatoriamente dois indivíduos na população e comparamos as suas respectivas aptidões. Isso permite a seleção da solução mais apta ao problema. É possível selecionar n indivíduos, dessa forma, o campeonato possuirá diversos níveis e o indivíduo de melhor desempenho será obrigatoriamente selecionado dentre o conjunto de n soluções. Uma das principais vantagens desse método é a facilidade de sua implementação e baixo custo computacional. Aliado a isto, esse método não promove uma pressão de seleção grande o suficiente à ponto de levar a perda rápida de diversidade da população.

O DEAP promove o método descrito acima com controle de um processo indesejável chamado de *bloat*. Esse processo é caracterizado por um aumento do comprimento dos indivíduos, sem retorno significativo em termos de desempenho [16], causando um aumento proporcional ao custo computacional de evoluir tais indivíduos. O método de seleção *selDoubleTournament* promove dois campeonatos: um baseado na aptidão e outro no tamanho dos indivíduos, de forma que os que possuem menor comprimento máximo de árvore tenham chances maiores de serem selecionados. A ordem em que cada tipo de campeonato é aplicado aparenta não ter significância [?].

Função selDoubleTournament (*indList*, *k*, *fitSize*, *lenSize*, *fitFirst*, *fitness*)

Seleciona *k* indivíduos de uma lista *indList* a partir de um campeonato com *fitSize* indivíduos. Caso *fitFirst* seja verdadeiro, o segundo campeonato será baseado no comprimento e *lenSize* $\in [1, 2]$ determina a pressão de seleção do campeonato, de modo que, se *lenSize* = 2, o menor indivíduo será selecionado deterministicamente.

Seguindo o mesmo conceito da implementação da função de avaliação, registraremos a função de seleção sob o nome *select*.

3.4 OPERADORES GENÉTICOS

Ao longo desta seção discutiremos a aplicação dos operadores genéticos (cruzamento, mutação e replicação) nos indivíduos selecionados. Além disso, serão definidas as probabilidades associadas a cada operação.

3.4.1 Replicação

A replicação é o operador mais simples de ser implementado pois consiste apenas na cópia de um indivíduo para a próxima geração. A importância desse operador reside em garantir uma estabilidade no processo de convergência [?], mais especificamente, esse mecanismo permite que parte da população permaneça nas proximidades de mínimos locais do espaço de busca. Os outros operadores, mutação e cruzamento irão, por sua vez, realizar os conceitos discutidos na Seção 1 de busca global e local, respectivamente.

3.4.2 Mutação

A mutação é um operador genético eficiente e sua importância foi demonstrada por [19]. Uma de suas principais utilidades reside em promover a necessária variabilidade na população, além de realizar buscas amplas por regiões de convergência. A operação de mutação começa selecionando um ponto aleatório dentro da árvore de representação do indivíduo. Tal ponto pode ser uma função ou variável terminal. É comum substituir o elemento desse ponto por uma raiz que irá gerar uma sub-árvore aleatória. É possível utilizar

a mesma função geradora de expressões da inicialização de indivíduos, ou implementar uma outra função com diferentes parâmetros de comprimento mínimo e máximo.

Função `mutUniform (individual, expr, pset)`

Seleciona um ponto aleatório em um indivíduo *individual*, e substitui esse ponto por uma sub-árvore gerada com a expressão *expr*, utilizando os operadores de *pset*.

Utilizaremos a classe Toolbox, abordada na seção 3.1.4 para registrar uma função que implementa a mutação. Basicamente essa função chamará *mutUniform* com alguns argumentos fixos. Para que seja possível utilizar a mesma função de mutação em algoritmos que realizam o ciclo evolucionário, é proveitoso nomeá-la *mutate*.

Como uma ferramenta adicional para o controle de *bloat*, é possível alterar o funcionamento da função descrita acima de forma a automaticamente descartar indivíduos com comprimento máximo maior que um certo limite.

O conceito de *decoração* em Python, que altera o funcionamento de uma determinada função, é implementado na biblioteca por meio do método *decorate*. É possível alterar o funcionamento da operação de mutação de forma a eliminar indivíduos gerados cujo comprimento máximo ultrapasse um limite pré-estabelecido.

Com isso, caso um indivíduo criado a partir de uma mutação possua um comprimento máximo de árvore maior que 17, valor de referência utilizado em [1], o mesmo será automaticamente removido. O objetivo desse procedimento é controlar o processo de *bloat* mencionado anteriormente.

3.4.3 Cruzamento

Cruzamento é o operador genético capaz de explorar estruturas aptas e realizar a otimização local [9]. Geralmente, a probabilidade de realizar esta operação é maior, se comparada às outras duas já abordadas, por se tratar do principal mecanismo de busca local.

O cruzamento de um ponto, assim como definido na 11, é implementado na biblioteca através da função *cxOnePoint*.

Função cxOnePoint (*ind1*, *ind2*)

Executa uma operação de cruzamento entre os indivíduos *ind1* e *ind2*, selecionando aleatoriamente um ponto da árvore de cada um. Retorna uma tupla de dois indivíduos.

Utilizando o mesmo princípio da implementação de mutação, adicionaremos ao objeto Toolbox a operação de cruzamento, dando o nome de *mate*. Em seguida, adicionamos o controle de bloat com o mesmo limite estático de comprimento dos indivíduos.

Em seguida, adicionamos o controle de bloat com o mesmo limite estático de comprimento dos indivíduos.

3.4.4 Seleção Aleatória do Operador

Não foi mencionado até o momento como associar a cada operador genético uma probabilidade característica, permitindo a seleção aleatória de uma operação. Com base na Figura 5, verifica-se que os operadores genéticos atuam em indivíduos selecionados. Como aplicar um operador genético de forma aleatória será um dos tópicos abordados nesta etapa.

É interessante notar que existem duas abordagens fundamentais com relação a aplicação dos operadores. A primeira, objeto de estudo deste trabalho, funciona exatamente como descrito na Figura 13: o indivíduo selecionado sofre uma, e apenas uma, operação genética (não é possível que um indivíduo sofra mutação e replicação no mesmo ciclo, por exemplo), além disso, não é possível que o indivíduo selecionado **não** sofra ação de qualquer um dos operadores genéticos. Já que o indivíduo pode sofrer variação por meio de um operador genético **ou** outro, esse processo é denominado *varOr* no DEAP, união das palavras *variation* e *or*.

A segunda abordagem, denominada *varAnd*, realiza a variação de um indivíduo com, possivelmente, mais de uma operação genética, isto é, o indivíduo pode sofrer mutação **e** cruzamento, no mesmo ciclo. É possível, inclusive, que o indivíduo selecionado não sofra qualquer operação genética.

Para a abordagem *varOr* que será utilizada, basta que sejam selecionadas as probabilidades dos operadores de mutação e cruzamento. Como a soma de P_c , P_m e P_r deve

ser igual a 1, de imediato temos que a probabilidade de replicação será $1 - P_c - P_m$.

3.5 CICLO ITERATIVO

Ao longo desta seção foram criadas as ferramentas necessárias para a inicialização e modificação de uma população de indivíduos. Mais especificamente, foram tratados os problemas de representação, inicialização, seleção e operações genéticas. Entretanto, ainda não foi visto como essas implementações irão interagir entre si, em um processo iterativo, com o objetivo de gerar uma solução eficaz.

Basicamente, o objetivo é implementar o processo descrito no fluxograma da Figura 5. A biblioteca fornece algoritmos que realizam o processo iterativo evolucionário, utilizando as funções registradas na classe Toolbox. Tais funções se encontram no módulo *Algorithms* e permitem a simplificação do código criado. A função que se assemelha ao ciclo descrito na Figura 13 é a *eaMuPlusLambda*.

Função eaMuCommaLambda (*population, toolbox, mu, lambda, cxpb, mutpb, ngen, stats, halloffame, verbose*)

Repete *ngen* vezes o ciclo: aplica a função de avaliação (*evaluate*) na população (*population*), aplica a função de seleção (*select*) escolhendo *mu* indivíduos, em seguida recompõe a população original de *lambda* membros através da função *VarOr*.

Notamos que é necessário fornecer como argumento o objeto *Toolbox*, já que o mesmo possui as funções registradas: *evaluate*, *select*, *mate* e *mutate*. As probabilidades associadas a cada operador genético são definidas na própria chamada da função.

O número de gerações (*ngen*) define o critério de término do ciclo iterativo. Esse número será alterado de acordo com a complexidade do problema. Os outros parâmetros (*stats, halloffame* e *verbose*) ajustam a captura de estatísticas relacionadas à evolução.

Finalmente, a função *eaMuCommaLambda* também toma cuida do paralelismo da avaliação dos indivíduos. Já que a simulação de cada solução não depende de outra, é possível agilizar a execução do ciclo nesta, que é justamente a de maior custo computacional.

4 DEAP: ESTUDOS DE CASO

Ao longo do capítulo 3 lidamos com os principais aspectos da implementação da PG no contexto da biblioteca DEAP. Principalmente, abordamos a implementação da: representação e inicialização de indivíduos, avaliação, seleção e alteração dos indivíduos. O que se espera em cada ciclo iterativo são valores de aptidão médios e máximos crescentes.

Começaremos pelo problema básico abordado na seção 2, o pêndulo invertido. Ao longo desta seção, abordaremos outros problemas que podem ser simulados na biblioteca Gym.

Para cada caso, será adicionada uma tabela com os principais parâmetros do ciclo evolucionário. Além disso, alguns aspectos adicionais particulares a cada caso serão incluídos, como por exemplo, a função de avaliação.

Abaixo, será incluída uma breve explicação de cada parâmetro e sua implementação.

- *Tamanho da população (tam_pop)*: O número de indivíduos da população.
- *Probabilidade de cruzamento (pb_cx)*: Probabilidade de que a operação escolhida seja cruzamento.
- *Probabilidade de mutação (pb_mut)*: Probabilidade de que a operação escolhida seja mutação.
- *Número de gerações (n_geracoes)*: Quantas vezes o ciclo de avaliação, seleção e variação da população ocorrerá.
- *Tipo de aptidão (tipo_apt)*: Define se deseja-se minimizar ou maximizar uma medida de aptidão. Ex: 1.0 indica que o objetivo é maximizar a função de aptidão, enquanto que -1.0 indica o objetivo de minimizar a essa medida.
- *Número de entradas (n_entradas)*: Número de variáveis de estado (terminais).
- *Faixa para a constante efêmera (faixa_cst)*: Valor mínimo e máximo que limitam os valores possíveis para a variável terminal de valor aleatório.
- *Número de simulações (n_episodes)*: Número de episódios (simulações) para avaliação de cada indivíduo.

- *Tamanho do campeonato de aptidão (camp_apt)*: Número de rodadas para selecionar o indivíduo com maior aptidão. Ex: para $fittourn = 3$, serão selecionados 2^3 indivíduos e o de maior aptidão será selecionado.
- *Tamanho do campeonato de comprimento (camp_d)*: Pressão de seleção sobre indivíduos selecionados pelo campeonato de aptidão (valor entre 1 e 2, quanto maior o valor, maior a pressão de seleção sobre o menor indivíduo).
- *Operações*: Operações que irão compor os nós não-terminais da árvore.
- *Comprimento mínimo e máximo de inicialização (d_min, d_max)*: comprimentos mínimos e máximos possíveis para inicialização de cada indivíduo.
- *Comprimento máximo de mutação (max_d_mut)*: Utilizaremos uma função geradora de expressões diferente para a mutação. O comprimento mínimo será sempre zero, entretanto o máximo irá variar, dependendo da complexidade da solução esperada.
- *Límite de comprimento dos indivíduos (limite_d)*: Tamanho máximo dos indivíduos gerados através das operações de mutação e cruzamento (controle de bloat).

Em cada execução do algoritmo, as seguintes estatísticas relacionadas à evolução da população e à execução do programa serão obtidas:

- a) Em relação à aptidão: A aptidão é o principal parâmetro a ser observado e a expectativa é que essa medida aumente ao longo do tempo. Ao longo de cada geração, as seguintes medidas de aptidão serão obtidas:
 - *Mínimo*: Valor de aptidão do indivíduo de pior desempenho.
 - *Máximo*: Valor de aptidão do indivíduo de melhor desempenho.
 - *Média*: Aptidão média da população.
- b) Em relação ao comprimento: Medida que indica o maior comprimento (ou profundidade) da árvore, conforme indica a Figura 23. As medidas relacionadas à esse parâmetro, ao longo de cada geração, serão:
 - *Mínimo*: Valor mínimo de profundidade encontrado na geração.
 - *Máximo*: Valor máximo de profundidade encontrado na geração.

- *Média*: Valor médio de profundidade da população.
- c) Em relação à complexidade: Essa medida indica o número total de nós em um indivíduo, isto é, o número de operadores somado ao número de variáveis terminais. Serão consideradas medidas semelhantes à utilizadas nos items anteriores, também ao longo de cada geração.
- *Mínimo*: Menor número de nós encontrado em um indivíduo na população.
 - *Máximo*: Maior número de nós encontrado em um indivíduo na população.
 - *Média*: Número médio de nós da população, para cada indivíduo.

Será feita também uma comparação com dois algoritmos de aprendizagem por reforço: **DQN** (*Deep Q-Learning*) [20] e **DDPG** (*Deep Deterministic Policy Gradient*) [21].

4.1 Pêndulo Invertido

O problema básico que foi escolhido para servir de base para a aplicação do algoritmo foi o pêndulo invertido. Com o que foi abordado ao longo deste projeto, é possível montar a seguinte tabela de parâmetros utilizados:

Tabela 4: Parâmetros utilizados para o problema do pêndulo invertido.

Parâmetro	Valor
Tamanho da População	500
Probabilidade de Cruzamento	0.75
Probabilidade de Mutação	0.05
Número de Gerações	15
Número de Entradas	4
Faixa para Constante Efêmera	(-1.0, 1.0)
Número de Simulações	10
Tamanho do Campeonato de Aptidão	6
Tamanho do Campeonato de Comprimento	1.2
Operações	add, sub, mul, div, sr, cr, cos, gt, sgn
Comprimento Mínimo e Máximo de Inicialização	(1, 3)
Comprimento Máximo de Mutação	5
Límite de Comprimento dos Indivíduos	17

A função de cálculo de aptidão, é tal como descrita na equação 5, isto é, a aptidão de um indivíduo é, em suma, o tempo total em que o bastão permanece equilibrado, respeitando os limites estabelecidos na Tabela 1. Já que a simulação tem uma duração limite de 500 instantes de tempo, a aptidão máxima possível é 500.

A aptidão final de um indivíduo, após a avaliação, é a aptidão média obtida em todos os episódios que o indivíduo participou (parâmetro representado pelo número de simulações).

Utilizando os dados da Tabela 4, o algoritmo foi executado 10 vezes em sequência e a média das estatísticas foram obtidas.

Parte do algoritmo é dedicada à obtenção de dados e, apesar das características estocásticas do processo, o pseudo-gerador de números aleatórios possui uma semente fixa, permitindo a reprodução dos resultados obtidos. O algoritmo completo encontra-se no apêndice A.

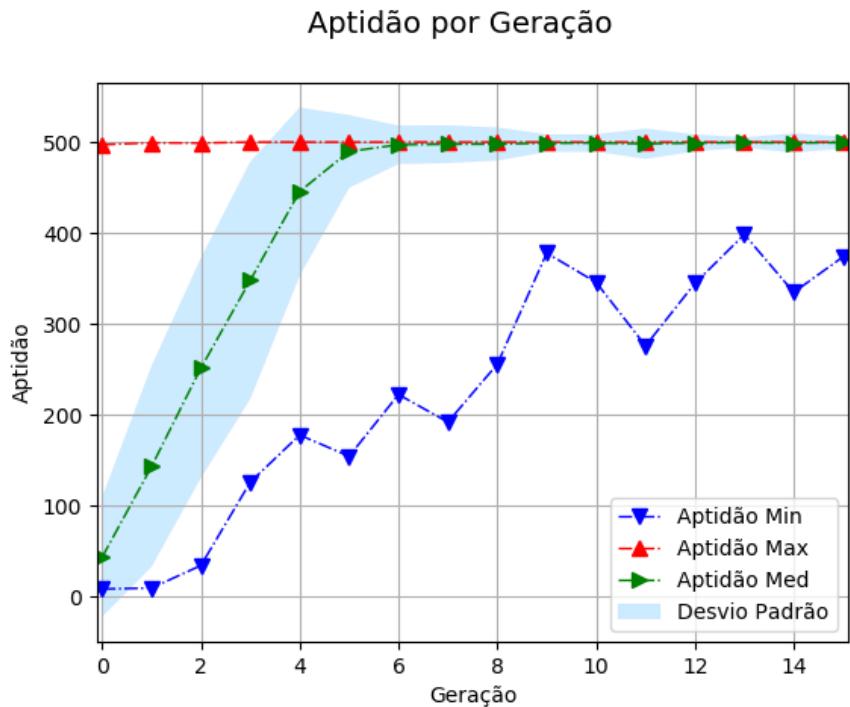


Figura 24: Média da aptidão de todos os indivíduos por geração (verde). Valor máximo de aptidão observado em cada geração (vermelho). Menor aptidão observada em cada geração (azul).

Observa-se no gráfico da Figura 24 que logo na primeira geração alguns indivíduos já são capazes de obter a aptidão máxima. Isto se deve ao grande número de indivíduos (500) gerados aleatoriamente o que funciona, de certa forma, como uma busca exaustiva aleatória.

Entretanto, é possível observar o aumento da aptidão média e mínima ao longo das gerações. O gráfico da Figura 25 mostra o número de indivíduos que atingiram uma determinada faixa de aptidão, em cada geração.

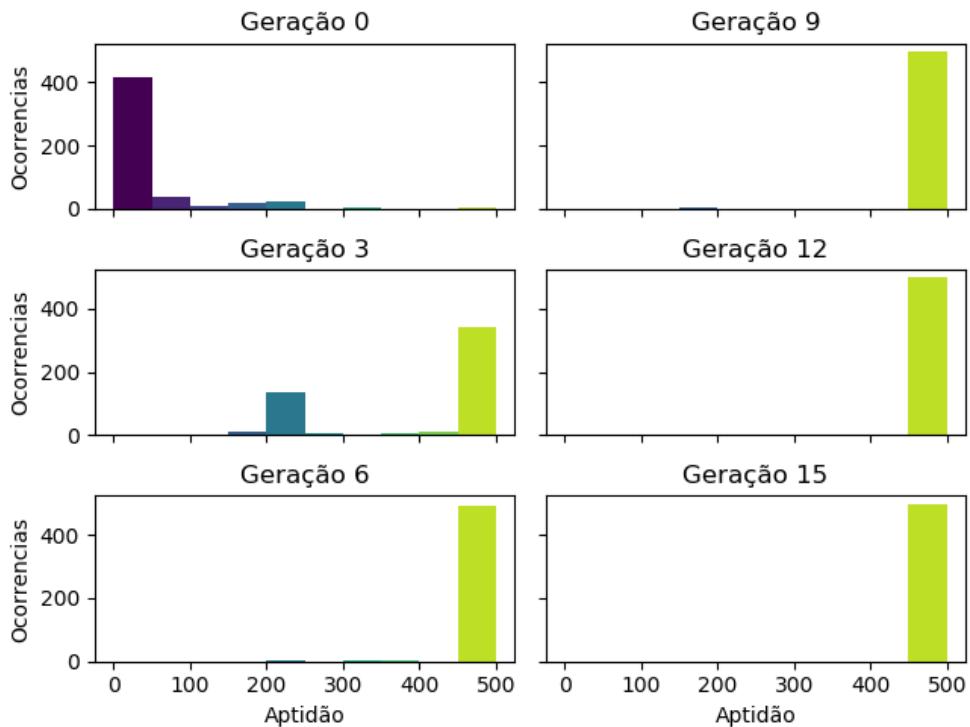


Figura 25: Número de indivíduos, em cada geração, que obtiveram cada faixa de aptidão. Verifica-se que já na 12º geração, a maior parte dos indivíduos possuem a aptidão máxima.

Conforme visto no capítulo 3.4, existe uma tendência de aumento do comprimento dos indivíduos ao longo do processo. O gráfico da Figura 26 mostra a média do comprimento de cada indivíduo em cada geração, assim como os valores mínimos e máximos de comprimento. Já que todas as estatísticas foram obtidas através de média de 10 execuções, o gráfico contém valores fracionários.

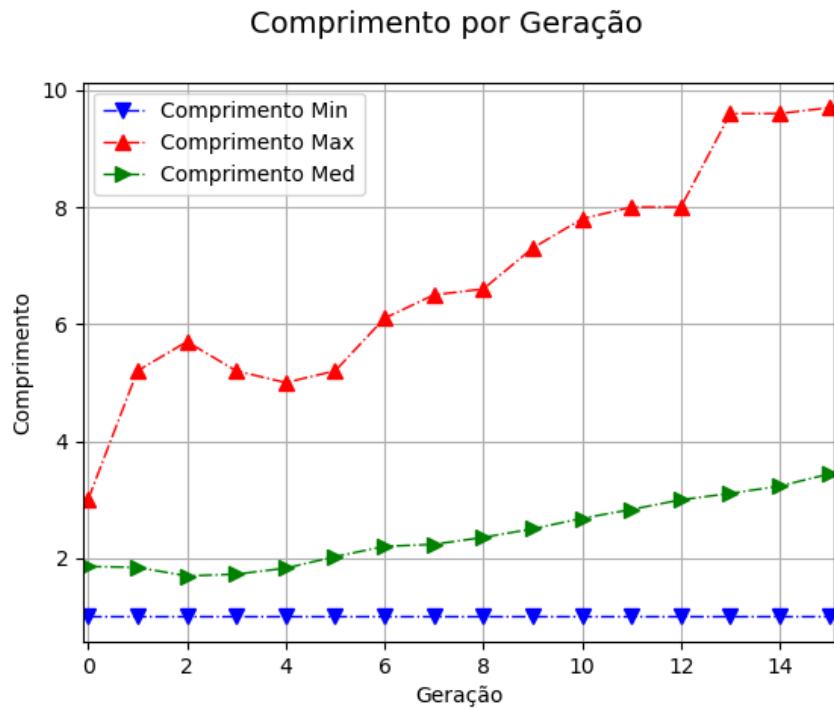


Figura 26: Comprimento dos indivíduos por geração.

A complexidade dos indivíduos ao longo do processo pode ser observada no gráfico da Figura 27.

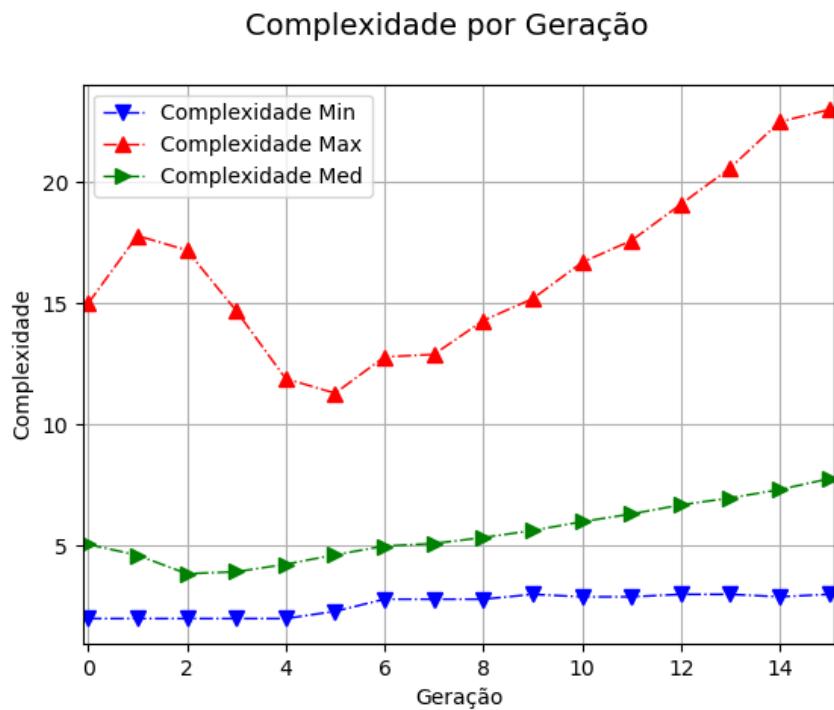


Figura 27: As medidas relacionadas à complexidade dos indivíduos em cada geração. Essa estatística tem uma alta correlação com a aridez das operações que ocorrem nos indivíduos.

Foi possível observar que alguns operadores e variáveis, pertencentes ao conjunto primitivo, são mais eficientes para a solução do problema, e tendem a aparecer com maior frequência à medida que a população se torna mais apta. Foi realizada a contagem dos operadores e variáveis de cada indivíduo da geração final, o resultado pode ser verificado no gráfico da Figura 28.

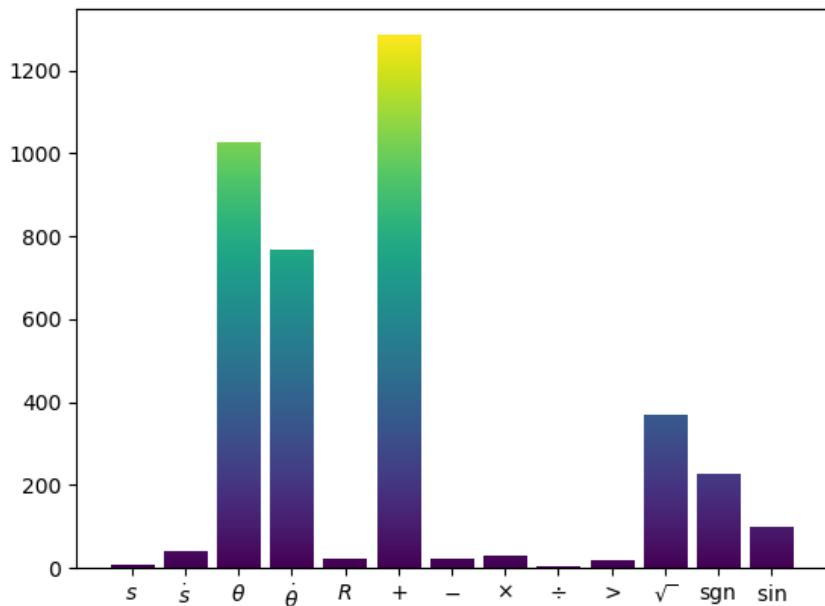


Figura 28: Ocorrência de cada operador e variável nos indivíduos da última geração.

Através do objeto *hall da fama*, é possível armazenar os indivíduos mais aptos que existiram na população ao longo de todo o processo de evolução. Esse objeto é atualizado a cada geração, de modo que o primeiro indivíduo possui a maior aptidão encontrada durante toda a execução do algoritmo evolucionário. Além disso, por ser um objeto de tamanho fixo, os indivíduos de gerações mais antigas possuem prioridade.

Na Figura 29, é possível ver o primeiro indivíduo do hall da fama. Já que na geração inicial alguns indivíduos obtiveram a aptidão máxima, a solução da Figura 29 tem prioridade sobre qualquer outra solução. O pequeno comprimento do indivíduo indica que, de fato, pertence às primeiras gerações.

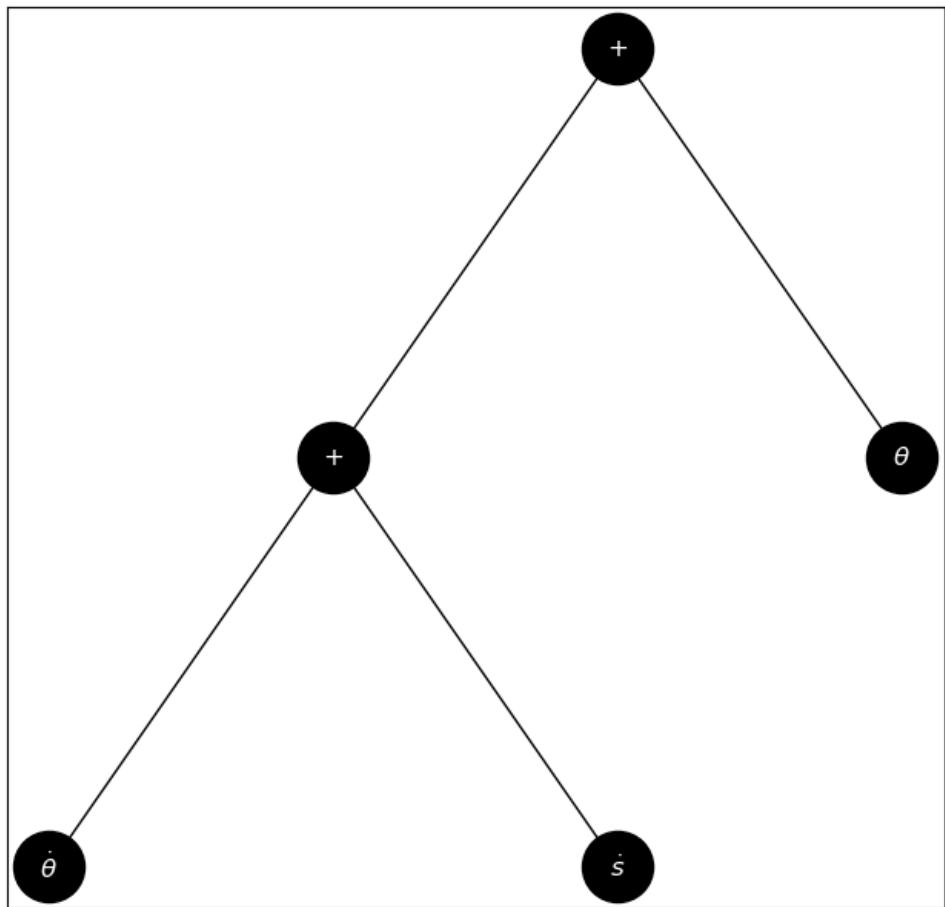


Figura 29: Primeiro indivíduo do hall da fama na primeira execução.

É possível perceber indivíduos de maior comprimento nas posições finais do hall da fama, como por exemplo, na Figura 30.

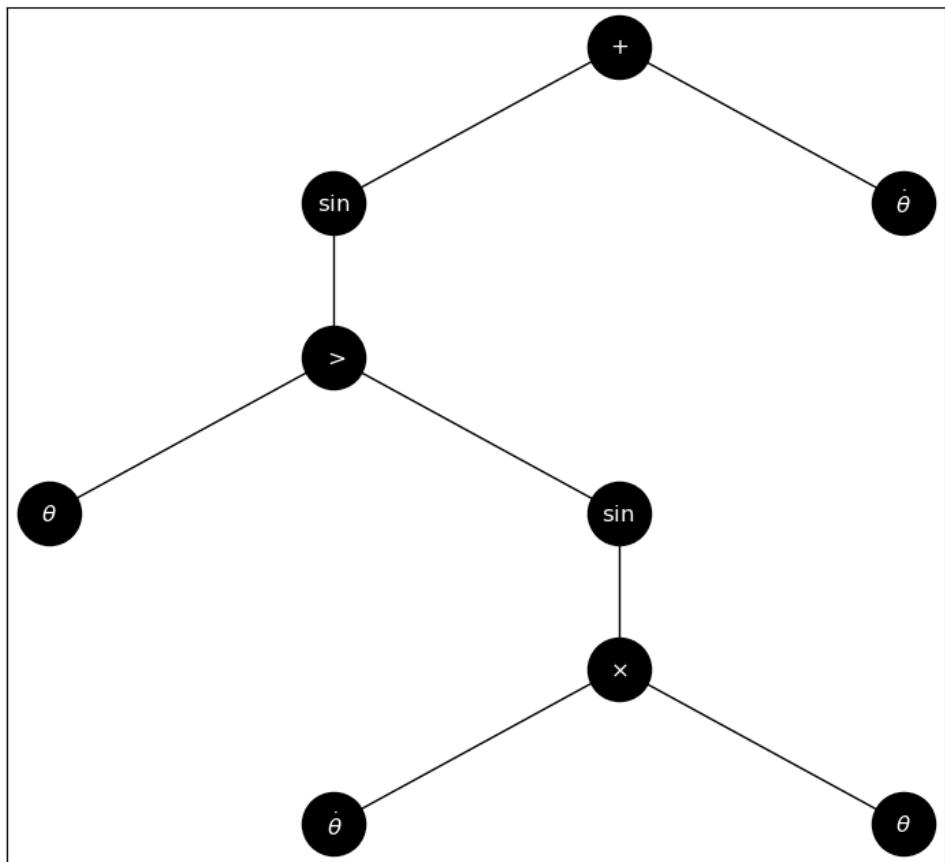


Figura 30: Vigésimo terceiro indivíduo do hall da fama, também na primeira execução do algoritmo.

A Figura 31 mostra os gráficos relacionados à avaliação do indivíduo da Figura 29, em um único episódio. O eixo *ação* indica o controle aplicado no ambiente a partir do *resultado* de saída ao avaliar a expressão matemática da árvore.

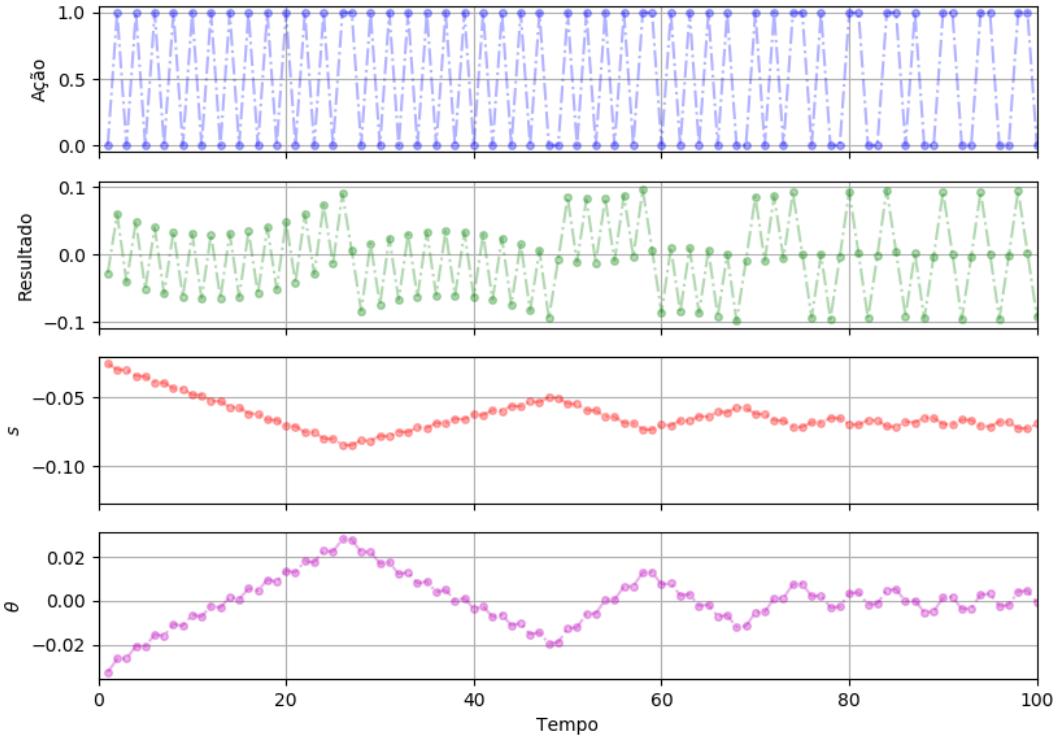


Figura 31: Controle do pêndulo pelo indivíduo da Figura 29. O pêndulo é levado rapidamente à um estado de baixa oscilação nos valores angulares.

Foi possível observar que os indivíduos aptos pertencentes ao hall da fama são capazes de manter um valor baixo de oscilação do ângulo ao redor do zero, entretanto, algumas soluções causavam a posição do carrinho a tender para um dos extremos, o que pode levar ao término antecipado do episódio.

Já que as aptidões dos indivíduos foram estimadas a partir de 10 simulações, é possível que as melhores soluções tenham sido beneficiadas por condições iniciais vantajosas. Não é possível, entretanto, aumentar o número de simulações sem que haja aumento considerável no tempo de execução do algoritmo.

Dessa forma, é interessante verificar a aptidão dos indivíduos do hall da fama, em um grande número de episódios. Portanto, encontraremos o indivíduo mais apto para qualquer condição inicial.

A situação inicial do ambiente é determinada de forma aleatória, dentro de uma faixa específica característica de cada problema na biblioteca Gym. As condições iniciais de cada ambiente podem ser observadas no apêndice B.

É proveitoso realizar, nesse momento, a comparação desses resultados com a abordagem proposta pelo algoritmo DQN, uma vez que a medida de aptidão é a mesma. O tempo médio de execução do algoritmo de PG foi 334s. O agente DQN será treinado, aproximadamente, pelo mesmo tempo. Em seguida, as recompensas médias acumuladas por episódio, em 100 simulações serão comparadas. Destaca-se que a avaliação da PG foi realizada pela média de recompensa acumulada pelos 10 primeiros membros do hall da fama.

Utilizando a biblioteca *stable-baselines* [22], o agente foi treinado por 334s (código no apêndice). O gráfico da Figura 32 mostra a recompensa obtida pelo agente ao longo do treinamento.

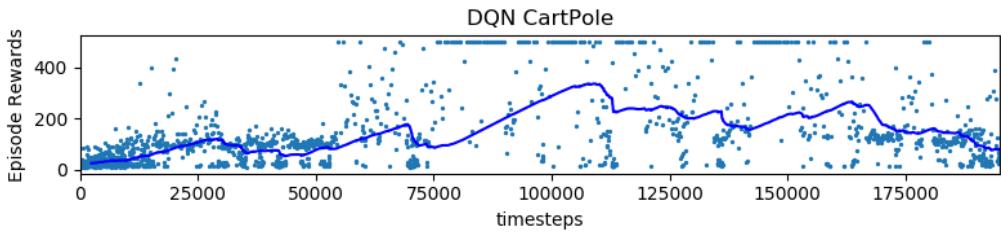


Figura 32: Recompensa média acumulada em função do número de passos de simulação. A linha azul representa a média móvel.

Na Figura 32 é possível notar a degradação da recompensa média acumulada a partir dos 110000 passos de simulação. Já que a rede neural tende a exibir esse comportamento, sem a devida otimização dos hiperparâmetros, o algoritmo foi executado novamente com o número de passos totais reduzido. O resultado pode ser verificado na Figura 33.

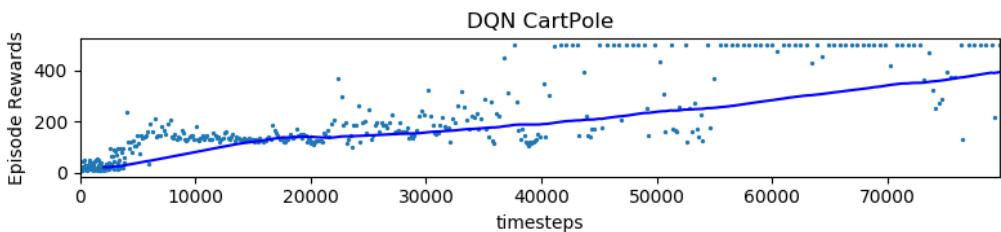


Figura 33: Recompensa média ao longo dos passos de simulação e a média móvel.

A Tabela 5 sumariza uma breve comparação entre o desempenho da programação genética com o algoritmo DQN, em termos de custos computacionais.

Tabela 5: Comparação entre a programação genética e DQN para o pêndulo invertido.

	PG	DQN
Desempenho	497	500
Tempo de execução (s)	334	280
Passos de simulação	16647338	80000
Número de episódios	65095	450 ¹

A Figura 34 mostra a atuação do agente DQN ao longo de um episódio.

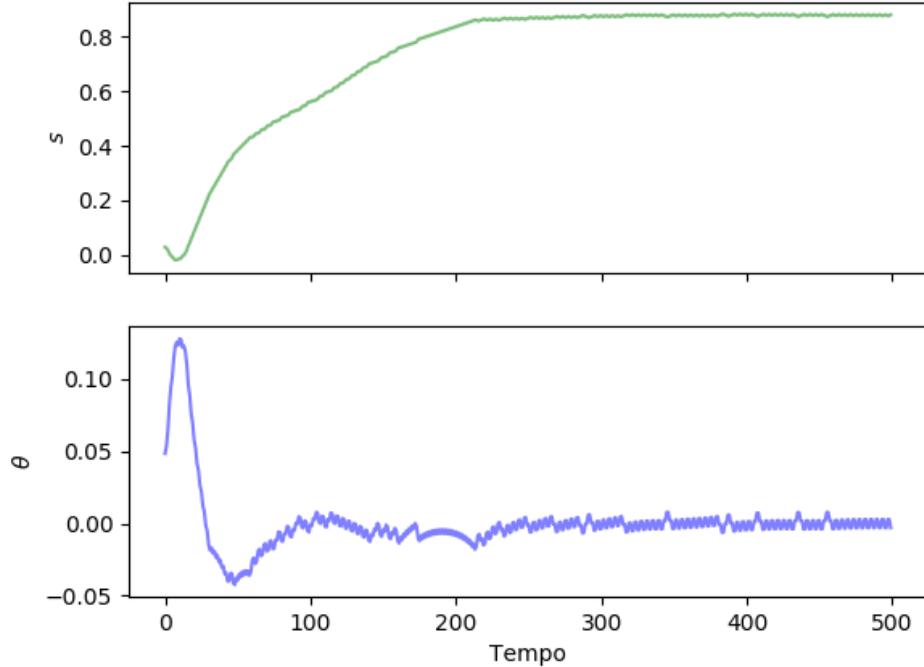


Figura 34: Posição do carrinho (topo) e ângulo do bastão em função do tempo.

Observamos, por fim, que as duas abordagens foram capazes de encontrar soluções satisfatórias para o problema. A seguir, serão abordados outros problemas que envolvem pêndulos e sua estabilização.

4.2 Pêndulo *Swing-up*

O próximo sistema simulado está disponível na biblioteca Gym, na seção *classic control*, direcionada à implementação de ambientes de simulações para problemas clássicos

¹Valor estimado.

de controle. Conforme a abordagem do problema anterior, serão introduzidos os critérios de término do episódio, as variáveis de estado observáveis e a implementação da recompensa. A formulação da função de aptidão, a partir da recompensa disponível também será abordada.

A dinâmica envolve um bastão com uma de suas extremidades fixadas, sendo possível a atuação do agente a partir da aplicação de um torque, em qualquer sentido, buscando a manutenção da extremidade livre na posição mais alta, ou seja, o pêndulo deve ser mantido verticalmente para cima. A Figura 35 ilustra o problema.

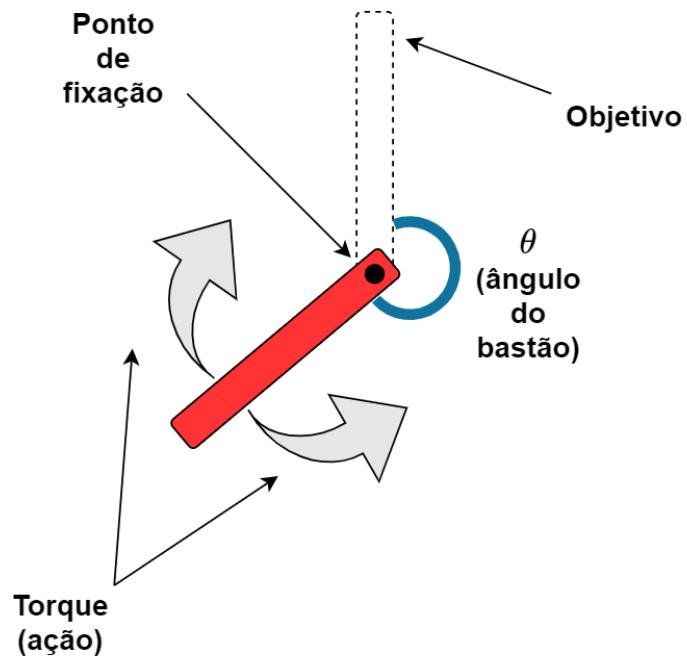


Figura 35: Pêndulo *Swing-up*.

A seguir serão enumerados alguns aspectos que tornam o problema mais complexo e outras diferenças fundamentais do ambiente, em comparação com o pêndulo invertido:

- O ponto de equilíbrio é estável.
- O torque aplicado em um único sentido, a partir da posição de repouso, não é capaz de alcançar o objetivo em um movimento contínuo (o bastão precisa adquirir momento para superar a força gravitacional).
- As ações possíveis se dão no espaço contínuo.
- As recompensas são implementadas a partir de uma função custo.

A Tabela 6 sumariza as variáveis de estado do sistema, que são fornecidas como uma *observação* após cada ação do agente. Já que o ponto de equilíbrio do sistema é estável, o único critério de término é o tempo de simulação. Com base na documentação da biblioteca Gym [5], o episódio termina após 200 passos de tempo.

Tabela 6: Variáveis de estado para o pêndulo swing-up.

Variável	Significado
$\cos(\theta)$	Cosseno do ângulo do bastão
$\sin(\theta)$	Seno do ângulo do bastão
$\dot{\theta}$	Velocidade angular do bastão

Seguindo a abordagem da seção 3.2, será criada uma função de aptidão para o indivíduo. A recompensa a cada instante de tempo é uma função custo, que penaliza desvios do objetivo.

$$r(t) = -[(\theta(t))^2 + 0.1(\dot{\theta}(t))^2 + 0.001(a(t))^2] \quad -\pi \leq \theta \leq \pi \\ -2 \leq a(t) \leq 2 \quad (8)$$

A variável $a(t)$ representa a ação (torque) do agente no instante de tempo t . Foi mencionado na seção 1 a possibilidade de projetar a aptidão de um indivíduo a partir de uma função custo, dessa forma, a aptidão será dada pela soma dos custos (implementados como recompensas negativas) ao longo de um episódio:

$$A(t) = r(t) = -[(\theta(t))^2 + 0.1(\dot{\theta}(t))^2 + 0.001(a(t))^2] \\ A_{tot}^{ep} = \sum_{t=0}^T A(t) = -\sum_{t=0}^T [(\theta(t))^2 + 0.1(\dot{\theta}(t))^2 + 0.001(a(t))^2] \quad T \leq 200 \quad (9)$$

$$\bar{A} = \frac{1}{nep} \sum_{ep=1}^{nep} A_{tot}^{ep} = -\frac{1}{nep} \sum_{ep=1}^{nep} \sum_{t=0}^T [(\theta(t))^2 + 0.1(\dot{\theta}(t))^2 + 0.001(a(t))^2]$$

De forma semelhante ao problema do pêndulo invertido, o objetivo será maximizar a aptidão média na Equação 9.

Buscando demonstrar a robustez do método, poucas mudanças foram realizadas nos hiperparâmetros da Tabela 4, mais especificamente, foram alterados: o número de entradas, comprimentos de inicialização e o comprimento máximo de mutação.

Tabela 7: Parâmetros da programação genética aplicada ao pêndulo swing-up.

Parâmetro	Valor
Tamanho da População	500
Probabilidade de Cruzamento	0.75
Probabilidade de Mutação	0.05
Número de Gerações	15
Número de Entradas	3
Faixa para Constante Efêmera	(-1.0, 1.0)
Número de Simulações	10
Tamanho do Campeonato de Aptidão	6
Tamanho do Campeonato de Comprimento	1.2
Operações	add, sub, mul, div, sr, cr, cos, gt, sgn
Comprimento Mínimo e Máximo de Inicialização	(2, 5)
Comprimento Máximo de Mutação	7
Limite de Comprimento dos Indivíduos	17

É interessante destacar que o problema da seção 4.1 utilizava ações discretas. Um mapeamento simples de números resultantes da avaliação da árvore para ações foi utilizado: números positivos produziam uma força para a direita no carrinho, enquanto que números negativos geravam uma força de sentido contrário.

Neste problema, o espaço de ações é contínuo, portanto é necessário apenas garantir que o resultado matemático da compilação de um indivíduo obedeça os limites de torque definidos na equação 8. Isto pode ser concebido pela utilização da função *clip*, cujo comportamento é demonstrado na Figura 36.

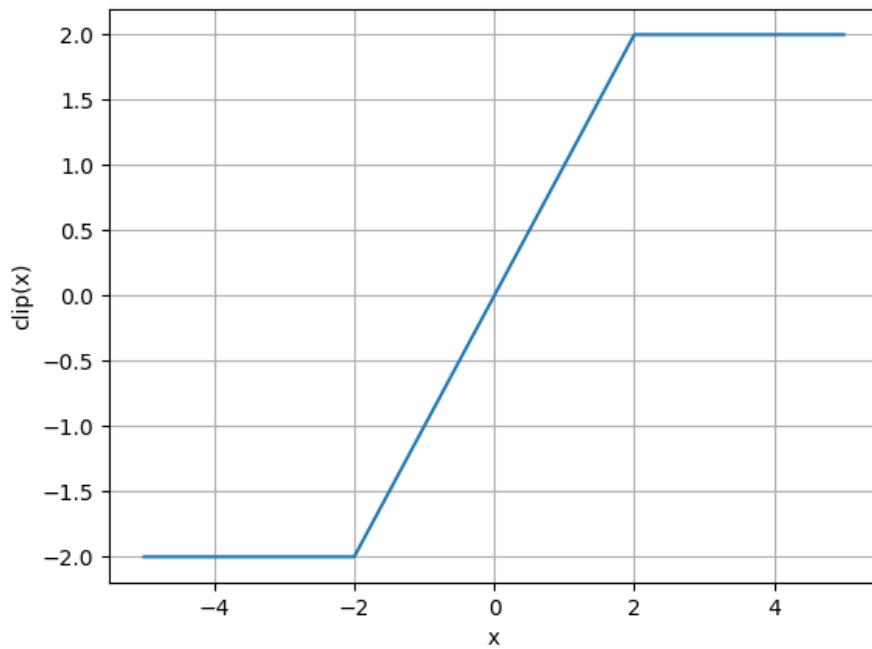


Figura 36: Função *clip*.

Essa função funcionará como *wrapper*, isto é, um mapeamento do resultado de avaliação da árvore para uma ação dentro do espaço de ações possíveis.

Novamente, o algoritmo foi executado 10 vezes e a média das estatísticas relevantes foram obtidas. A começar pela aptidão ao longo das gerações, onde é possível perceber que a busca inicial da primeira geração, através da inicialização, não é capaz de obter um resultado satisfatório como no problema anterior. Porém, as próximas gerações encontram uma solução eficaz para o pêndulo, conforme pode ser visto nas Figuras 37 e 38.

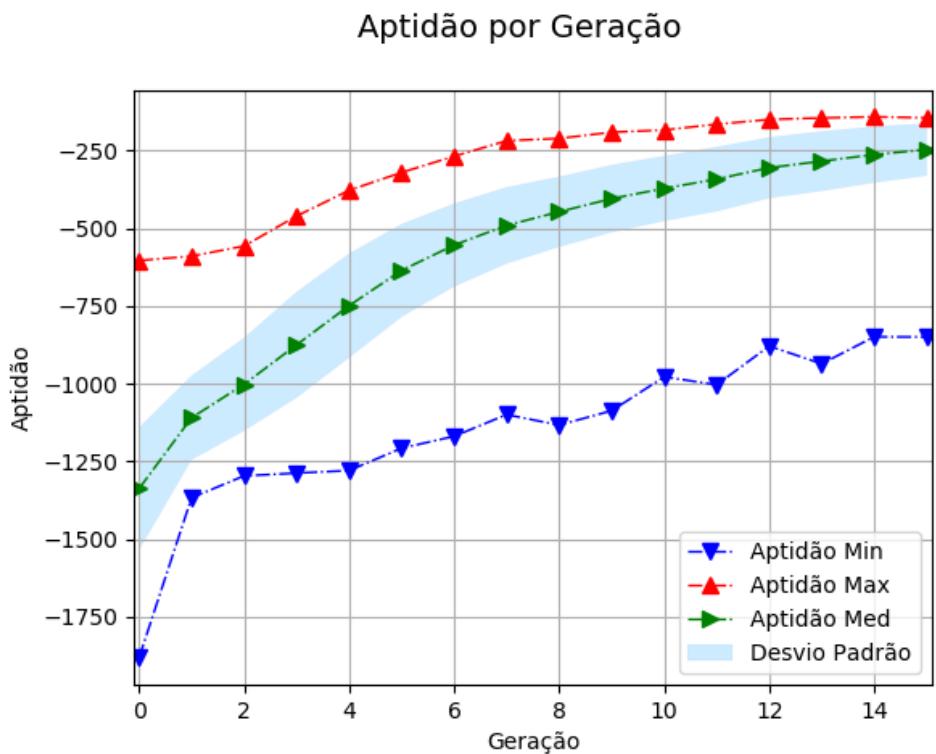


Figura 37: Aptidão dos indivíduos ao longo das gerações.

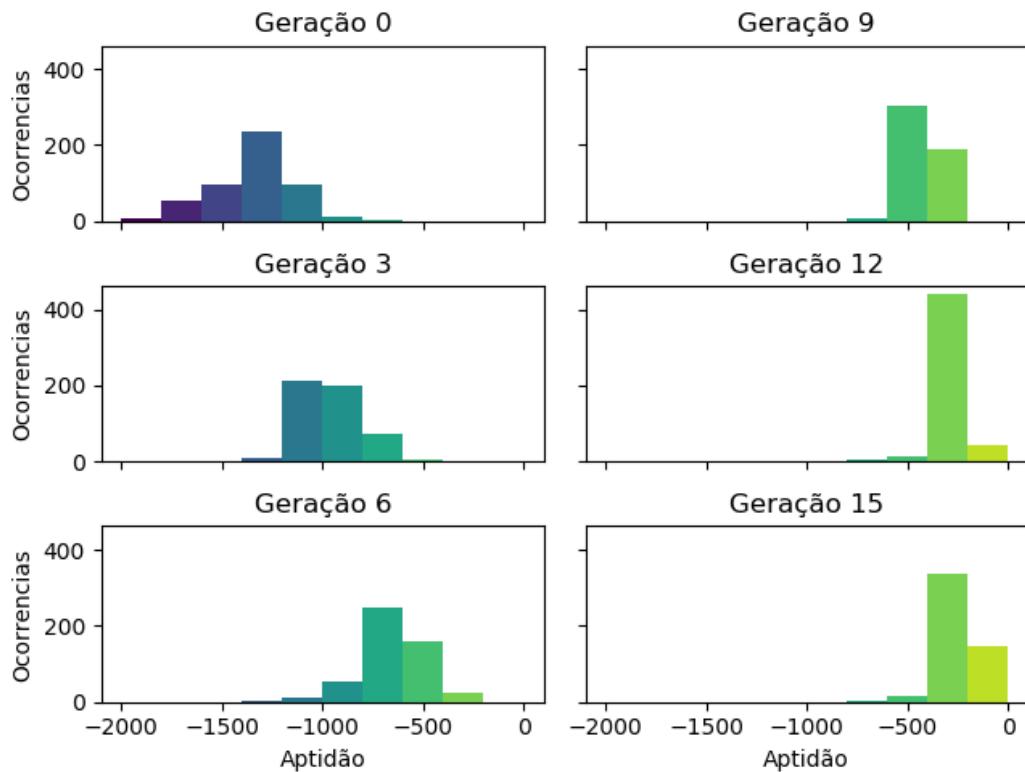


Figura 38: Histograma da aptidão dos indivíduos em cada geração.

O comprimento e a complexidade dos indivíduos da população, ao longo das

gerações, podem ser vistos nas Figuras 39 e 40, respectivamente.

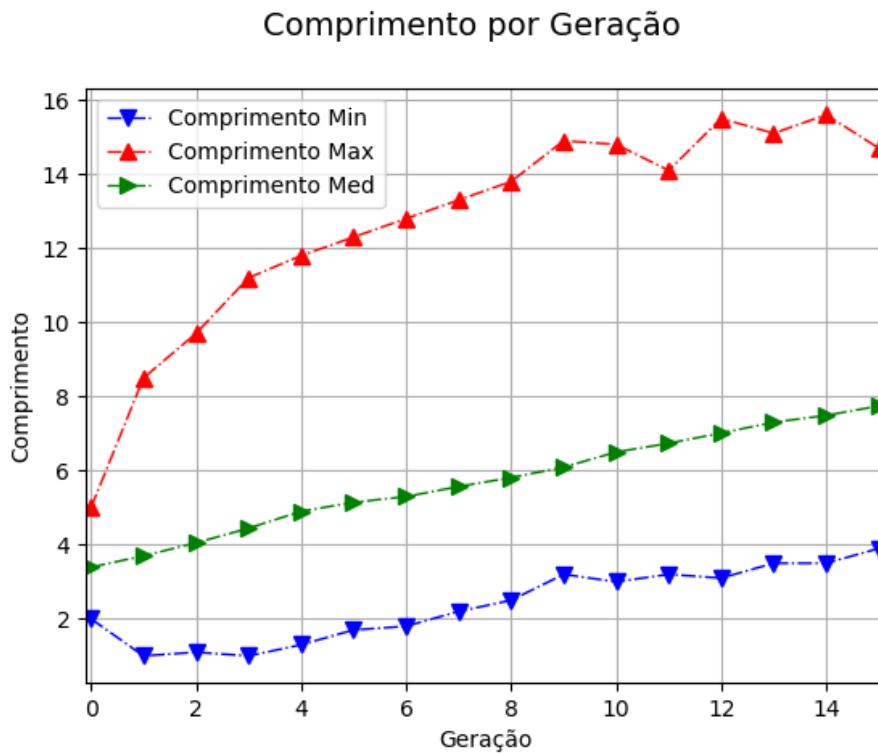


Figura 39: Comprimento dos indivíduos ao longo das gerações.

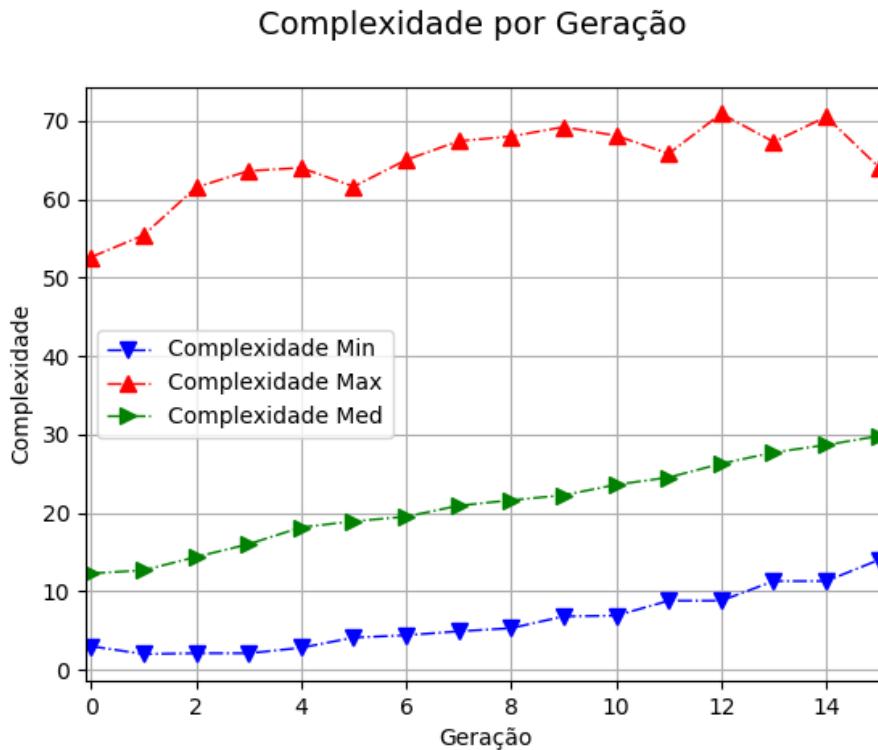


Figura 40: Medida da complexidade dos indivíduos em cada geração.

É possível perceber o aumento dessas duas medidas em relação ao pêndulo invertido, devido também à mudanças na inicialização e no comprimento máximo da mutação.

A Figura 41 mostra o histograma de ocorrência dos operadores e variáveis terminais na população da última geração.

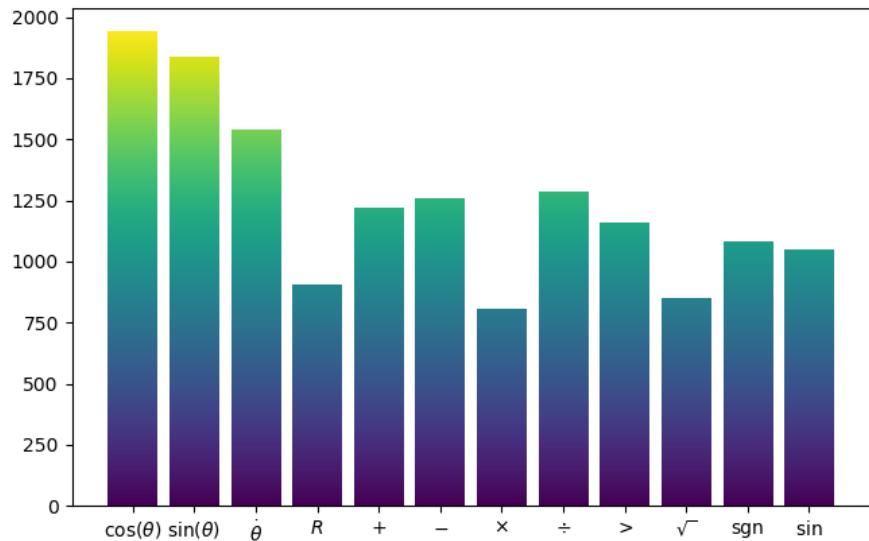


Figura 41: Histograma de operadores e variáveis terminais na última geração.

O indivíduo de maior aptidão observado, na primeira execução, é mostrado na Figura 42.

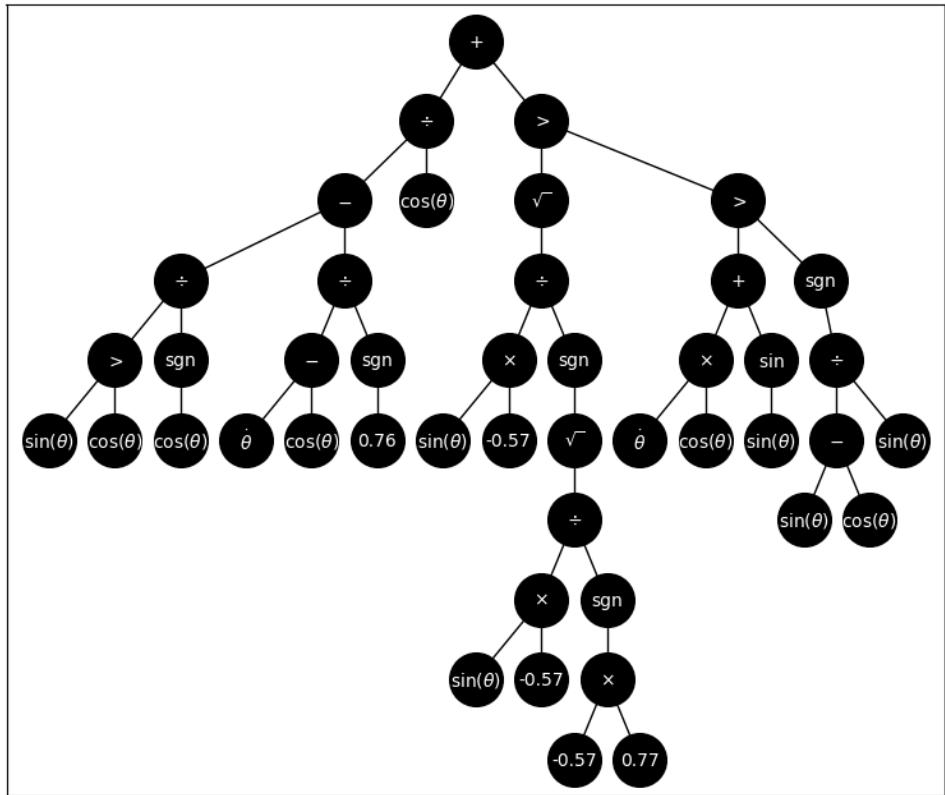


Figura 42: Primeiro integrante do hall da fama.

O segundo integrante do hall da fama, na primeira execução, é mostrado na Figura 43.

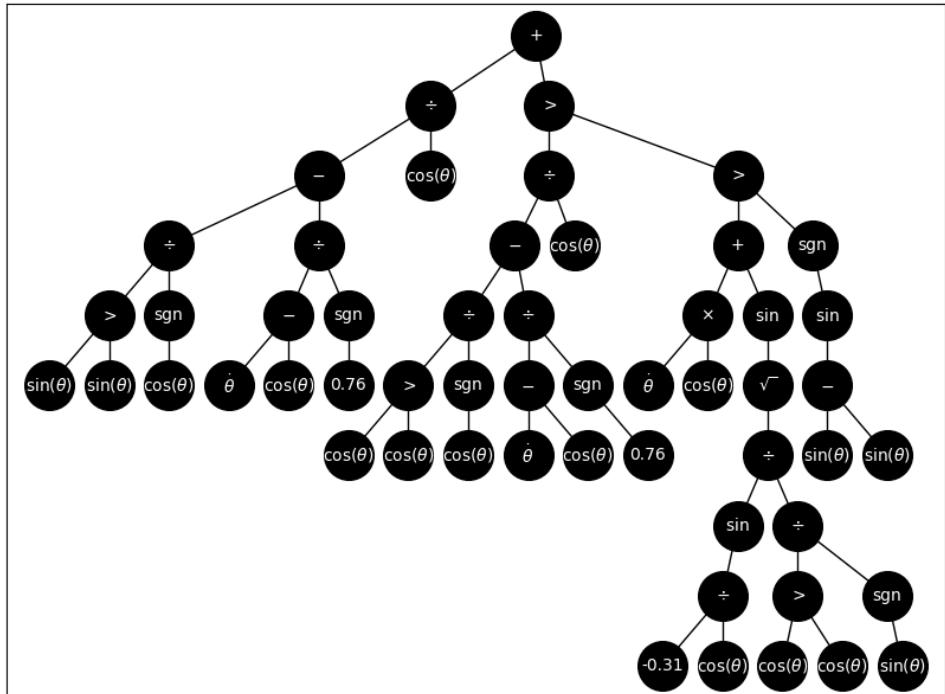


Figura 43: Segundo integrante do hall da fama.

A Figura 44 mostra as variáveis relevantes ao avaliar o indivíduo da Figura 42 em um episódio.

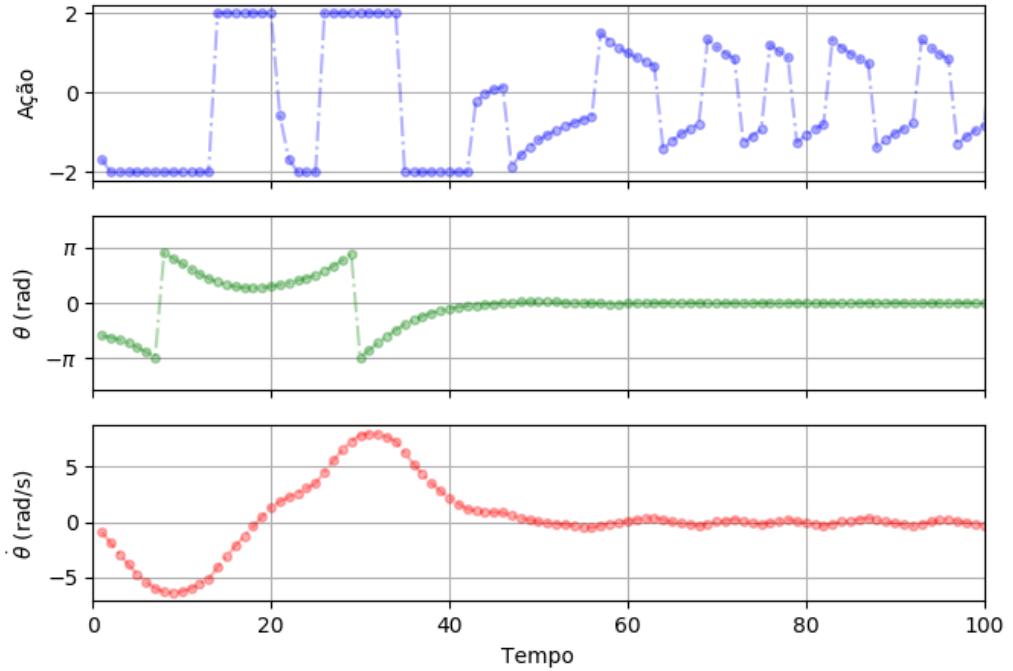


Figura 44: Controle exercido pelo indivíduo da Figura 42.

As transições bruscas no ângulo θ indicam a passagem da extremidade livre do pêndulo no ponto mais baixo, onde o custo adquiri seu valor máximo. Após esse momento, o ângulo oscila em torno do zero de forma estável.

Utilizando a mesma metodologia da seção 4.1, será feita uma avaliação em 100 episódios dos indivíduos do hall da fama. O maior valor de aptidão obtido será comparado com a recompensa acumulada de um agente treinado com o algoritmo DDPG, que pode ser visto como uma extensão do algoritmo DQN para ambientes com ações no espaço contínuo. Novamente, a medida de desempenho de um agente ou indivíduo é a função de recompensa disponibilizada pelo ambiente de simulação. Com isso, é possível realizar uma comparação direta entre as duas abordagens.

A Figura 45 mostra o agente DDPG sendo treinado em 100000 passos de tempo.

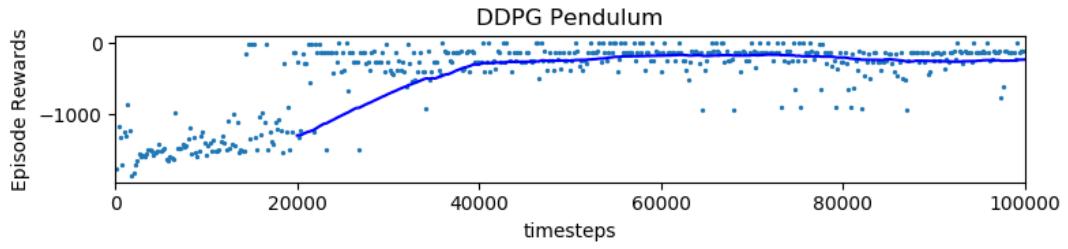


Figura 45: Evolução da recompensa acumulada para o agente DDPG no pêndulo swing-up.

A atuação do agente em um episódio pode ser vista na Figura 46.

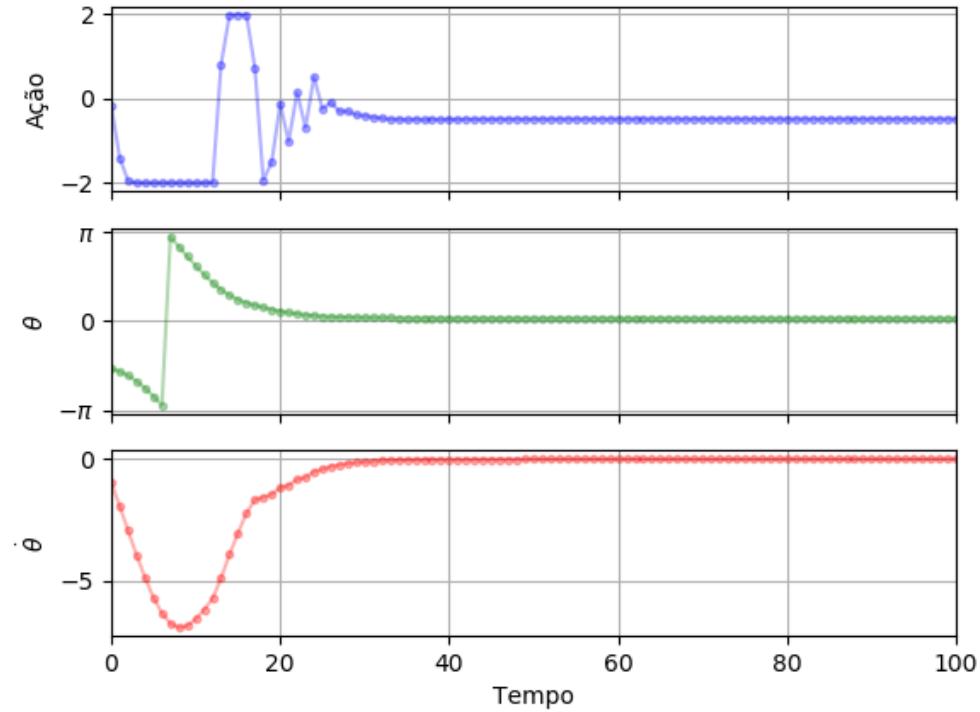


Figura 46: Dinâmica do pêndulo swing-up sob ação do agente DDPG.

Tabela 8: Comparação entre a programação genética e DDPG para o pêndulo swing-up.

	PG	DDPG
Desempenho	-230	-253
Tempo de execução (s)	1649	194
Passos de simulação	13034400	100000
Número de episódios	65172	500

É possível notar a partir da Tabela 8 que as duas abordagens são capazes de resolver o problema de controle proposto.

4.3 Pêndulo Duplo Invertido

Este problema é similar ao pêndulo invertido, pois a estabilização do bastão envolve a movimentação do objeto (que contém o ponto de fixação) em uma trilha. A diferença reside na existência de um outro bastão, fixado na extremidade antes livre, aumentando consideravelmente a complexidade do problema. As Figuras 47 e 48 mostram o sistema proposto, em um ambiente tridimensional.

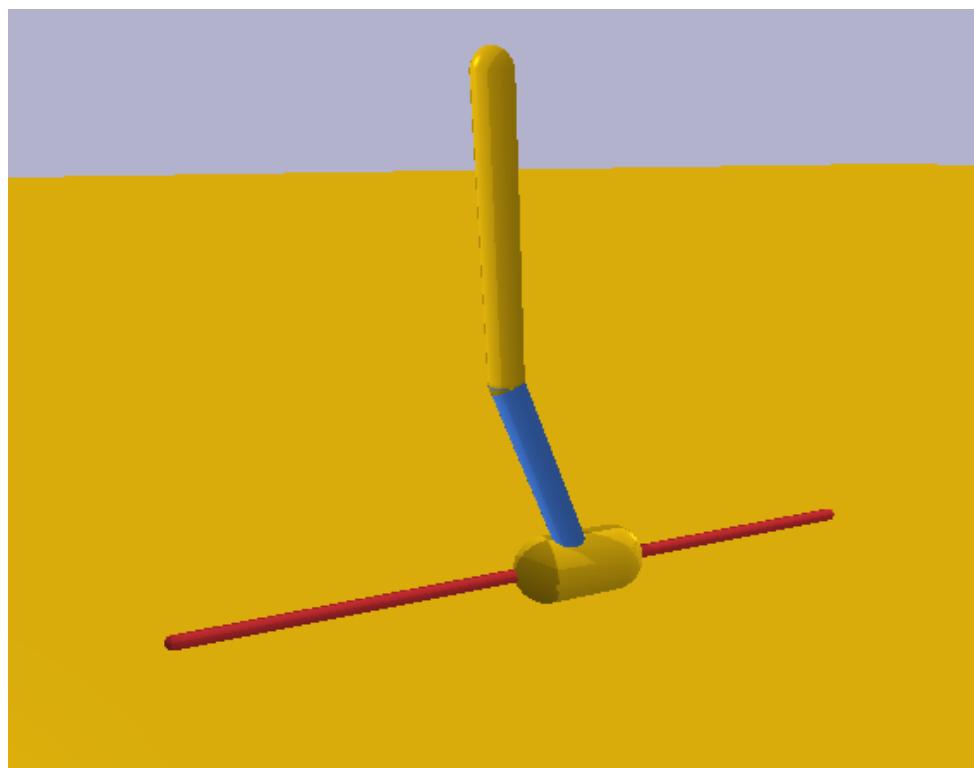


Figura 47: Renderização 3D do sistema pêndulo duplo invertido.

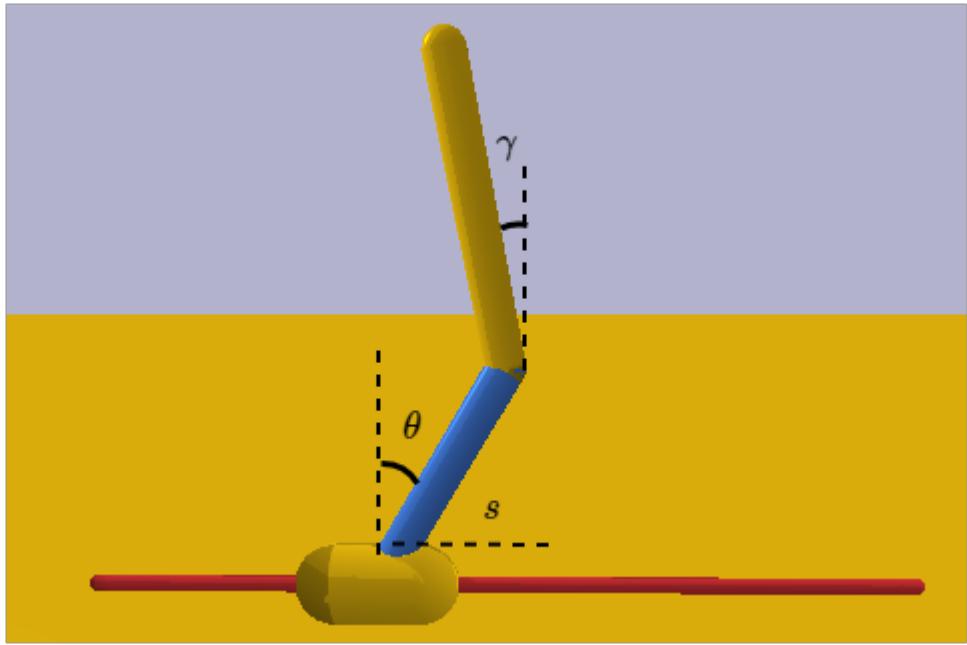


Figura 48: Ângulos θ , γ e a posição do carrinho (s).

A observação do sistema é composta por 11 variáveis:

Tabela 9: Variáveis de estado para o pêndulo duplo invertido.

Variável	Significado
s	Posição do carrinho
$\sin(\theta)$	Seno do ângulo do bastão inferior
$\sin(\gamma)$	Seno do ângulo do bastão superior
$\cos(\theta)$	Cosseno do ângulo do bastão inferior
$\cos(\gamma)$	Cosseno do ângulo do bastão superior
\dot{s}	Velocidade do carrinho
$\dot{\theta}$	Velocidade angular do bastão inferior
$\dot{\gamma}$	Velocidade angular do bastão superior
$f_r(s)$	Força de restrição em função da posição
$f_r(\theta)$	Força de restrição em função de θ
$f_r(\gamma)$	Força de restrição em função de γ

Conforme a abordagem realizada até o momento, todas as variáveis da Tabela 9 farão parte do conjunto primitivo. Os outros parâmetros foram mantidos iguais ao do problema anterior (Tabela 7).

A função de recompensa do ambiente de simulação, a cada instante de tempo, é dada na equação (10).

$$r(t) = 10 - d_p(t) - v_p(t)$$

$$d_p(t) = 0.01(s(t))^2 + (y(t) - 2)^2 \quad (10)$$

$$v_p(t) = 0.001(\dot{\theta}(t))^2 + 0.005(\dot{\gamma}(t))^2$$

Na equação (10), $d_p(t)$ e $v_p(t)$ são penalidades dadas em função da distância e das velocidades angulares dos bastões, respectivamente. A variável y representa a altura da extremidade livre do bastão superior. Além de auxiliar o cálculo do custo, a quantidade y também auxilia na verificação do término do episódio. Mais especificamente, a simulação encerra após 1000 passos de tempo ou quando a variável y assume valores menores ou iguais a 1.

Naturalmente, a recompensa da equação (10) será utilizada para o cálculo da aptidão de um indivíduo. Utilizando a mesma formulação das equações em (9):

$$A(t) = r(t) = 10 - d_p(t) - v_p(t)$$

$$A_{tot}^{ep} = \sum_{t=0}^T A(t) = - \sum_{t=0}^T [10 - d_p(t) - v_p(t)] \quad T \leq 1000 \quad (11)$$

$$\bar{A} = \frac{1}{nep} \sum_{ep=1}^{nep} A_{tot}^{ep} = -\frac{1}{nep} \sum_{ep=1}^{nep} \sum_{t=0}^T [10 - d_p(t) - v_p(t)]$$

Como o espaço de ações é contínuo, será utilizada a função clip, da Figura 36, como wrapper para o resultado compilado dos indivíduos. Entretanto, os valores extremos serão -1 e 1 .

As Figuras 49 a 52 mostram os resultados obtidos, através de 10 execuções do algoritmo.

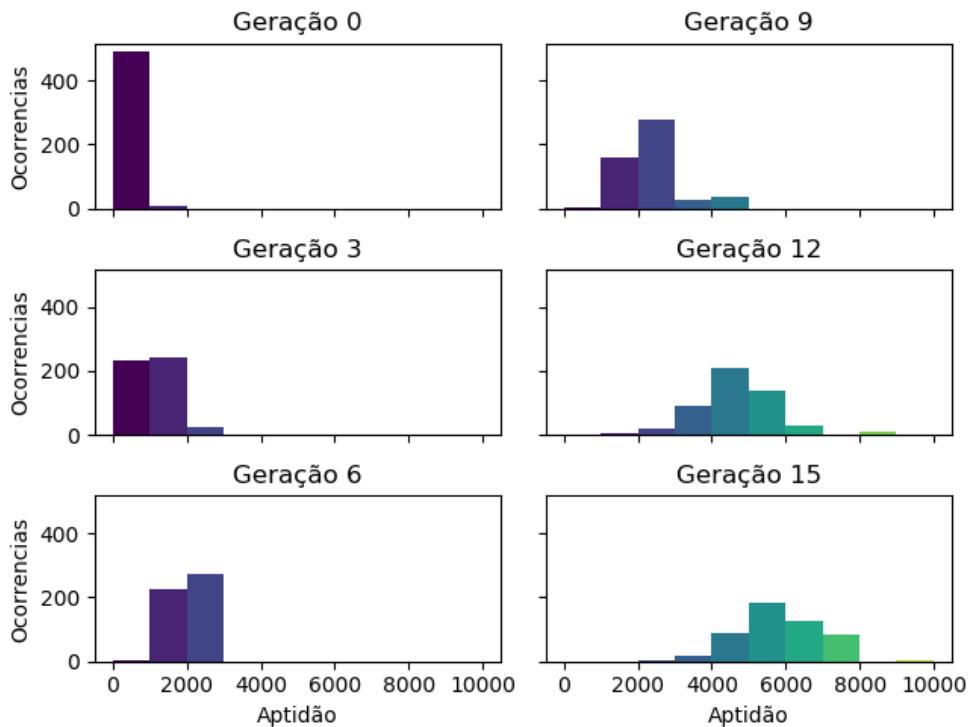


Figura 49: Histograma da aptidão dos indivíduos, em algumas gerações, para o pêndulo duplo invertido.

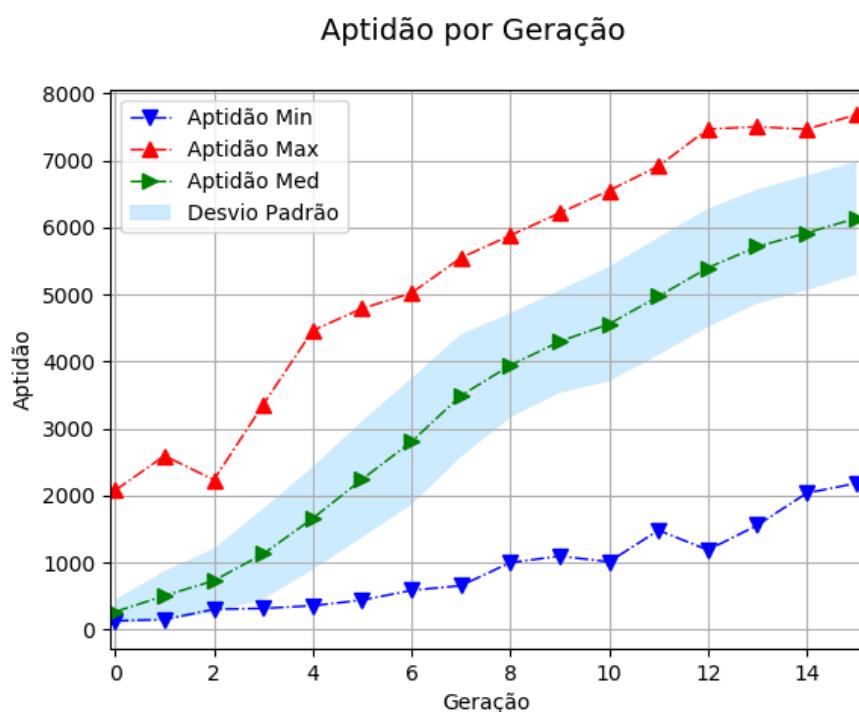


Figura 50: Medidas de aptidão da população, em cada geração.

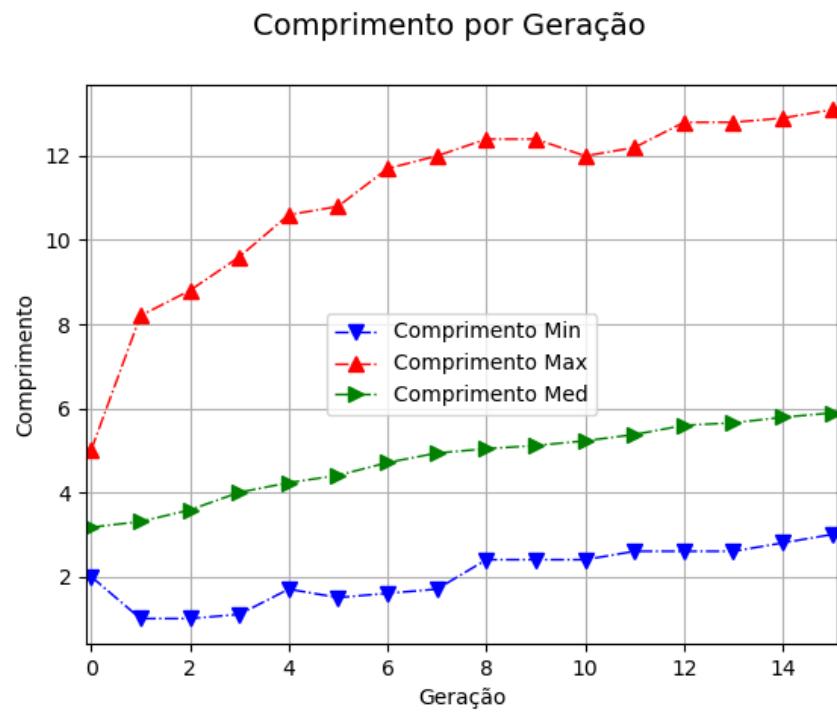


Figura 51: Medidas de comprimento da população ao longo das gerações.

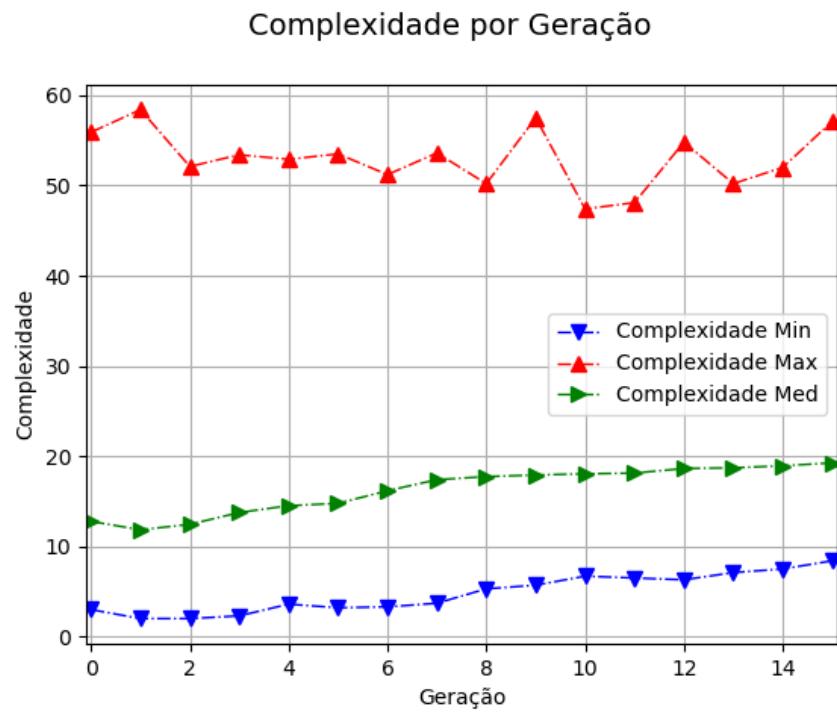


Figura 52: Medidas de complexidade dos indivíduos em função das gerações.

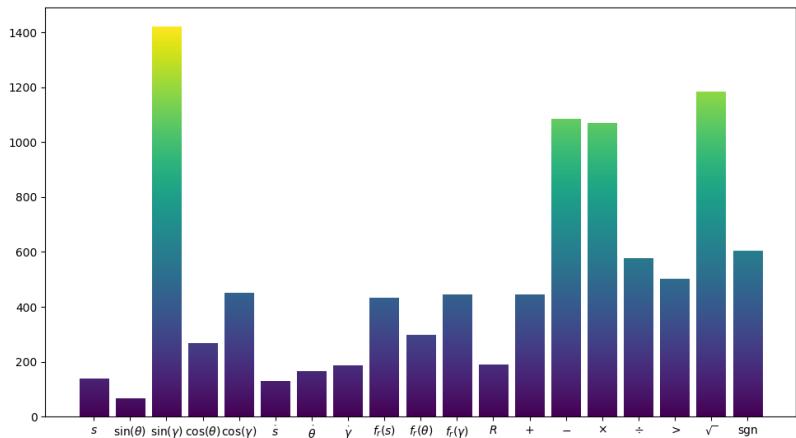


Figura 53: Média de ocorrências dos operadores e variáveis terminais na última geração.

Para cada indivíduo do hall da fama, conforme feito anteriormente, 100 simulações foram realizadas, com o intuito de verificar a solução mais apta e generalista. O indivíduo da Figura 54 obteve uma aptidão média de 7794.

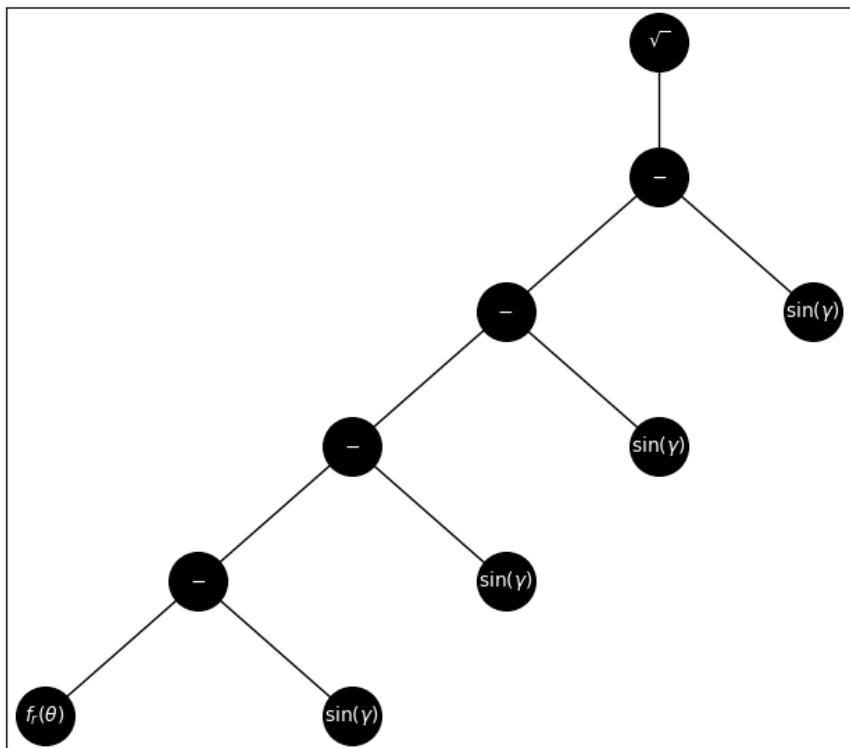


Figura 54: Indivíduo mais apto da primeira execução do algoritmo.

Os gráficos da Figura 55 mostram os ângulos e a posição do carrinho, quando a solução da Figura 54 atua.

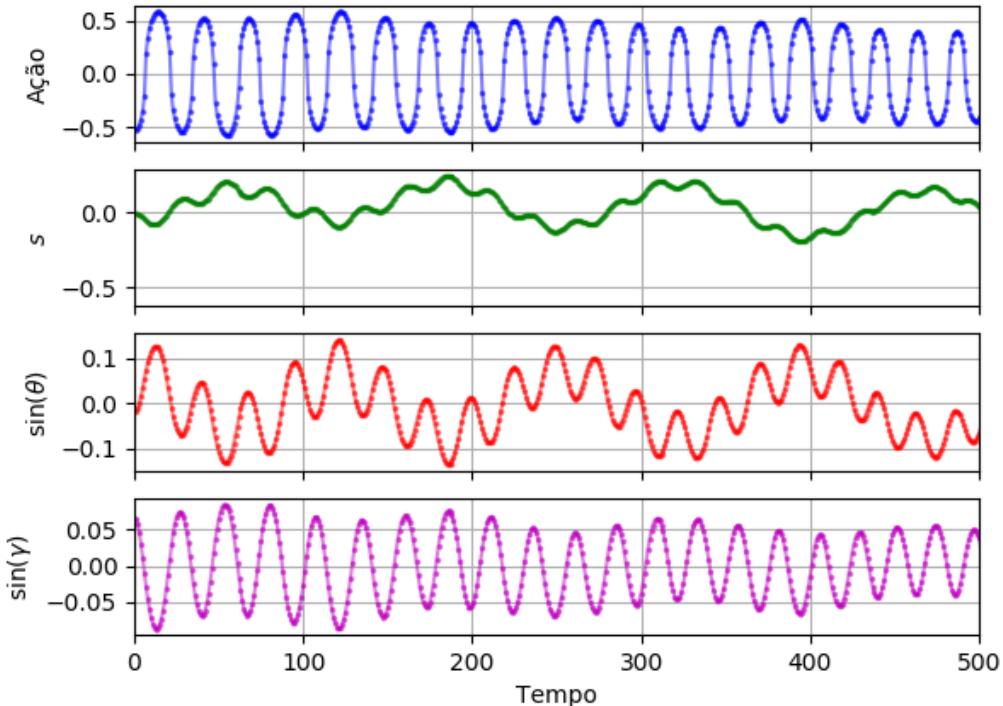


Figura 55: Avaliação do melhor indivíduo observado na primeira execução.

O gráfico da Figura ?? mostra um agente sendo treinado com o algoritmo DDPG, por 400000 passos de tempo. De forma similar à Figura 55, um episódio foi simulado sob ação do agente até 500 instantes de tempo. O resultado desta avaliação, em termos das variáveis de estado, pode ser observado na Figura 56.

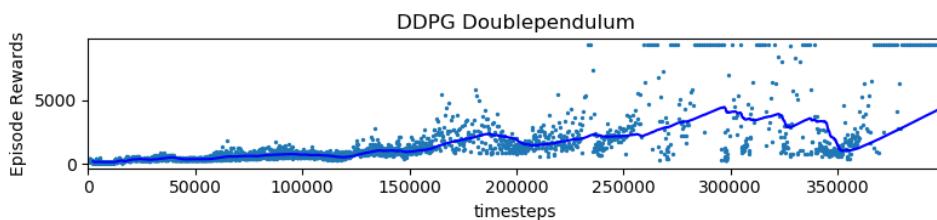


Figura 56: Evolução da recompensa acumulada do agente DDPG. A linha azul representa a média móvel.

A recompensa média acumulada pelo agente, em 100 episódios, pode ser vista na Tabela 10, junto à outras estatísticas relevantes.

Tabela 10: Comparação entre PG e DDPG para o pêndulo duplo invertido.

	PG	DDPG
Desempenho	7794	9019
Tempo de execução (s)	2406	1230
Passos de simulação	10392643	400000
Número de episódios	64989	1230

4.4 Carro na Ladeira

O problema do carro na ladeira foi descrito inicialmente por Andrew Moore [23] e se tornou um dos problemas mais importantes na teoria de aprendizagem por reforço. O problema consiste em um carro cujo objetivo é chegar ao topo de uma montanha, com uma determinada velocidade. Entretanto, o motor não é forte o suficiente a ponto de permitir que o carro chegue no objetivo em um único movimento. É necessário, portanto, que o carro ganhe energia se movimentando na direção contrária. A Figura 57 ilustra o problema.

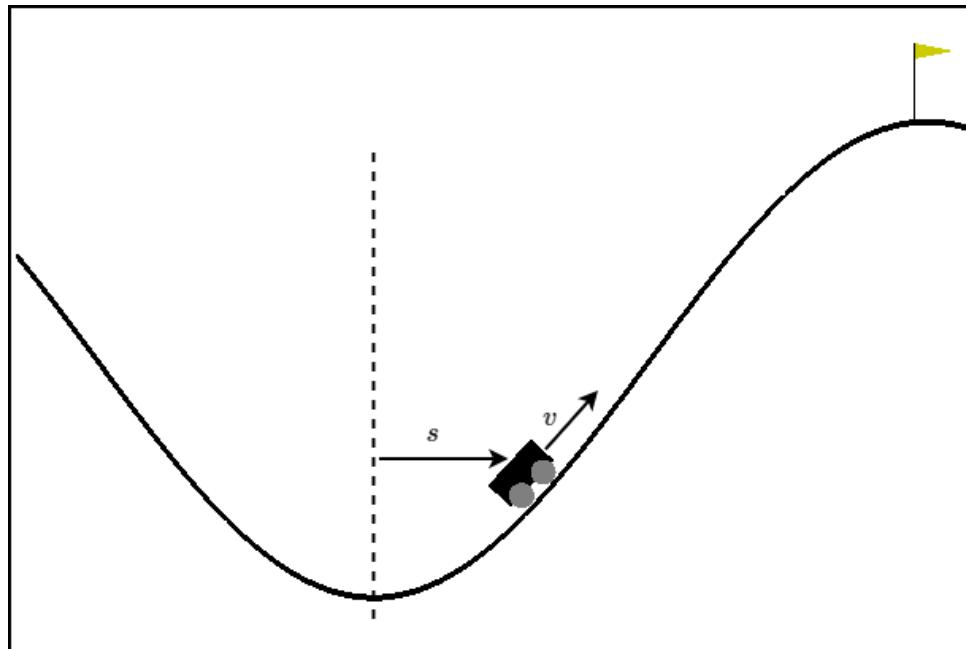


Figura 57: Problema do carro na ladeira.

A implementação do ambiente na biblioteca permite definir qual a velocidade que o carro precisa ter ao chegar no topo da montanha. A configuração padrão, que será

utilizada, não impõe um valor mínimo de velocidade. Dessa forma, basta que o carro chegue à posição indicada pela bandeira, na Figura 57.

A observação do ambiente é simples e consiste apenas na posição e velocidade do carrinho, conforme a Tabela 11 indica. O agente recebe uma penalização unitária (equação 12) a cada instante de tempo, buscando indivíduos que cheguem ao objetivo no menor tempo possível.

Tabela 11: Variáveis de estado para o problema do carro na ladeira.

Variável	Significado
s	Posição do carro.
\dot{s}	Velocidade do carro

$$r(t) = -1 \quad (12)$$

O espaço de ações é discreto e indica uma força que atua no carro:

$$a(t) = \begin{cases} 0 & \rightarrow \text{Força aplicada à esquerda} \\ 1 & \rightarrow \text{Não aplica força} \\ 2 & \rightarrow \text{Força aplicada à direita} \end{cases} \quad (13)$$

Na seção 4.1 vimos como transformar um número no espaço contínuo (resultado da expressão matemática de um indivíduo, em um instante de tempo) em uma ação mapeada em inteiros discretos. Com duas ações discretas, bastava mapear números positivos à uma ação e negativos à outra. No problema atual, existem três ações possíveis, portanto, será definida uma faixa de valores para a ação que não aplica força (representada pelo inteiro 1).

Utilizando a mesma abordagem dos problemas anteriores, a aptidão de um indivíduo será a média da recompensa acumulada em um determinado número de episódios.

$$\bar{A} = \frac{1}{nep} \sum_{ep=1}^{nep} \sum_{t=0}^T r(t) \quad (14)$$

É essencial que alguns indivíduos sejam capazes de chegar ao topo da montanha, caso contrário, os indivíduos tenderão a apresentar um comportamento estático, ao evitar

a penalização associada à tomada de ações. É interessante, portanto, utilizar uma taxa maior de mutação, visando auxiliar a busca global por soluções. A Tabela 12 mostra os parâmetros utilizados na execução da PG.

Tabela 12: Parâmetros utilizados para o problema do carro na ladeira.

Parâmetro	Valor
Tamanho da População	500
Probabilidade de Cruzamento	0.70
Probabilidade de Mutação	0.10
Número de Gerações	15
Número de Entradas	4
Faixa para Constante Efêmera	(-1.0, 1.0)
Número de Simulações	10
Tamanho do Campeonato de Aptidão	6
Tamanho do Campeonato de Comprimento	1.2
Operações	add, sub, mul, div, sr, cr, cos, gt, sgn
Comprimento Mínimo e Máximo de Inicialização	(2, 5)
Comprimento Máximo de Mutação	9
Limite de Comprimento dos Indivíduos	17

Os resultados obtidos são mostrados nas Figuras ?? a ??

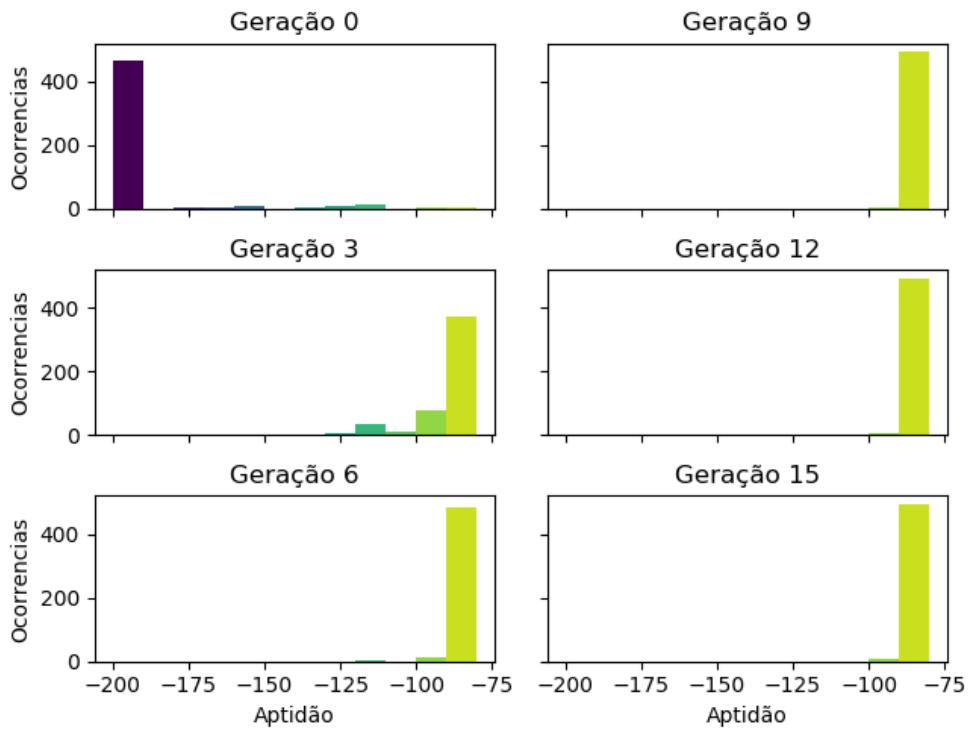


Figura 58: Histograma da aptidão dos indivíduos, em algumas gerações, para o pêndulo duplo invertido.

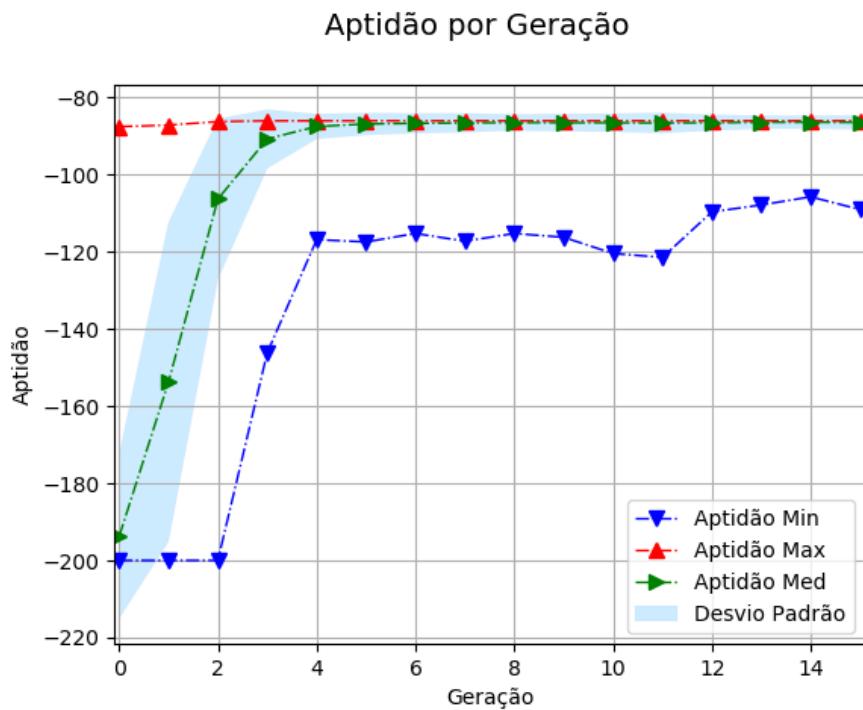


Figura 59: Aptidão da população, em cada geração.

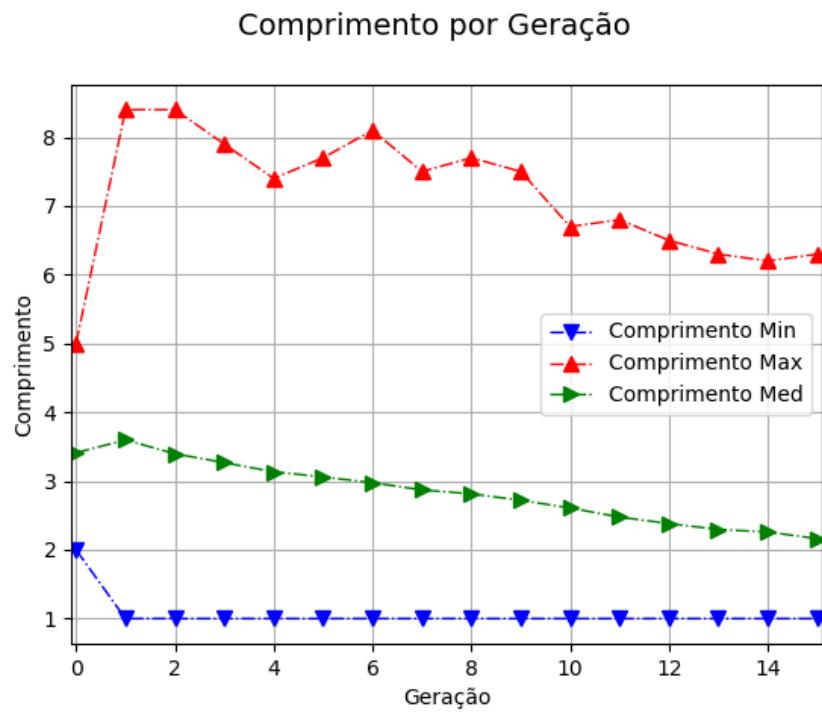


Figura 60: Comprimento médio dos indivíduos da população ao longo das gerações.

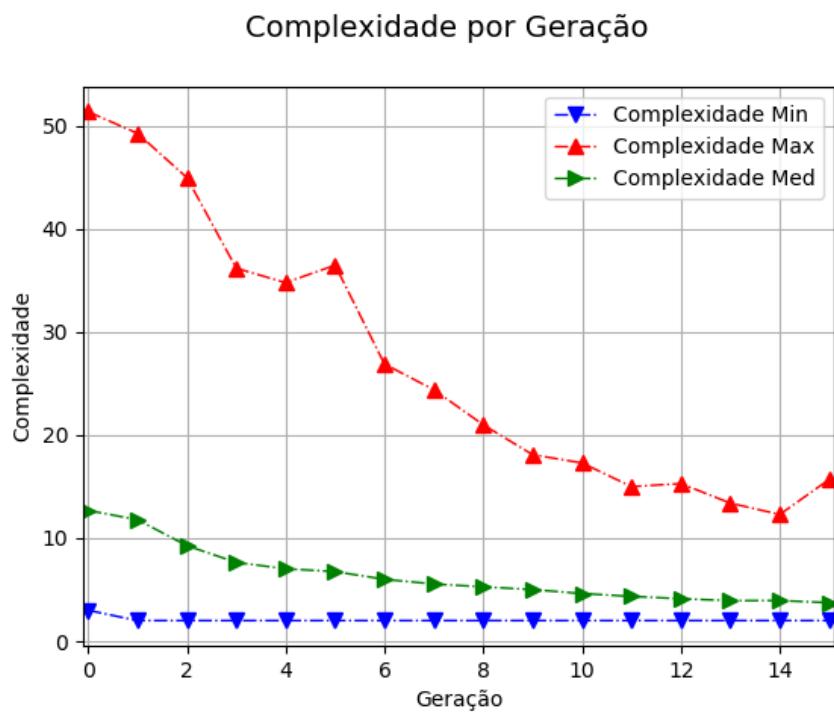


Figura 61: Complexidade média dos indivíduos da população ao longo das gerações.

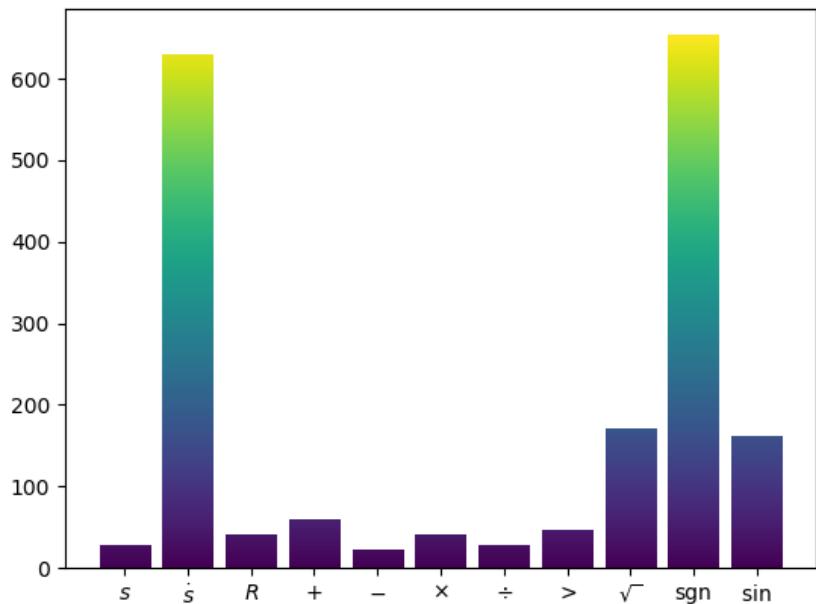


Figura 62: Ocorrências dos operadores e variáveis terminais na última geração.

O melhor indivíduo foi obtido ao simular a atuação de todas as soluções do hall da fama, em 100 episódios. A Figura ?? mostra a composição do indivíduo.

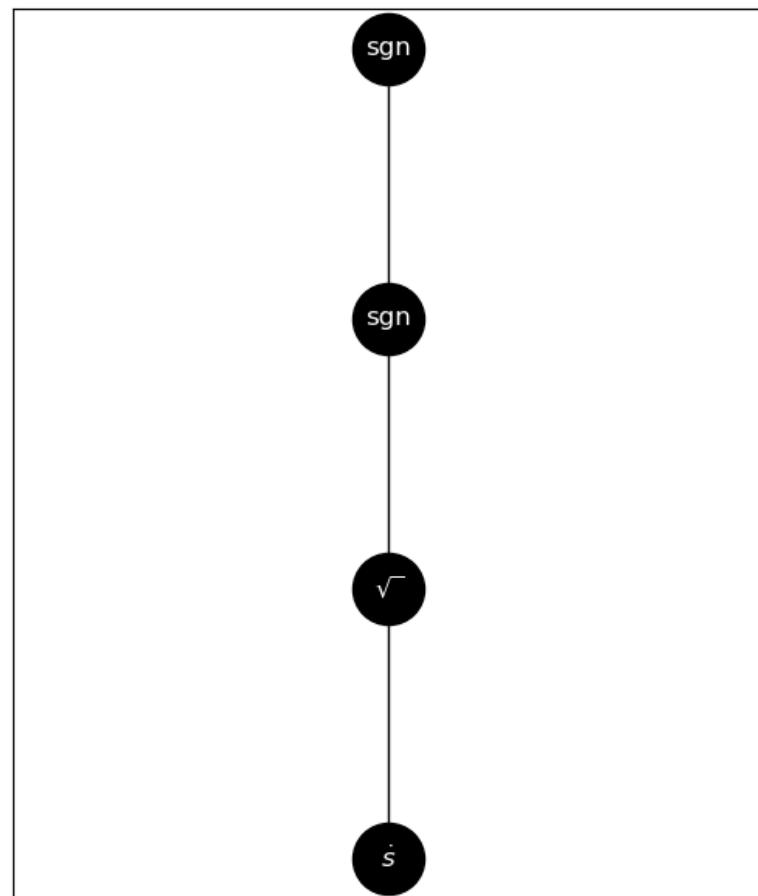


Figura 63: Melhor indivíduo da primeira execução.

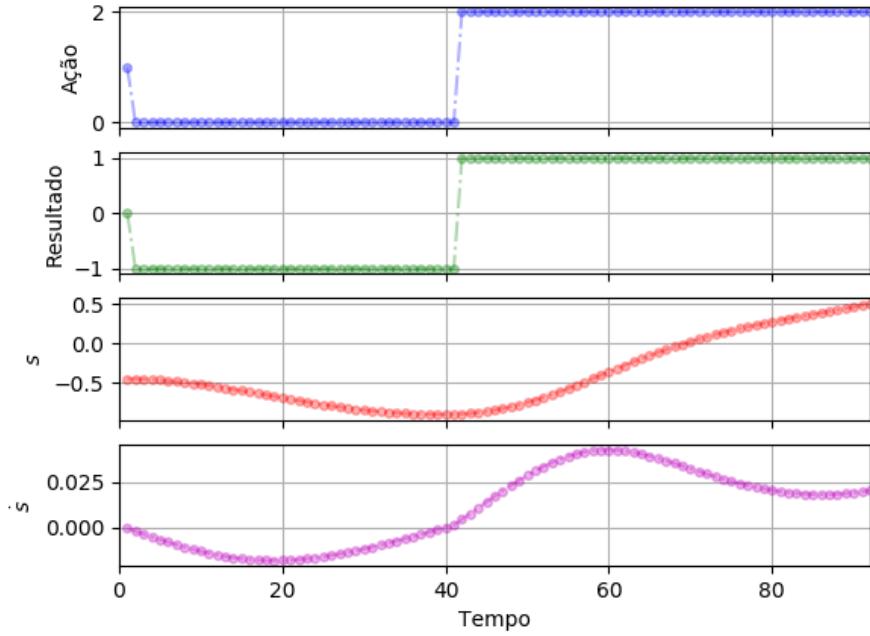


Figura 64: Atuação do indivíduo da Figura 63 em um episódio aleatório.

Como o espaço de ações é discreto, um agente foi treinado utilizando o algoritmo DQN. A Figura ?? mostra a recompensa acumulada pelo agente DQN ao longo do tempo de execução. A Tabela 13 sumariza os resultados encontrados com a abordagem evolucionária comparado à aprendizagem por reforço.

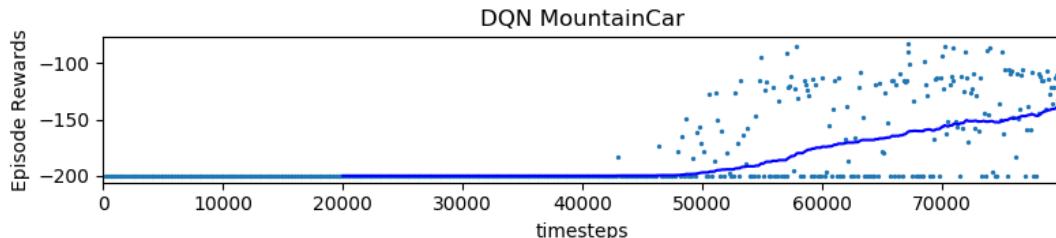


Figura 65: Agente DQN e a recompensa acumulada ao longo dos passos de tempo.

Tabela 13: Comparação entre PG e DQN para o problema do carro na ladeira.

	PG	DQN
Desempenho	-108	-140
Tempo de execução (s)	79	311
Passos de simulação	966211	80000
Número de episódios	6514	450 ²

É possível perceber, em termos de tempo de execução, que a PG se mostrou bem superior ao algoritmo DQN.

²Valor aproximado.

CONCLUSÃO

Ao longo deste trabalho, foi feita uma verificação dos conceitos teóricos da programação genética, e como ela pode ser utilizada para a resolução de problemas de controle. A programação genética busca otimizar programas de computador através de um processo inspirado na evolução biológica. As principais etapas desse processo envolvem a criação de soluções potenciais para o problema, seguido da avaliação de cada indivíduo, seleção dos candidatos mais aptos e aplicação subsequente dos operadores genéticos.

Vimos que as operações de cruzamento, mutação e replicação, possuem diferentes propósitos na busca por soluções. E a busca é guiada, em grande parte, pela função de aptidão; isto é, buscamos estabelecer um critério de avaliação que oriente o processo evolucionário de acordo com o objetivo. Muitas vezes, o critério mais prático e eficaz é a utilização de uma função custo, que calcula a diferença entre um estado de referência e uma situação atual influenciada pelo agente.

Os algoritmos de aprendizagem por reforço utilizam uma função de recompensa, que fornece *feedback* ao agente à cada instante de tempo. Muitas vezes essa função de recompensa é similar à uma função custo, logo, pode ser utilizada como critério aptidão para os indivíduos da programação genética. A partir deste conceito, este trabalho buscou realizar simulações e atribuir aptidões com base em uma função de recompensa. Dessa forma, foi possível aproveitar ambientes de simulações disponíveis em uma outra área de estudo, no caso, aprendizagem por reforço.

Python é uma das linguagens de programação mais populares em aplicações relacionadas à inteligência artificial. Com isso, diversas bibliotecas estão disponíveis, o que permite o rápido desenvolvimento de protótipos e testes de novas abordagens. A biblioteca Gym [5] fornece diversos ambientes de simulação, com uma interface relativamente simples, o que permite a comparação de algoritmos de aprendizagem de máquinas. Apesar de ser direcionada à algoritmos de aprendizagem por reforço, vimos que é possível utilizar algoritmos de aprendizagem evolucionária. A implementação dos agentes foi feita a partir da biblioteca DEAP, que permite a implementação de todas as etapas do ciclo evolucionário artificial, além de agrupar diversas ferramentas para a obtenção de estatísticas.

A integração dessas duas bibliotecas possibilitou ainda a comparação qualitativa da programação genética com algoritmos de aprendizagem por reforço.

Foi possível notar que o problema do pêndulo invertido foi solucionado de forma relativamente simples, já que na primeira geração alguns indivíduos obtiveram a aptidão máxima em 10 episódios. Entretanto, esta medida é uma aproximação do desempenho real do indivíduo, uma vez que algumas condições iniciais podem não ser resolvidas corretamente. Porém, o tempo de execução diminui consideravelmente com um baixo número de simulações para cada indivíduo.

Ao longo das gerações, a média de aptidão da população cresceu, conforme o esperado. Na décima quinta geração, a maior parte dos indivíduos possuía a aptidão máxima, indicando que o pêndulo permaneceu equilibrado durante 500 instantes de tempo. Na última geração, muitos indivíduos apresentaram o operador de soma e a variável terminal que indicava o ângulo do pêndulo. Isto indica que esses elementos foram eficazes para a resolução do problema abordado. A tendência de aumento do comprimento dos indivíduos ao longo das gerações também pode ser observada, indicando a importância de exercer o controle sobre esse processo. Em termos de desempenho e tempo de execução, os resultados obtidos através da programação genética foram similares aos obtidos com o algoritmo DQN.

Para o segundo problema abordado, ficou evidente a possível complexidade das soluções que surgem a partir do processo evolucionário, com alguns indivíduos apresentando mais de 70 nós. Por exemplo, o indivíduo da Figura 42 apresenta 41 nós, representando uma função matemática capaz de equilibrar um sistema de alta complexidade. Isso demonstra a capacidade da programação genética em aproximar funções utilizando apenas o conceito da evolução biológica.

De certa forma, os indivíduos são sistemas que respondem à entradas, produzindo uma saída (ou sinal de controle), com isso, se assemelham muito à redes neurais, utilizadas de forma extensiva nos algoritmos DQN e DDPG. Umas das principais distinções entre as abordagens de aprendizagem por reforço e a programação genética reside na realização da otimização. Frequentemente, redes neurais são otimizadas a partir da propagação reversa do gradiente de uma função custo, em relação aos pesos da rede neural, enquanto métodos evolucionários são otimizações livres de gradiente (*gradient free optimization*).

No ambiente do pêndulo duplo, o algoritmo DDPG apresentou-se superior. Entretanto, vale notar que não foram feitas mudanças significativas nos parâmetros da programação genética. É possível que algumas alterações, como por exemplo, nas probabili-

dade de ocorrência dos operadores genéticos, tenham um efeito positivo no desempenho do algoritmo. Os hiperparâmetros do algoritmo DDPG não foram alterados à partir da implementação utilizada [22], com exceção do coeficiente de ruído das ações, um elemento que se mostrou importante para garantir o balanço entre exploração e convergência da rede neural.

Curiosamente, o melhor indivíduo da primeira execução da PG no pêndulo duplo possui uma estrutura bastante simples e, basicamente, verifica a velocidade do carrinho, utilizando-a para exercer um controle que varia entre os valores -1 e 1. Muitos dos problemas que podem ser resolvidos por buscas exaustivas no espaço de soluções costumam ser facilmente solucionados pela busca direcionada provida pela PG.

O problema do carro na ladeira evidenciou a dificuldade do algoritmo de aprendizagem por reforço, quando a função de recompensa não é muito informativa. Basicamente, o sinal de recompensa é estável, até que o objetivo final seja alcançado. Dessa forma, o aprendizado é demorado e a programação genética se mostrou superior em tempo de execução e desempenho.

Referências

- [1] KOZA, J. R.; KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*. [S.l.]: MIT press, 1992.
- [2] JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015.
- [3] MARSLAND, S. *Machine learning: an algorithmic perspective*. [S.l.]: Chapman and Hall/CRC, 2014.
- [4] KOZA, J. R. *Genetic Programming*. 1997.
- [5] OpenAI Gym. <https://gym.openai.com/>. Acessado: 18/07/2019.
- [6] DEAP Distributed Evolutionary Algorithms in Python. <https://github.com/deap/deap>. Acessado: 18/07/2019.
- [7] MILLER, J. F.; HARDING, S. L. Cartesian genetic programming. In: ACM. *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. [S.l.], 2008. p. 2701–2726.
- [8] DIAS, D. M. *Síntese Automática de Programas para Microcontroladores Digitais por Programação Genética*. Tese (Doutorado). Disponível em: <<https://doi.org/10.17771/pucrio.acad.6666>>.
- [9] DURIEZ, T.; BRUNTON, S. L.; NOACK, B. R. *Machine Learning Control-Taming Nonlinear Dynamics and Turbulence*. [S.l.]: Springer, 2017.
- [10] BOUBAKER, O. The inverted pendulum benchmark in nonlinear control theory: a survey. *International Journal of Advanced Robotic Systems*, SAGE Publications Sage UK: London, England, v. 10, n. 5, p. 233, 2013.
- [11] WANG, J.-J. Simulation studies of inverted pendulum based on pid controllers. *Simulation Modelling Practice and Theory*, Elsevier, v. 19, n. 1, p. 440–449, 2011.

- [12] BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, IEEE, n. 5, p. 834–846, 1983.
- [13] OPENAI Gym - Cart Pole Environment. https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py. Acessado: 23/07/2019.
- [14] SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.
- [15] DEAP - Documentation. <https://deap.readthedocs.io/en/master/>. Acessado: 27/07/2019.
- [16] POLI, R. et al. *A field guide to genetic programming*. [S.l.]: Lulu. com, 2008.
- [17] PISZCZ, A.; SOULE, T. Genetic programming: Optimal population sizes for varying complexity problems. In: ACM. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. [S.l.], 2006. p. 953–954.
- [18] GIACOBINI, M.; TOMASSINI, M.; VANNESCHI, L. Limiting the number of fitness cases in genetic programming using statistics. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. [S.l.], 2002. p. 371–380.
- [19] KÖTZING, T. et al. The max problem revisited: the importance of mutation in genetic programming. *Theoretical computer science*, Elsevier, v. 545, p. 94–107, 2014.
- [20] MNIIH, V. et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [21] LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [22] HILL, A. et al. *Stable Baselines*. [S.l.]: GitHub, 2018. <https://github.com/hill-a/stable-baselines>.
- [23] MOORE, A. W. Efficient memory-based learning for robot control. Citeseer, 1990.

APÊNDICES

Apêndice A Códigos

Todos os códigos utilizados neste trabalho podem ser encontrados no repositório abaixo:

<https://github.com/jpuerj/gp-openai-gym>

Apêndice B Condições Iniciais

As condições iniciais de um ambiente são determinadas pela faixa de valores para as quais as variáveis de estado podem ser inicializadas, indicando o início de um episódio. Todos os valores aleatórios são gerados a partir de uma distribuição probabilística uniforme.

B.1 Pêndulo Invertido

Tabela 14: Condições iniciais do ambiente *CartPole-v1*.

Variável	Valor Mínimo	Valor Máximo
s	-0.05	0.05
\dot{s}	-0.05	0.05
v	-0.05	0.05
\dot{v}	-0.05	0.05

B.2 Pêndulo Swing-up

Tabela 15: Condições iniciais do ambiente *Pendulum-v0*.

Variável	Valor Mínimo	Valor Máximo
$\cos(\theta)$	-1	1
$\sin(\theta)$	-1	1
$\dot{\theta}$	-8	8

B.3 Carro na Ladeira

Tabela 16: Condições iniciais do ambiente *Pendulum-v0*.

Variável	Valor Mínimo	Valor Máximo
s	-0.6	-0.4
\dot{s}	0	0