



Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

João Pedro Barros Ferreira

**Programação Genética Aplicada a
Problemas de Aprendizagem por Reforço**

Rio de Janeiro

2020

João Pedro Barros Ferreira

**Programação Genética Aplicada a
Problemas de Aprendizagem por Reforço**

Projeto Final de Curso apresentado, como
requisito parcial para obtenção do título de
Graduado em Engenharia Elétrica, à Facul-
dade de Engenharia, da Universidade do Es-
tado do Rio de Janeiro. Área de concen-
tração: Sistemas Eletrônicos.

Orientador: Prof. Dr. Douglas Mota Dias

Coorientador: Prof. Dr. Téo Cerqueira Revoredo

Rio de Janeiro

2020

CATALOGAÇÃO NA FONTE
UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

S237

Ferreira, João Pedro Barros

Programação Genética Aplicada a Problemas de Aprendizagem por Reforço / João Pedro Barros Ferreira. - 2019.
127 f.

Orientadores: Douglas Mota Dias; Téo Cerqueira Revoredo.

Projeto Final apresentado à Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia, para obtenção do grau de bacharel em Engenharia Elétrica.

1. Programação Genética - Monografias. 2. Aprendizagem por Reforço - Monografias. I. Dias, Douglas Mota. II. Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia. III. Título.

CDU 621.3

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

Assinatura

Data

João Pedro Barros Ferreira

**Programação Genética Aplicada a
Problemas de Aprendizagem por Reforço**

Projeto Final de Curso apresentado, como requisito parcial para obtenção do título de Graduado em Engenharia Elétrica, à Faculdade de Engenharia, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Eletrônicos.

Aprovado em:

Banca Examinadora:

Prof. Dr. Douglas Mota Dias (Orientador)
Departamento de Eletrônica e Telecomunicações da Faculdade de Engenharia da UERJ

Prof. Dr. Téo Cerqueira Revoredo (Coorientador)
Departamento de Eletrônica e Telecomunicações da Faculdade de Engenharia da UERJ

Prof. Dr. Jorge Luís Machado Do Amaral
Departamento de Eletrônica e Telecomunicações da Faculdade de Engenharia da UERJ

Rio de Janeiro

2020

DEDICATÓRIA

Dedico este trabalho à minha falecida vó, cuja sabedoria, bondade e coragem serviu de exemplo para todos que a conheceram.

“Aceita as surpresas que transformam teus planos, derrubam teus sonhos, dão rumo totalmente diverso ao teu dia e, quem sabe, à tua vida. Não há acaso. Dá liberdade ao Pai, para que Ele mesmo conduza a trama dos teus dias.” - Dom Helder

AGRADECIMENTO

Aos meus pais Ricardo e Jane pelo apoio e incentivo que serviram de alicerce para as minhas realizações.

Ao meu irmão Gabriel pela amizade e atenção dedicadas quando sempre precisei.

Ao meu professores orientadores Dr. Douglas e Dr. Téo pelas valiosas contribuições dadas durante todo o processo.

A todos os meus amigos do curso de graduação que compartilharam dos inúmeros desafios que enfrentamos, sempre com o espírito colaborativo.

Também quero agradecer à UERJ e o seu corpo docente que demonstrou estar comprometido com a qualidade e excelência do ensino.

RESUMO

Ferreira, J.P.B. *Programação Genética Aplicada a Problemas de Aprendizagem por Reforço.* 127 f. Projeto Final de Curso (Graduação em Engenharia Elétrica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2020.

Este trabalho busca aplicar o algoritmo de programação genética em problemas formulados com base nos conceitos da aprendizagem por reforço. São utilizadas duas bibliotecas na linguagem de programação Python: DEAP e Gym, que implementam algoritmos evolucionários distribuídos e ambientes de simulação inspirados em processos de decisão de Markov, respectivamente. Foi utilizado como exemplo, ao longo do trabalho, a aplicação da proposta no problema clássico de controle do pêndulo invertido. O método foi aplicado em outros sistemas, incluindo uma aplicação real que utiliza um veículo terrestre autônomo, construído por alunos do curso de engenharia elétrica da UERJ.

Palavras-chave: Programação Genética, Aprendizagem por Reforço.

ABSTRACT

This work seeks to apply the genetic programming algorithm to problems formulated with reinforcement learning concepts. Two libraries are used in the Python programming language: DEAP and Gym, which implements distributed evolutionary algorithms and simulation environments inspired by Markov's decision processes, respectively. As an example, throughout the work, the classic case of the control of an inverted pendulum was used. The method was also applied to other systems, including a real application that uses an autonomous vehicle, built by students of the electrical engineering course at the Rio de Janeiro State University.

Keywords: Genetic Programming, Reinforcement Learning.

Listas de Figuras

FIGURA 1 – Um exemplo de indivíduo representado por uma sequência de caracteres com tamanho fixo.	18
FIGURA 2 – Representação de um indivíduo na <i>programação genética</i> . As letras “abcdef” podem representar operações matemáticas, constantes ou até tomadas de decisão.	19
FIGURA 3 – Pêndulo invertido.	20
FIGURA 4 – A biblioteca <i>Gym</i> fornece alguns problemas de controle clássico como o pêndulo invertido (esquerda) e o pêndulo <i>swingup</i>	20
FIGURA 5 – Aprendizagem evolucionária.	22
FIGURA 6 – Um indivíduo aleatório representando uma lei de controle. c_1 e c_2 são constantes. s_1 e s_2 são leituras de um sensor.	23
FIGURA 7 – Um exemplo de inicialização de uma população de três indivíduos, com as características descritas na Equação 1.	24
FIGURA 8 – Sistema de controle por realimentação. O sinal de erro, $e(t)$, é a entrada do controlador, que atua na planta através do sinal de controle, $u(t)$	25
FIGURA 9 – O indivíduo interage com a planta utilizando apenas medições do estado atual do sistema.	25
FIGURA 10 – Representação 2D da busca pelo mínimo da função custo. A operação de mutação permite uma busca em larga escala (em vermelho). Em azul, a operação de cruzamento busca a convergência para o mínimo local.	27
FIGURA 11 – Mutação aplicada à um ponto correspondente a uma variável terminal gerando um <i>ramo</i>	27
FIGURA 12 – Mutação aplicada em um ponto correspondente a um <i>ramo</i> . A nova sub-árvore possui apenas uma <i>variável terminal</i>	28
FIGURA 13 – Exemplo da operação de cruzamento atuando em dois indivíduos. . .	29
FIGURA 14 – Operação de <i>replicação</i> aplicada em um indivíduo selecionado. . .	29
FIGURA 15 – Ciclo evolucionário para programação genética.	31
FIGURA 16 – Sistema do pêndulo invertido.	32

FIGURA 17 – Processo de decisão de Markov..	34
FIGURA 18 – Diagrama de classes para o ambiente de simulação do pêndulo invertido, gerado com o utilitário <i>pyreverse</i>	36
FIGURA 19 – Forma geral de um algoritmo para realização de uma simulação no <i>OpenAi Gym</i>	38
FIGURA 20 – <i>Box</i> é uma classe que permite a criação de conjuntos multidimensionais, no ambiente do pêndulo invertido é apenas uma lista (unidimensional).	39
FIGURA 21 – Recompensa recebida pelo agente.	39
FIGURA 22 – Valor lógico que indica se o episódio terminou ou não.	39
FIGURA 23 – Módulos da biblioteca DEAP e suas funcionalidades.	41
FIGURA 24 – A saída depende do sinal da expressão que resulta da avaliação do indivíduo.	44
FIGURA 25 – As linhas pontilhadas indicam a maior distância entre a <i>raiz</i> e uma <i>variável terminal</i> . Este exemplo utiliza o conjunto primitivo \mathcal{P} definido anteriormente.	47
FIGURA 26 – 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).	62
FIGURA 27 – Número de indivíduos, em algumas gerações, que obtiveram cada faixa de aptidão.	63
FIGURA 28 – Comprimento dos indivíduos por geração.	64
FIGURA 29 – Complexidade dos indivíduos em cada geração.	64
FIGURA 30 – Número de ocorrências dos operadores e variáveis terminais, nos indivíduos da última geração.	65
FIGURA 31 – Primeiro indivíduo do hall da fama, na primeira execução do algoritmo.	66
FIGURA 32 – Vigésimo terceiro indivíduo do hall da fama, na primeira execução do algoritmo.	67
FIGURA 33 – Comprimento médio dos indivíduos que obtiveram aptidão maior que 400.	68

FIGURA 34 – Controle do sistema pelo indivíduo da Figura 31. O pêndulo é levado rapidamente a um estado de baixa oscilação.	69
FIGURA 35 – Recompensa média acumulada em função do número de passos de simulação. A linha contínua representa a média móvel..	70
FIGURA 36 – Recompensa média ao longo dos passos de simulação. A linha azul contínua indica a média móvel.	70
FIGURA 37 – Posição do carrinho e ângulo do bastão, em função do tempo, com a atuação do agente DQN.	71
FIGURA 38 – Pêndulo <i>Swing-up</i>	72
FIGURA 39 – Função <i>clip</i>	75
FIGURA 40 – Aptidão dos indivíduos, ao longo das gerações.	76
FIGURA 41 – Histograma da aptidão dos indivíduos..	76
FIGURA 42 – Comprimento dos indivíduos.	77
FIGURA 43 – Complexidade dos indivíduos.	77
FIGURA 44 – Histograma de operadores e variáveis terminais, na última geração. .	78
FIGURA 45 – Primeiro integrante do hall da fama.	79
FIGURA 46 – Segundo integrante do hall da fama..	80
FIGURA 47 – Controle exercido pelo indivíduo da Figura 45.	81
FIGURA 48 – Evolução da recompensa acumulada para o agente DDPG no pêndulo swing-up.	82
FIGURA 49 – Dinâmica do pêndulo swing-up sob ação do agente DDPG.	82
FIGURA 50 – Renderização 3D do sistema pêndulo duplo invertido.	83
FIGURA 51 – Ângulos θ , γ e a posição do veículo (s).	84
FIGURA 52 – Histograma da aptidão dos indivíduos, em algumas gerações, para o pêndulo duplo invertido..	86
FIGURA 53 – Medidas de aptidão da população, em cada geração..	86
FIGURA 54 – Medidas de comprimento da população, ao longo das gerações. . . .	87
FIGURA 55 – Medidas de complexidade dos indivíduos, em função das gerações. .	87
FIGURA 56 – Número de ocorrências dos operadores e variáveis terminais, na última geração..	88
FIGURA 57 – Indivíduo mais apto da primeira execução do algoritmo.	89
FIGURA 58 – Avaliação do melhor indivíduo observado na primeira execução. . . .	90

FIGURA 59 – Evolução da recompensa acumulada do agente DDPG. A linha azul representa a média móvel.	90
FIGURA 60 – Problema do carro na ladeira.	91
FIGURA 61 – Histograma da aptidão dos indivíduos, em algumas gerações, para o pêndulo duplo invertido.	94
FIGURA 62 – Aptidão da população.	94
FIGURA 63 – Comprimento dos indivíduos.	95
FIGURA 64 – Complexidade da população.	95
FIGURA 65 – Ocorrências dos operadores e variáveis terminais, na última geração. .	96
FIGURA 66 – Melhor indivíduo da primeira execução.	97
FIGURA 67 – Atuação do indivíduo da Figura 66 em um episódio aleatório.	98
FIGURA 68 – Agente DQN e a recompensa acumulada ao longo dos passos de tempo.	98
FIGURA 69 – Modelo de bicicleta com o ponto de referência no centro de gravidade.	100
FIGURA 70 – Visão geral das variáveis fornecidas ao agente e outras informações sobre a implementação da simulação do veículo autônomo.	101
FIGURA 71 – Protótipo do veículo autônomo.	102
FIGURA 72 – <i>Frame</i> do vídeo em um episódio de simulação.	104
FIGURA 73 – Primeira situação.	106
FIGURA 74 – Aptidão dos indivíduos.	108
FIGURA 75 – Histograma das aptidões.	108
FIGURA 76 – Árvores de controle do melhor indivíduo encontrado.	109
FIGURA 77 – Trajetória do indivíduo de melhor desempenho. As setas indicam a orientação do veículo ao longo da trajetória.	109
FIGURA 78 – Saída das funções de controle ao longo da simulação.	110
FIGURA 79 – Visão geral do problema 2.	111
FIGURA 80 – Aptidão dos indivíduos.	112
FIGURA 81 – Histograma das aptidões.	112
FIGURA 82 – Indivíduo que obteve melhor desempenho.	113
FIGURA 83 – Trajetória do indivíduo da Figura 82.	114
FIGURA 84 – Funções temporais geradas.	114
FIGURA 85 – Visão geral do problema 3.	116

FIGURA 86 – Aptidão da população ao longo de 30 gerações.	117
FIGURA 87 – Histograma de aptidões.	117
FIGURA 88 – Árvores de controle do indivíduo que apresentou melhor desempenho.	118
FIGURA 89 – Trajetórias exibidas pelo melhor indivíduo com ângulo de orientação inicial igual a 0. As coordenadas X_0 e Y_0 são os valores extremos possíveis (0.15 e 0.20) e o valor médio (0.20).	118
FIGURA 90 – $\theta_0 = 30^\circ$	119
FIGURA 91 – $\theta_0 = -30^\circ$	119
FIGURA 92 – Visão geral do problema 4.	121
FIGURA 93 – Aptidão ao longo das gerações para o problema 3.	122
FIGURA 94 – Histograma das aptidões.	122
FIGURA 95 – Árvores de controle do indivíduo de melhor desempenho encontrado..	123
FIGURA 96 – Trajetórias exibidas pelo melhor indivíduo com ângulo de orientação igual a 0. As coordenadas X_0 e Y_0 são os valores extremos (0.15 e 0.20) e o valor médio (0.20).	123
FIGURA 97 – $\theta_0 = 30^\circ$	124
FIGURA 98 – $\theta_0 = -30^\circ$	124

Lista de Tabelas

TABELA 1 – Variáveis de estado fornecidas como observação do estado atual (S_t)).	35
TABELA 2 – Variáveis terminais.	42
TABELA 3 – Conjunto de operadores lógicos e matemáticos.	42
TABELA 4 – Conjunto de operadores lógicos e matemáticos.	43
TABELA 5 – Parâmetros da PG para o problema do pêndulo invertido.	61
TABELA 6 – Comparaçāo entre a programação genética e DQN, para o problema do pêndulo invertido.	71
TABELA 7 – Variáveis que compõem a observação do pêndulo swing-up.	73
TABELA 8 – Parâmetros da programação genética aplicada ao pêndulo swing-up.	74
TABELA 9 – Comparaçāo entre a programação genética e DDPG para o pêndulo swing-up.	82
TABELA 10 – Variáveis disponibilizadas como observação do pêndulo duplo invertido.	84
TABELA 11 – Comparaçāo entre PG e DDPG para o pêndulo duplo invertido.	91
TABELA 12 – Variáveis de estado para o problema do carro na ladeira.	92
TABELA 13 – Parâmetros utilizados para o problema do carro na ladeira.	93
TABELA 14 – Comparaçāo entre PG e DQN para o problema do carro na ladeira.	98
TABELA 15 – Variáveis que fornecem informações sobre o sistema para o agente.	100
TABELA 16 – Limites definidos para algumas variáveis da Tabela 15.	102
TABELA 17 – Limites para as variáveis de controle v e ϕ .	103
TABELA 18 – Restrições ao controle exercido por meio das variáveis v e ϕ .	103
TABELA 19 – Parâmetros utilizados para o problema 1 do carro robô.	107
TABELA 20 – Problema 1 - custo computacional.	110
TABELA 21 – Problema 2: custo computacional.	115
TABELA 22 – Custo computacional para o problema 3.	120
TABELA 23 – Problema 4: custo computacional.	125
TABELA 24 – Condições iniciais do ambiente <i>CartPole-v1</i> .	133
TABELA 25 – Condições iniciais do ambiente <i>Pendulum-v0</i> .	133
TABELA 26 – Condições iniciais do ambiente <i>Pendulum-v0</i> .	134

TABELA 27 – Características do computador e dos programas utilizados para a execução do algoritmo 134

Sumário

<u>INTRODUÇÃO</u>	15
1 A PROGRAMAÇÃO GENÉTICA	22
2 SIMULAÇÃO DE SISTEMAS DINÂMICOS	32
2.1 PÊNDULO INVERTIDO	32
2.2 BIBLIOTECA “OPENAI GYM”	33
3 DEAP: IMPLEMENTAÇÃO DOS AGENTES	40
3.1 REPRESENTAÇÃO E INICIALIZAÇÃO	41
3.1.1 Representação	41
3.1.2 Aptidão	45
3.1.3 Inicialização	46
3.1.4 População	48
3.2 AVALIAÇÃO DE INDIVÍDUOS	50
3.3 SELEÇÃO DE INDIVÍDUOS	53
3.4 OPERADORES GENÉTICOS	54
3.4.1 Replicação	54
3.4.2 Mutação	54
3.4.3 Cruzamento	55
3.4.4 Seleção Aleatória do Operador	56
3.5 CICLO ITERATIVO	57
4 ESTUDOS DE CASO	59
4.1 Pêndulo Invertido	61
4.2 Pêndulo <i>Swing-up</i>	72
4.3 Pêndulo Duplo Invertido	83
4.4 Carro na Ladeira	91
4.5 Veículo Terrestre Autônomo	99
4.5.1 Problema 1	105
4.5.2 Problema 2	110
4.5.3 Problema 3	115

4.5.4	Problema 4	120
CONCLUSÃO		126
REFERÊNCIAS		130
APÊNDICES		133
A	Códigos	133
B	Condições Iniciais	133
B.1	Pêndulo Invertido	133
B.2	Pêndulo Swing-up	133
B.3	Carro na Ladeira	134
C	Hardware e Software Utilizados	134

INTRODUÇÃO

Há anos criam-se máquinas que podem ser consideradas inteligentes, dada a complexidade de suas tarefas. Entretanto, uma atenção especial vem sendo direcionada às máquinas que possuem a capacidade de aprender, ou evoluir com as experiências, isto é, não necessariamente é preciso ensiná-las como realizar essas tarefas complexas. Arthur Samuel 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?”

Este campo, conhecido atualmente por *aprendizagem de máquinas*, ou *Machine Learning*, se baseia principalmente na ideia mencionada por Arthur Samuel. Neste paradigma, a tarefa do humano é criar uma máquina complexa o suficiente que a permita o aprendizado por experiência.

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

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

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 desempenho [2].

Em geral, é possível dividir os algoritmos de aprendizagem de máquinas em quatro categorias [3]:

- a) **Aprendizado Supervisionado:** um conjunto de exemplos de treinamento (com a resposta correta) é fornecido ao algoritmo, com base nesses dados, as respostas são generalizadas para qualquer entrada possível. Em outras palavras, os dados são fornecidos ao algoritmo, incluindo o que se espera de resposta, cabe ao programa

aprender com a experiência fornecida e deduzir corretamente a solução 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 realiza buscas no espaço de possíveis soluções, até que encontre a solução correta.
- d) **Sistemas de Recomendação:** geram recomendações de produtos ou serviços ao buscar a previsão das preferências de um usuário.

A principal diferença da aprendizagem por reforço 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, enquanto a aprendizagem não-supervisionada pode 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.

Foi mencionado 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, é preciso realizar essa exploração de forma adaptativa ou inteligente [1].

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

- a) Mede-se o desempenho de diversos indivíduos, inicializados com características aleatórias.
- b) Os agentes que apresentarem um bom desempenho 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. Os agentes que apresentarem melhor desempenho são novamente selecionados e modificados.
- d) O processo se repete até um critério de término pré-estabelecido.

É possível verificar a semelhança da escolha de soluções 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. Essas alterações são conhecidas como 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 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 estrutura do 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.

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

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

Para muitos problemas a representação natural de uma solução é um programa hierárquico de tamanho variável, ao invés de uma estrutura de comprimento fixo. Isso ocorre porque, frequentemente, não se sabe de antemão as dimensões da solução procurada.

Programas de computador hierárquicos, particularmente escritos em linguagens funcionais e/ou recursivas, podem ser facilmente representados por árvores. Por essa razão, linguagens que processam listas são escolhas naturais para abordar a programação genética [4].

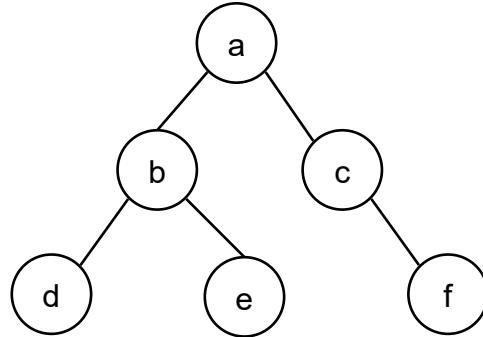


Figura 2: Representação de um indivíduo na *programação genética*. As letras “abcdef” podem representar operações matemáticas, constantes ou até tomadas de decisão.

A estrutura da Figura 2 representa um programa de computador. De fato, é possível 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 situações.

Este trabalho busca aplicar a *programação genética* (PG) em problemas clássicos de controle (4). O sistema do pêndulo invertido da Figura 3 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 a força F como variável de controle, podendo esta ser aplicada na direção positiva ou negativa do eixo x .

Para realizar a avaliação dos indivíduos, é necessário uma ferramenta de simulação. Foi utilizada a biblioteca *Gym* [5] da linguagem de programação *Python*. O principal propósito desta biblioteca é criar um ambiente padronizado para testar algoritmos de aprendizagem por reforço, porém será utilizada em um problema a ser solucionado por programação genética.

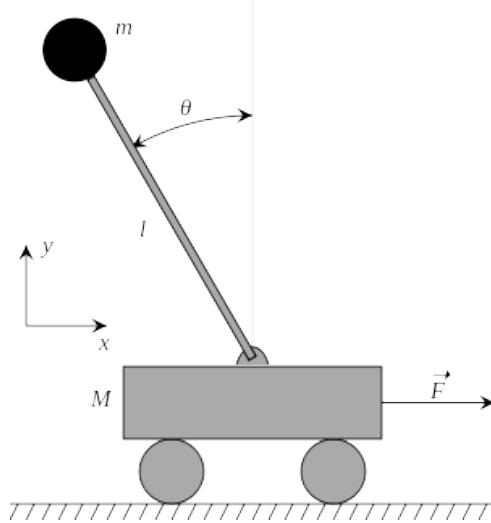


Figura 3: Pêndulo invertido.

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

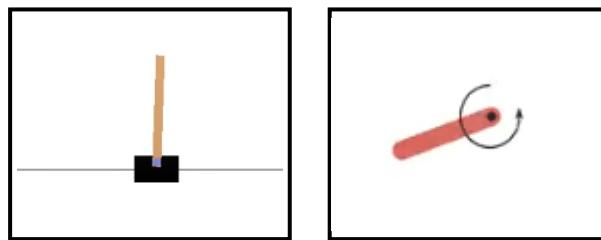


Figura 4: A biblioteca *Gym* fornece alguns problemas de controle clássico como o pêndulo invertido (esquerda) e o pêndulo *swingup*.

Fonte: [5]

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

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

Inicialmente, é feita uma análise mais detalhada do ciclo evolutivo artificial, no contexto da programação genética. Além disso, uma verificação detalhada da representação da Figura 2 e seus componentes é exigida. Então, aprofunda-se o estudo do pêndulo invertido, buscando formas de incluir na representação dos indivíduos informações importantes

para a resolução do problema.

Com o modo de representação dos indivíduos definido, pode-se pensar em como inicializá-los. O conjunto de ferramentas DEAP proporciona diversos métodos que simplificam este processo. Aborda-se, então, maneiras de verificar a aptidão dos indivíduos, que representa, em suma, o desempenho de cada solução encontrada ao longo da execução do algoritmo.

Com uma medida de desempenho atribuída a cada indivíduo, pode-se implementar um método para selecionar indivíduos com base em sua aptidão. São discutidas as possíveis abordagens para essa etapa. Finalmente, as operações genéticas são implementadas em seguida.

1 A PROGRAMAÇÃO GENÉTICA

Um breve sumário da programação genética foi incluído na Introdução. Entretanto, é preciso aprofundar os conceitos mencionados. Neste capítulo são abordados os aspectos mais específicos da representação da Figura 2, isto é, o que exatamente são os caracteres *abcdef* em cada problema. Uma visão geral do ciclo evolutivo artificial é incluída neste capítulo, e servirá de base para os próximos, já que o objetivo é implementar o algoritmo completo.

A aprendizagem evolucionária, em geral, pode ser sumarizada com etapas semelhantes às apresentadas na Figura 5.

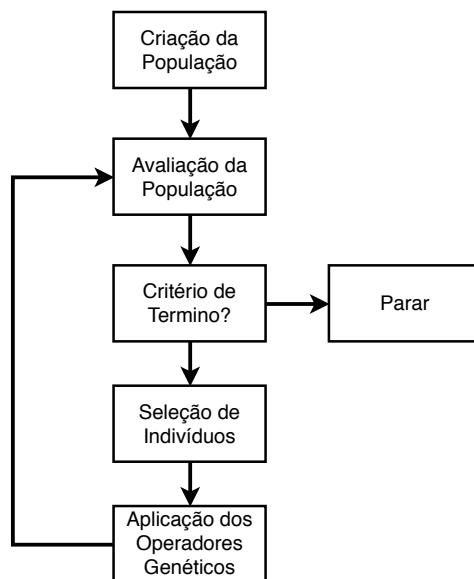


Figura 5: Aprendizagem evolucionária.

O primeiro passo é a criação da população de indivíduos, de acordo com a representação escolhida. A abordagem deste trabalho utiliza o conceito proposto por Koza [1], onde cada indivíduo possui uma estrutura recursiva (semelhante a uma árvore) que representa, em sua essência, um programa de computador.

Para os problemas que serão abordados, em suma, o objetivo é encontrar uma lei de controle capaz de estabilizar o sistema ou alcançar um propósito definido. Por exemplo, um indivíduo aleatório inicializado poderia ter uma estrutura igual à apresentada na Figura 6. A lei de controle seria dada pela expressão matemática que resulta do cálculo recursivo da árvore que representa o indivíduo.

Na Figura 6, círculos representam **operadores**, enquanto os quadrados, **variáveis**

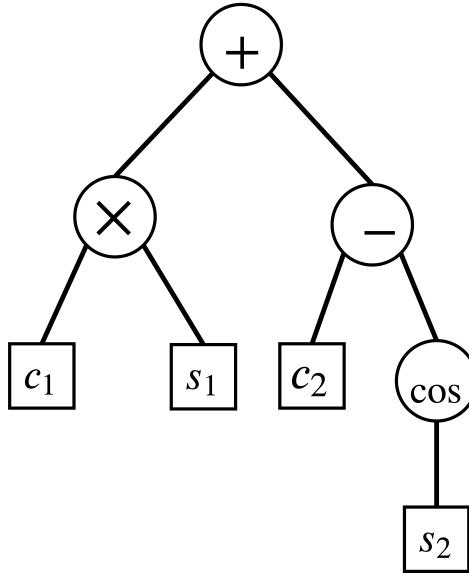


Figura 6: Um indivíduo aleatório representando uma lei de controle. c_1 e c_2 são constantes. s_1 e s_2 são leituras de um sensor.

terminais. Como a representação utilizada se assemelha à uma árvore, é possível definir como:

- Raiz: o operador do topo (i.e., que não é argumento de outro operador).
- Ramo: a estrutura composta por um operador e seus argumentos (mesmo que estes sejam outros ramos).
- Folha: a variável terminal, isto é, uma constante ou uma variável. Caracteriza-se por não ser operador e, portanto, não possui argumentos.

Os operadores podem ser matemáticos ou lógicos. Quanto à aridade, pode-se utilizar operadores que aceitem um ou mais argumentos. As variáveis terminais podem ser constantes como 1 e 0 ou até números aleatórios dentro de uma faixa. Em problemas de controle é comum admitir como variáveis terminais uma leitura de sensor, por exemplo.

Em linguagens de programação baseadas no processamento de listas, o indivíduo da Figura 6 poderia ser descrito através da expressão:

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

Geralmente, a representação em forma de árvore é mais intuitiva. Entretanto, costuma ser mais eficiente trabalhar com sequências de caracteres, representando uma lista.

É possível estipular, também, um tamanho mínimo e máximo de cada árvore, eliminando assim expressões 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 independentes, executadas de forma sequencial.

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:

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

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

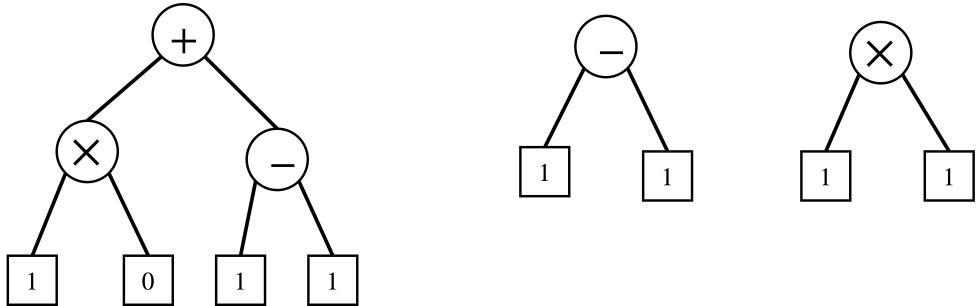


Figura 7: Um exemplo de inicialização de uma população de três indivíduos, com as características descritas na Equação 1.

Após a inicialização da população é feita a avaliação de cada indivíduo. O objetivo desta etapa é verificar a aptidão de todos os membros da população, através de uma medida que reflete o que se busca como solução para o problema, ou seja, a função que determina o desempenho da solução é específica a cada aplicação.

Em problemas de controle, geralmente o objetivo é manter a posição de um objeto o mais próximo possível de um valor de referência, X_{ref} , utilizando uma medição do valor

atual, X_{med} . Diagramas como o da Figura 8 são comuns na teoria clássica de controle.

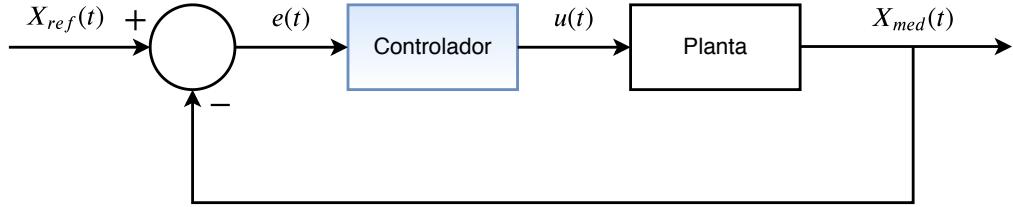


Figura 8: Sistema de controle por realimentação. O sinal de erro, $e(t)$, é a entrada do controlador, que atua na planta através do sinal de controle, $u(t)$.

A avaliação de um indivíduo pode ser representada como uma função custo, \mathbf{J} . O objetivo do algoritmo é, portanto, achar um sinal de controle que minimiza \mathbf{J} , uma métrica que leva em consideração o erro $e(t)$, em um intervalo de tempo.

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

Na programação genética, o indivíduo age como o controlador da Figura 8, porém, recebe apenas uma medida do estado atual para guiar sua atuação. Isto é, o erro é utilizado apenas por um processo externo à interação do indivíduo com a planta, como um meio de realizar o cálculo de \mathbf{J} . Este, por sua vez, é um valor que pode representar a aptidão deste indivíduo. Este processo pode ser observado na Figura 9.

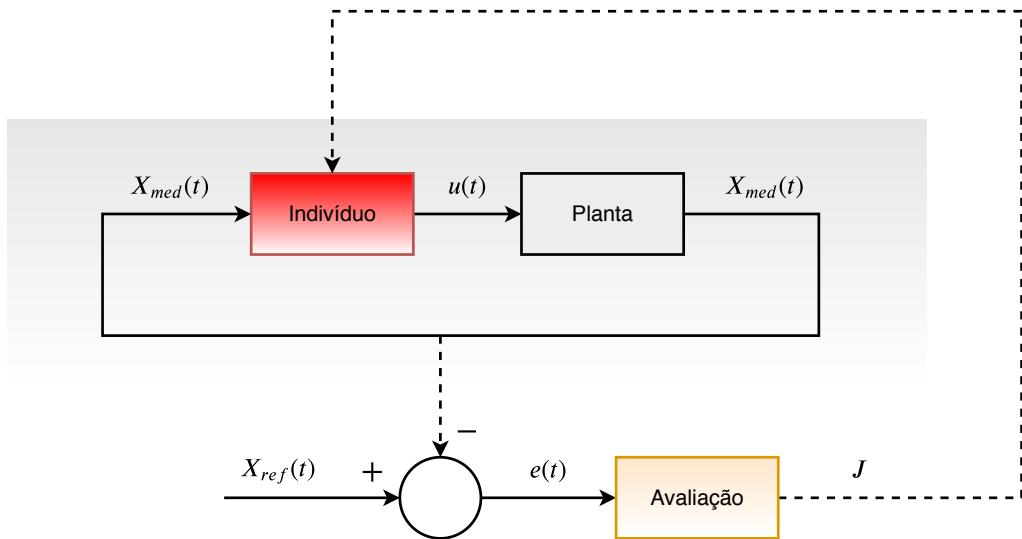


Figura 9: O indivíduo interage com a planta utilizando apenas medições do estado atual do sistema.

Com uma medida de aptidão associada a cada indivíduo, é possível selecionar os mais aptos. Um método básico e eficaz de seleção é comparar dois indivíduos aleatórios

na população, utilizando a métrica de desempenho estipulada, e escolher o que possui maior valor numérico de aptidão.

Esse método conhecido como *torneio* pode ser feito em dois ou mais níveis, isto é, são escolhidos quatro ou mais indivíduos aleatórios na população, realizando comparações dois a dois, de modo que apenas um indivíduo seja escolhido ao final.

Um dos operadores genéticos é aplicado neste indivíduo e a 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 o número de indivíduos gerados seja igual a quantidade de membros da população antiga.

Para cada indivíduo selecionado, apenas uma operação genética é aplicada, após ser escolhida de forma aleatória. Cada operação genética tem, portanto, uma probabilidade de ser selecionada. A soma das probabilidades de cada operação deve ser igual a um.

$$\left. \begin{array}{l} P_c: \text{probabilidade de cruzamento} \\ P_m: \text{probabilidade de mutação} \\ P_r: \text{probabilidade de replicação} \end{array} \right\} P_c + P_m + P_r = 1 \quad (3)$$

É interessante lembrar que o ciclo evolutivo 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 a uma busca em larga escala, enquanto *exploitation* refere-se a uma procura local.

De acordo com Brunton et al. [9], a procura pelo mínimo da função custo pode ser representada por gráficos semelhantes ao da Figura 10, onde a operação de cruzamento realiza uma busca local (*exploitation*), enquanto a mutação opera em larga escala (*exploration*).

A operação de cruzamento, demonstrada na Figura 13, 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 indivíduos gerados possuem **toda** sua estrutura formada por ramos 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* começa ao selecionar um ponto aleatório na árvore,

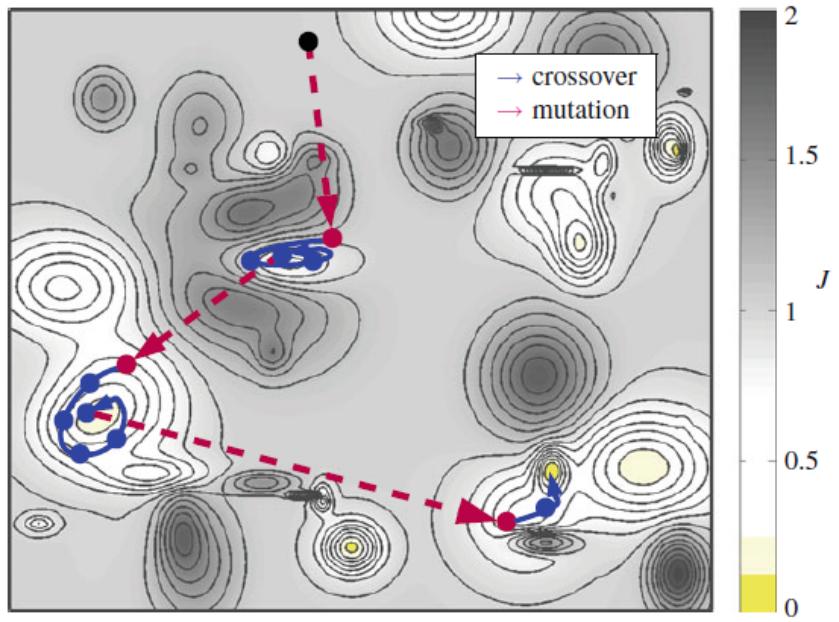


Figura 10: Representação 2D da busca pelo mínimo da função custo. A operação de mutação permite uma busca em larga escala (em vermelho). Em azul, a operação de cruzamento busca a convergência para o mínimo local.

Fonte: [9]

podendo este ser interno ou externo (operadores ou variáveis terminais). A mutação remove este ramo (ou folha) e uma nova sub-árvore é gerada aleatoriamente, produzindo uma variável terminal (Figura 11) ou um ramo (Figura 12).

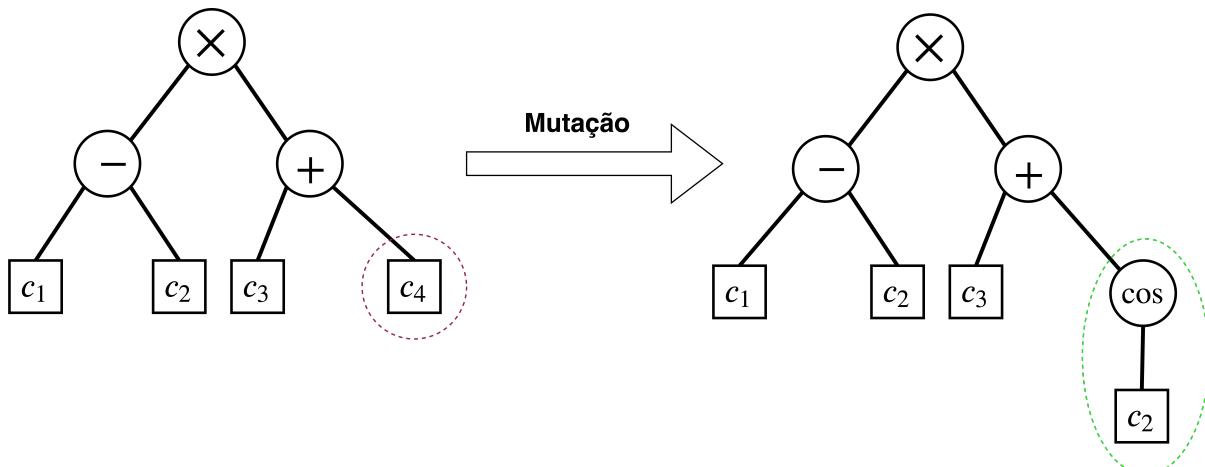


Figura 11: Mutação aplicada à um ponto correspondente a uma variável terminal gerando um *ramo*.

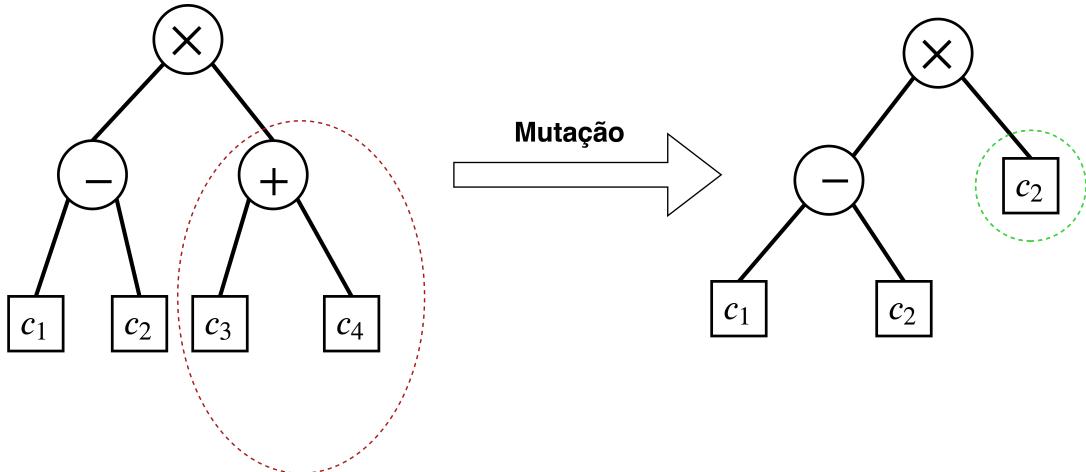


Figura 12: Mutação aplicada em um ponto correspondente a um *ramo*. A nova sub-árvore possui apenas uma *variável terminal*.

A operação de *cruzamento* atua a partir de dois indivíduos selecionados. Assim como na operação de *mutação*, um ponto aleatório na árvore de cada indivíduo é escolhido aleatoriamente e os ramos correspondentes são trocados entre as soluções (Figura 13).

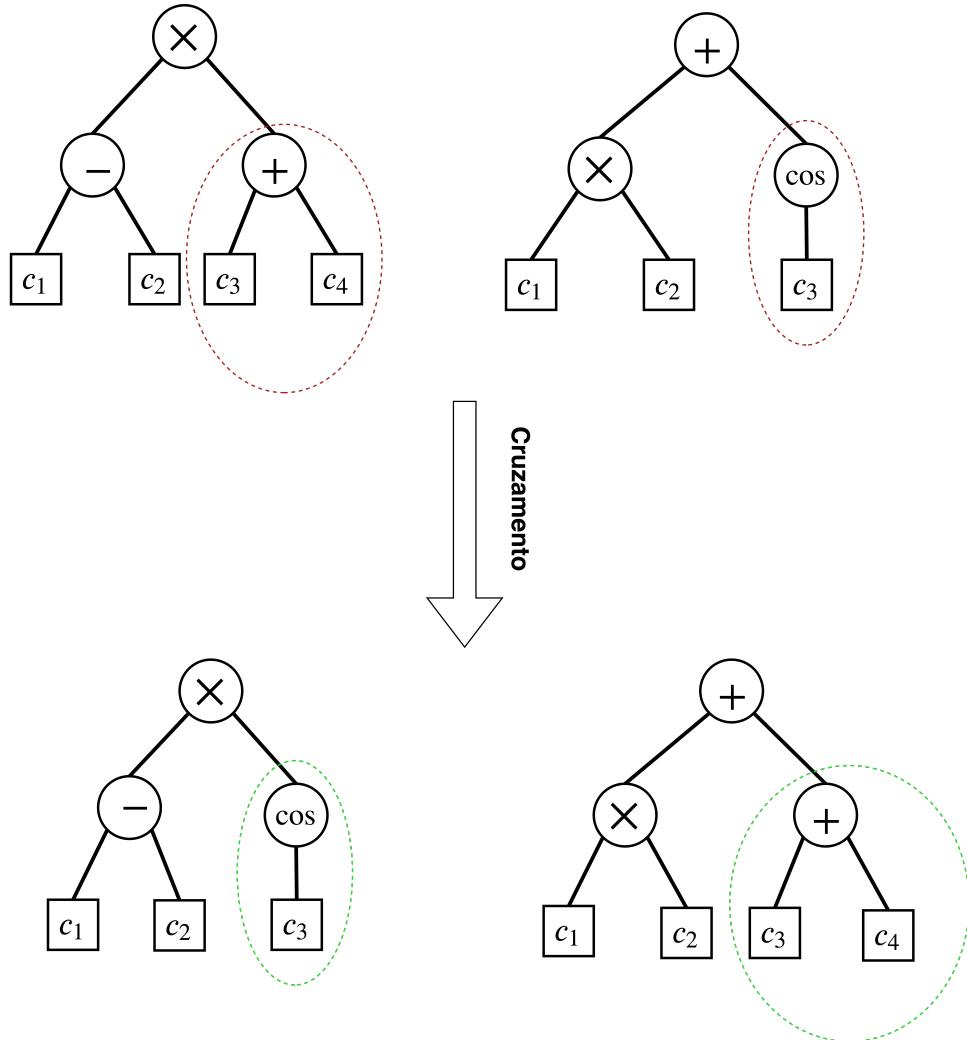


Figura 13: Exemplo da operação de cruzamento atuando em dois indivíduos.

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, conforme indica a Figura 14.

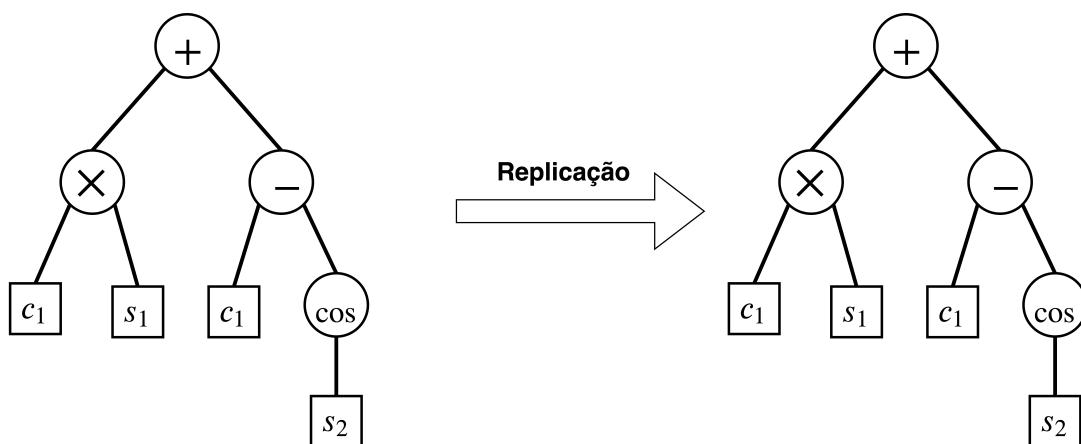


Figura 14: Operação de *replicação* aplicada em um indivíduo selecionado.

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; *Encapsulamento*, um método de identificação de ramos úteis que podem ser referenciados e *elitismo*, que copia os indivíduos de maior aptidão diretamente para a próxima geração.

Geralmente, o critério de término do ciclo evolutivo é 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.

Uma implementação de programação genética não se torna completa sem o estabelecimento de alguns parâmetros de controle, como por exemplo, o número de indivíduos da população ou a probabilidade de cada operação genética. Os principais parâmetros da PG são enumerados a seguir:

1. Em relação à 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

A partir dos parâmetros estabelecidos e dos métodos esclarecidos neste capítulo, é possível reformular o ciclo da Figura 5 para a inclusão de outros detalhes.

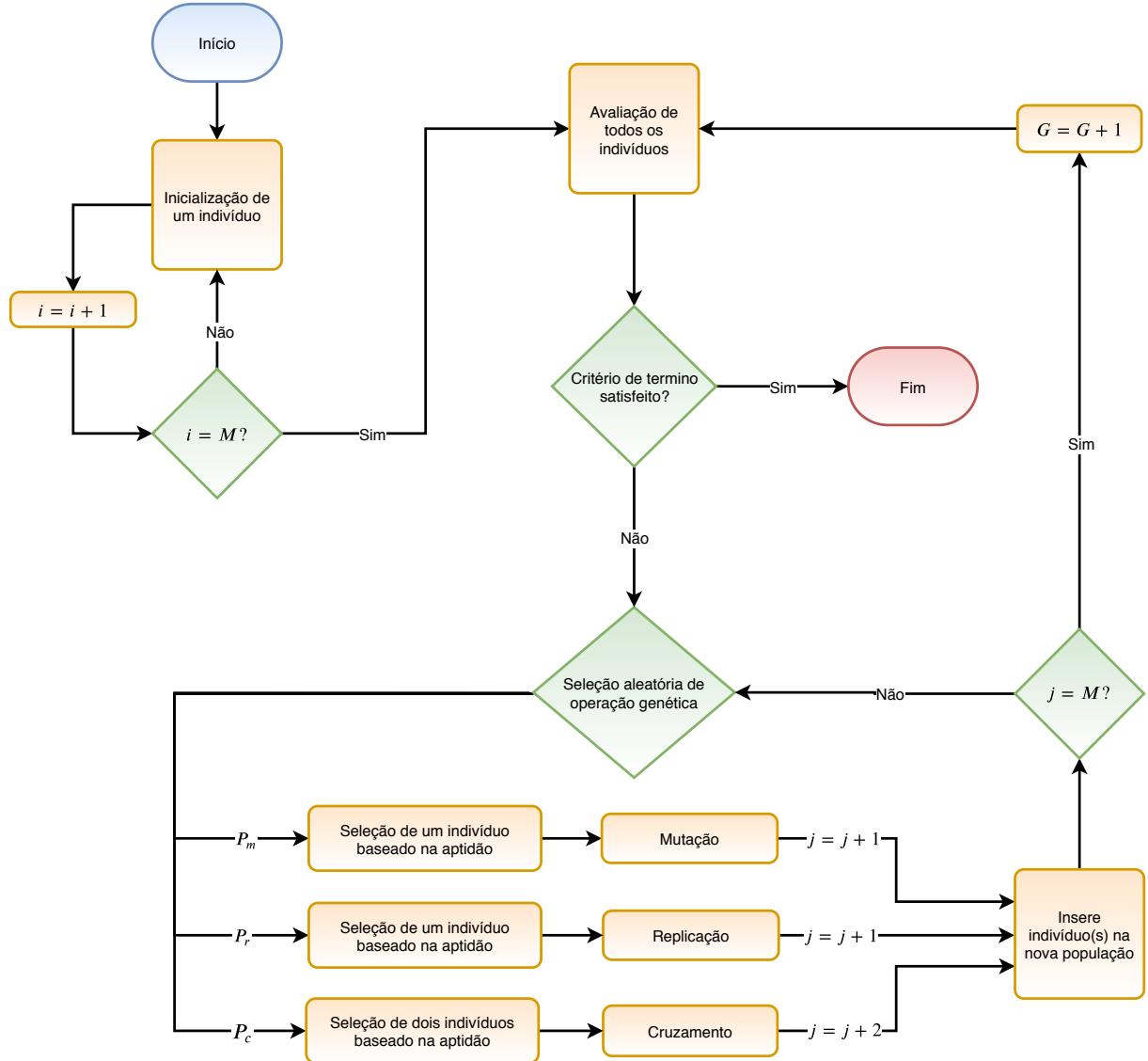


Figura 15: Ciclo evolucionário para programação genética.

Com base no fluxograma da Figura 15 é possível realizar a programação genética para um problema genérico. Resta, portanto, verificar as particularidades da implementação para cada problema abordado.

2 SIMULAÇÃO DE SISTEMAS DINÂMICOS

Foi mencionada a importância de se analisar o problema em que se deseja aplicar a programação genética. Essa análise permite a escolha da função de avaliação, das variáveis terminais e dos operadores. Ao estudar o pêndulo invertido, é possível verificar como um sistema dinâmico pode ser abordado com a PG.

2.1 PÊNDULO INVERTIDO

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 e, por isso, o sistema tem importância considerável na área de controle não-linear [10].

Existem diversas variações desse sistema [11]. O caso abordado neste trabalho trata de um bastão apoiado sobre um veículo que se movimenta com apenas um grau de liberdade, sendo a sua posição dada pela variável s . O ângulo θ representa a inclinação do bastão em relação a uma reta normal à superfície. O controle deve ser exercido por uma força \vec{F} , aplicada paralelamente à superfície, em qualquer sentido, com a finalidade de manter $\theta \approx 0^\circ$.

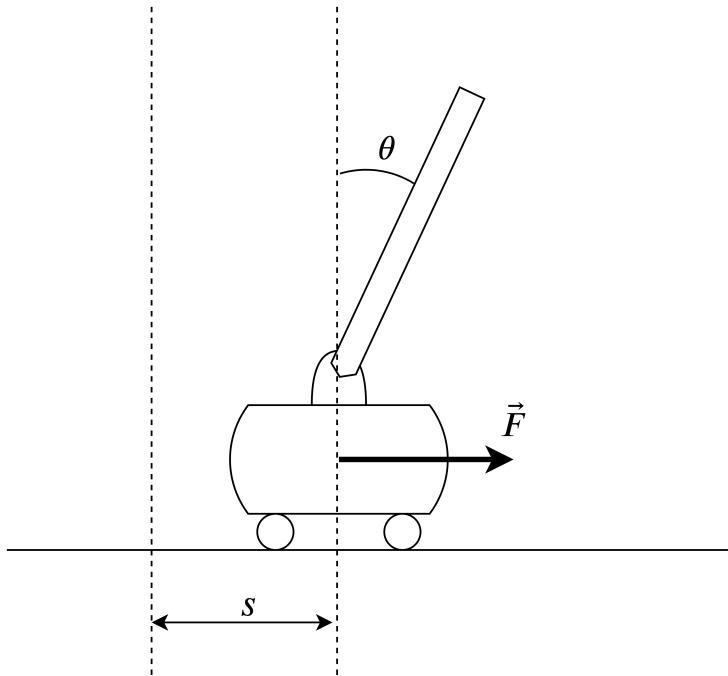


Figura 16: Sistema do pêndulo invertido.

O sistema de equações diferenciais que modela a dinâmica do problema é descrito na Equação 4.

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

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

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 o subscrito t (omitido neste trabalho) para algumas variáveis, com a finalidade de explicitar a dependência do tempo discreto. A Equação 4 serviu de base para a implementação do ambiente simulado na biblioteca *Gym* [13].

2.2 BIBLIOTECA “OPENAI GYM”

Antes de discutir a implementação do sistema na biblioteca *Gym*, é feita 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 através de *processos de decisão Markovianos*.

A principal ideia dos sistemas Markovianos é a interação de um *agente* com um *ambiente* externo, através 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). Uma ação (A_t), gerada pelo agente, influenciará o sistema dinâmico no próximo instante. O objetivo do agente é maximizar a expectativa de recompensa acumulada ao longo do tempo, através de suas ações. Este processo é descrito graficamente na Figura 17.

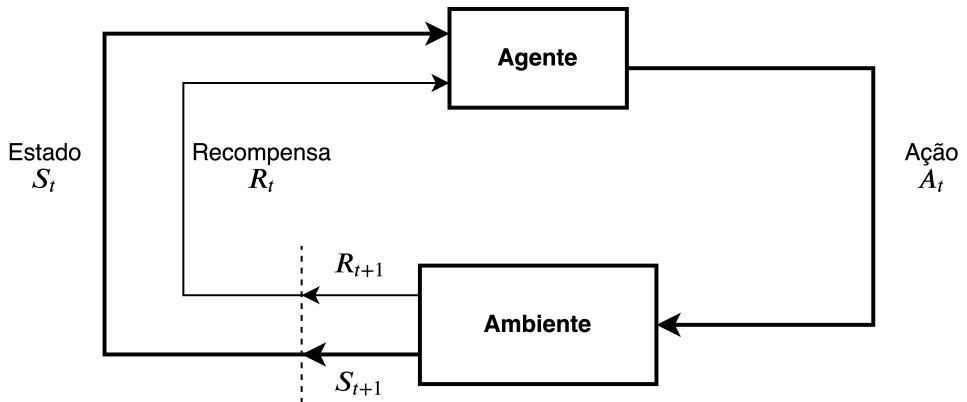


Figura 17: Processo de decisão de Markov.

Fonte: Adaptado de [14].

Para processos de decisão Markovianos, R_t e S_t são variáveis aleatórias com distribuições de probabilidade que dependem apenas do estado e ação anteriores, S_{t-1} e A_{t-1} , respectivamente. Para sistemas determinísticos, como os que são abordados neste trabalho, há um mapeamento $S_t, A_t \rightarrow S_{t+1}$.

As simulações da biblioteca Gym são baseadas em processos de decisão Markovianos. Portanto, a cada iteração existe uma variável acessível ao agente, que indica o estado atual do ambiente e sua recompensa, S_t e R_t , respectivamente. O agente será a máquina que aprende com a experiência, mais especificamente, com informações de estados e recompensas obtidas ao longo do tempo.

Estados Markovianos devem conter informações suficientes para que, dadas uma ação e uma observação do estado atual, seja possível prever o próximo estado. Desta forma, fica claro que a observação fornecida deve conter dados sobre as variáveis de estado do sistema.

A simulação ocorre em episódios de duração limitada: o término acontece quando as variáveis de estado ou o tempo decorrido excedem um certo valor, conforme exemplifica a Tabela 1.

Tabela 1: Variáveis de estado fornecidas como observação do estado atual (S_t)).

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	$-\infty$	∞

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.

Os componentes da simulação estão implementados na linguagem de programação *Python*, utilizando classes e categorias de dados específicos.

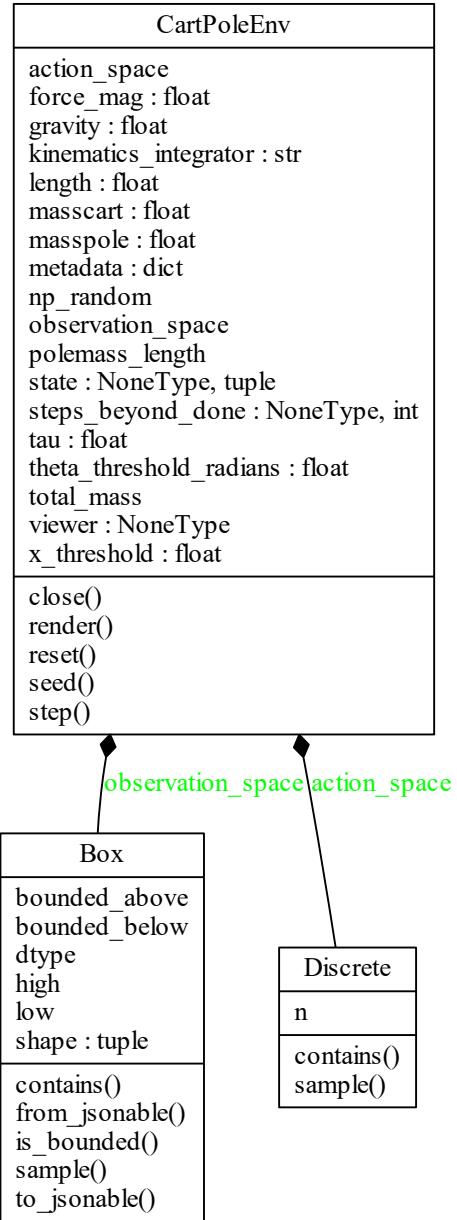


Figura 18: Diagrama de classes para o ambiente de simulação do pêndulo invertido, gerado com o utilitário *pyreverse*.

Na Figura 18, é possível verificar que os objetos `observation_space` e `action_space` pertencem às classes `Box` e `Discrete`, respectivamente, e compõem o ambiente de simulação. Essas classes determinam quais são as observações e ações possíveis na simulação, isto é, impõe restrições às ações que o agente pode tomar e à observação recebida. Na simulação do pêndulo invertido, as ações estão restritas aos valores inteiros 0 e 1, por exemplo, indicando a aplicação da força para a esquerda ou para a direita. Os

métodos disponíveis são utilizados para realizar a interação do agente com o ambiente. As principais funções existentes são 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*) Ambiente de simulação.

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) uma tupla 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 pode ser exemplificada no fluxograma da Figura 19.

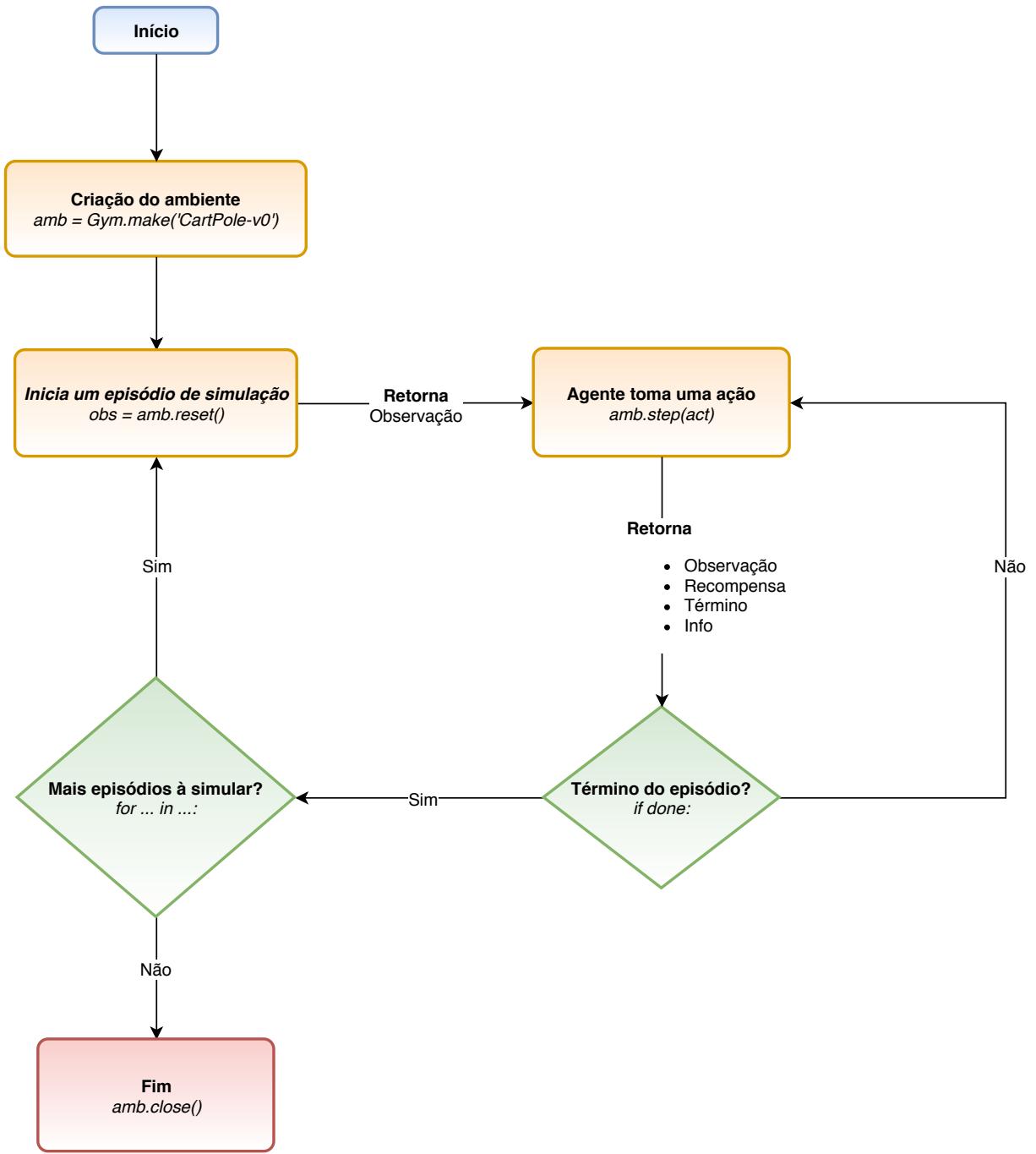


Figura 19: Forma geral de um algoritmo para realização de uma simulação no *OpenAi Gym*.

É realizado, a seguir, uma análise breve de cada objeto da interação demonstrada na Figura 19, com o objetivo de facilitar a criação do algoritmo.

O principal objeto é o retornado pela função `step`. Trata-se de uma tupla de quatro objetos, os quais o agente utilizará para obter informações sobre o sistema, quais sejam:

- *Observação*: lista indexável (Figura 20) que contém o valor das variáveis de estado do sistema logo após a *ação* do agente.

Observação

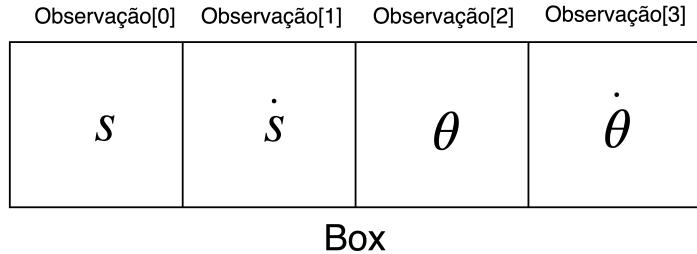


Figura 20: *Box* é uma classe que permite a criação de conjuntos multidimensionais, no ambiente do pêndulo invertido é apenas uma lista (unidimensional).

- *Recompensa*: valor retornado pelo sistema após ter sofrido uma *ação* do agente. Pode apresentar os valores 0 ou 1, para o ambiente do pêndulo invertido (Figura 21).

Recompensa

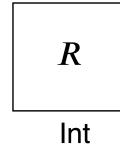


Figura 21: Recompensa recebida pelo agente.

- *Término*: valor lógico (Figura 22) que indica se o episódio de simulação terminou ou não, de acordo com os critérios estabelecidos.

Término

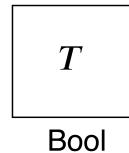


Figura 22: Valor lógico que indica se o episódio terminou ou não.

- *Info*: fornece informações adicionais sobre a simulação ao programador. Geralmente, é utilizada para *debugging*.

Ao longo deste capítulo foi possível estudar a implementação da simulação de sistemas dinâmicos na biblioteca *Gym*. Esta é uma etapa essencial para verificar o desempenho das soluções encontradas ao longo da execução do algoritmo. O próximo capítulo aborda outros aspectos da programação genética, através da biblioteca DEAP.

3 DEAP: IMPLEMENTAÇÃO DOS AGENTES

No capítulo anterior, foi abordada a utilização da biblioteca Gym para simular a interação de um sistema com um agente. Neste capítulo, o objetivo é verificar as funcionalidades providas pela biblioteca DEAP e como essas ferramentas foram utilizadas para representar indivíduos de uma população.

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 otimização por enxame de partículas [15].

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

Este capítulo descreve a implementação da população e dos indivíduos que a compõem, utilizando as ferramentas proporcionadas pela biblioteca. São analisados os parâmetros essenciais para a representação da solução, isto é, o tamanho da população, o número máximo de níveis das árvores, entre outros aspectos. Em seguida, é implementada a avaliação e seleção de indivíduos. Por fim, são definidos os operadores genéticos e o critério de término.

Já que a biblioteca DEAP será abordada ao longo deste capítulo, é proveitoso ter uma visão geral das funcionalidades de cada módulo da biblioteca, explicitados na Figura 23.

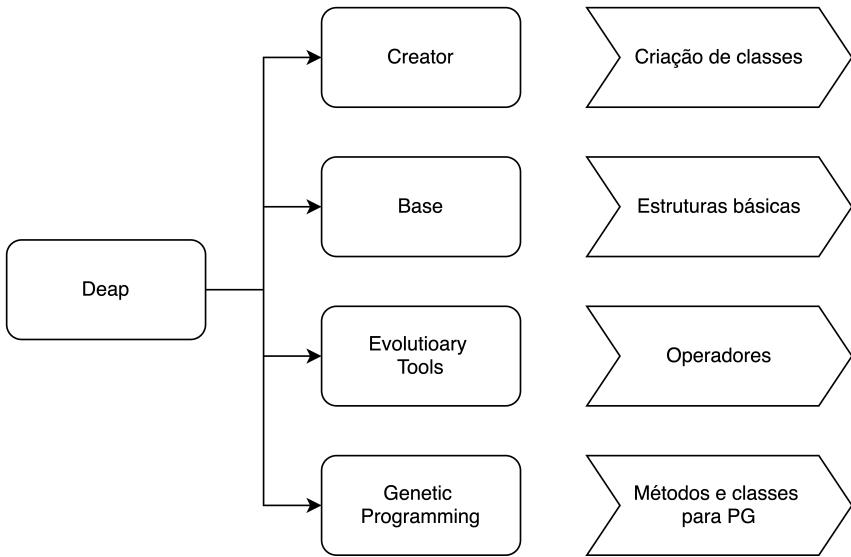


Figura 23: Módulos da biblioteca DEAP e suas funcionalidades.

3.1 REPRESENTAÇÃO E INICIALIZAÇÃO

No Capítulo 1, a lista dos principais parâmetros que definem uma implementação de PG é fornecida. Neste capítulo, foi realizado um estudo dos aspectos referentes à representação de cada indivíduo e sua respectiva inicialização, tornando possível a criação de uma população inicial aleatória pronta para sofrer transformações ao longo do ciclo evolutivo.

3.1.1 Representação

Para representar um indivíduo na PG deve-se definir: o conjunto de operações 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 compõem os indivíduos da população.

As variáveis terminais podem ser constantes ou entradas vindas do sistema. As entradas são informações sobre 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 operações que atuam sobre esses números podem moldar coeficientes de uma lei de controle ótima para o problema.

Para o estudo de caso básico (pêndulo invertido), as variáveis terminais utilizadas

estão descritas na Tabela 2.

Tabela 2: Variáveis terminais.

Variável Terminal	Significado	Tipo
s	Posição do carro	Entrada
\dot{s}	Velocidade do carro	Entrada
θ	Ângulo do bastão	Entrada
$\dot{\theta}$	Velocidade angular do bastão	Entrada
R	Número gerado aleatoriamente ($-1,0 < R < 1,0$)	Constante

Os operadores utilizados estão descritos na Tabela 4, e foram escolhidos com base na literatura abordada ([9], [1] e [16]).

Tabela 3: Conjunto de operadores lógicos e matemáticos.

Operador	Símbolo	Resultado	Aridade
Soma	+	$a + b$	2
Subtração	-	$a - b$	2
Multiplicação	\times	$a \cdot b$	2
Divisão	\div	a/b	2
Raiz Quadrada	$\sqrt{}$	\sqrt{a}	1
Seno	sin	$\cos(a)$	1
Comparação	>	$a, \text{ se } a \geq b$ $b, \text{ se } a < b$	2
Sinal	sgn	$1, \text{ se } a \geq 0$ $-1, \text{ se } a < 0$	1

Tabela 4: Conjunto de operadores lógicos e matemáticos.

Operador	Símbolo	Resultado	Aridade
Soma	+	$a + b$	2
Subtração	-	$a - b$	2
Multiplicação	\times	$a \cdot b$	2
Divisão	\div	a/b	2
Raiz Quadrada	$\sqrt{}$	\sqrt{a}	1
Seno	sin	$\cos(a)$	1
Comparação	>	$a, \text{ se } a \geq b$ $b, \text{ se } a < b$	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. Neste caso, o valor retornado será igual a 1.

Um problema se apresenta ao analisar uma possível lei de controle gerada através das funções da Tabela 4 com as variáveis terminais da Tabela 2: os valores que resultam da avaliação da expressão são contínuos. O ambiente do pêndulo invertido impõe uma restrição às ações, através do objeto *action space*. Mais especificamente, estas devem ser valores discretos: 0 ou 1.

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 contrário. Este processo pode ser observado na Figura 24.

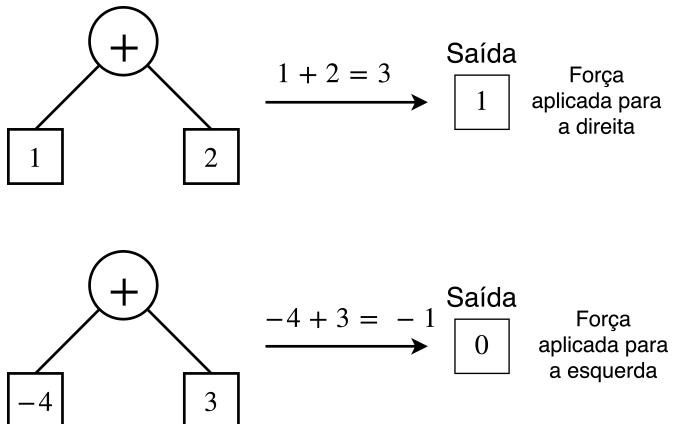


Figura 24: A saída depende do sinal da expressão que resulta da avaliação do indivíduo.

Com isso, fica claro que a ação realizada pelo indivíduo está contida no espaço de ações possíveis, determinado pelo ambiente de simulação.

É necessário agrupar as operações e variáveis terminais em um conjunto único. O DEAP 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. Geralmente, *arity* será igual ao número de variáveis fornecidas na observação de estado da simulação. É possível adicionar um prefixo *prefix* à nomeação de cada entrada.

O nome *main* foi dado ao conjunto primitivo criado, por convenção. Para o problema do pêndulo invertido, por exemplo, a aridade do conjunto primitivo é quatro, isto é, o número de variáveis de estado. 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 4 e a constante efêmera da Tabela 2. As outras variáveis terminais, que correspondem às leituras dos sensores, são definidas como **entradas** e são adicionadas na criação do conjunto \mathcal{P} .

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

Adiciona uma operação lógica/matemática *op* ao conjunto primitivo, sob o nome *name*. O parâmetro *arity*, neste caso, define o número de argumentos da função.

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

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

A seguir, um atributo de aptidão é adicionado ao indivíduo em sua inicialização.

3.1.2 Aptidão

Ao longo da Capítulo 1, foi visto que existem diversos métodos de representar a aptidão de um indivíduo. Em problemas de controle, é comum expressar esse critério de desempenho como funções custo, semelhantes à Equação 2. Um indivíduo apto, portanto, seria caracterizado por um valor baixo de aptidão, ou seja, desejaríamos minimizar esta quantidade. Entretanto, seria possível elaborar uma função de aptidão a ser maximizada. Para o problema do pêndulo invertido, por exemplo, basta que essa função represente o tempo total do episódio de simulação atingido por um indivíduo. Dessa forma, uma solução com desempenho insatisfatório causaria rapidamente o fim do episódio de simulação (pois o critério de término está relacionado à instabilidade do sistema), e consequentemente, um valor de aptidão baixo seria atribuído a este indivíduo.

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 evolutivo é gerar indivíduos com a aptidão maior possível.

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 desta quantidade. Além disso, este componente permite implementar funções de aptidão com múltiplos objetivos.

A biblioteca conta também com uma função de criação de classes, a partir de

outras, denominada *create* da biblioteca *creator*.

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

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

Já que o ambiente de simulação proporcionado pela biblioteca Gym provê uma recompensa unitária a cada intervalo de tempo discreto em que o episódio não termina, é possível calcular a aptidão de um indivíduo como a soma de recompensas acumulada ao longo do tempo. Como a busca é por indivíduos com o máximo de aptidão, essa classe pode ser nomeada *AptidaoMax*.

3.1.3 Inicialização

Os indivíduos de uma população possuem uma estrutura semelhante a uma árvore, cujo conteúdo pertence a um conjunto primitivo de operadores e variáveis terminais 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, restando, 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 à profundidade das árvores representando os indivíduos e o método de inicialização, ou seja, como essas árvores seriam geradas.

No Capítulo 1 foi definido 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.

Foram escolhidos os valores 2 e 5 para D_{\min} e D_{\max} , respectivamente, para o problema do pêndulo invertido. Na Figura 25 pode-se ver alguns exemplos de indivíduos inicializados com valores extremos de profundidade da árvore.

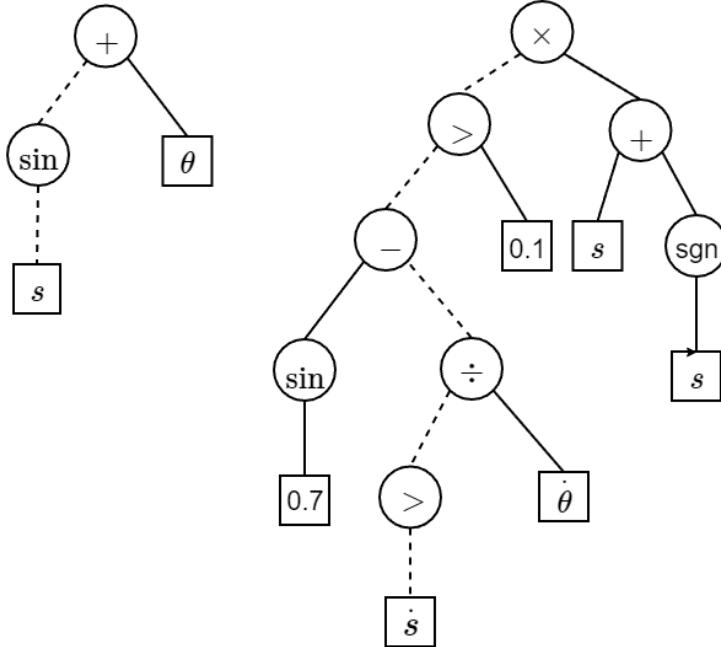


Figura 25: As linhas pontilhadas indicam a maior distância entre a *raiz* e uma *variável terminal*. Este exemplo utiliza o conjunto primitivo \mathcal{P} definido anteriormente.

Pode-se notar que uma expressão pode conter diferentes profundidades, com o valor entre os extremos pré-estabelecidos. Por exemplo, na Figura 25, o indivíduo da direita possui caminhos com comprimento mínimo apesar de conter um caminho de profundidade máxima. Se partirmos do princípio de que a criação dos indivíduos se 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 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} . Este método de inicialização é chamado de *full* [1]. Os indivíduos também podem ser gerados com caminhos de comprimento aleatório (dentro da faixa estipulada), isto é: a única restrição é que cada caminho deveria ter um comprimento l_i tal que $D_{\min} \leq l_i \leq D_{\max}$. Este método é conhecido como *grow*.

Koza [1] sugere um método híbrido, frequentemente empregado na literatura abordada, denominado *ramped half-and-half*. Nessa abordagem são gerados N indivíduos para cada nível máximo de profundidade $D'_{\max} = D_{\min} \dots D_{\max}$, 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 apenas a rotina

ramped half-and-half será utilizada, já que provê grande diversidade de estruturas iniciais. Este método é implementado 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.

Os indivíduos são representados, 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 à 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.

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, apenas um parâmetro será abordado, no momento: o número de indivíduos. É notável a importância deste parâmetro para a evolução de programas [1].

Em geral, a dificuldade de determinação do tamanho ótimo da população aumenta com a complexidade do problema. Em contrapartida, a eficiência da programação genética 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 os resultados obtidos serão 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 a função *register* do módulo *evolutionary tools*, que auxilia na criação de funções com argumentos fixos.

Mais especificamente, a classe *Toolbox* permite agrupar diversas funções em um único objeto. É possível, então, fornecer um objeto desta classe (que efetivamente significa “caixa de ferramentas”) a um algoritmo que implementa o ciclo genético de forma completa ou parcial.

Função register (*alias, fun, args*)

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

Se o *alias* da rotina registrada corresponder a **mate** ou **mutate**, está indicado que esta função implementa um cruzamento ou mutação, respectivamente. Alguns algoritmos que realizam o ciclo genético iterativo buscam dentro do objeto *Toolbox* as funções chamadas *mate* ou *mutate*, ao realizar as operações de cruzamento ou mutação.

As expressões recursivas geradas com a função *genHalfAndHalf* 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 listas (como o método *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*, ao chamar a função de geração de indivíduos 500 vezes.

Com isso, todas as funções necessárias para inicializar uma população completa foram abordadas.

3.2 AVALIAÇÃO DE INDIVÍDUOS

Com a criação da população inicial abordada ao longo do Capítulo 3.1, é necessário avaliar cada indivíduo de acordo com uma função de aptidão. 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 aptidão utiliza os resultados da interação do indivíduo com o sistema dinâmico, através da simulação. De maneira geral, a partir dos conceitos apresentados na Figura 19, a função de avaliação utiliza, como um dos argumentos, uma observação do sistema. 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 à referência, $\theta = 0$.

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 da função de aptidão para os ambientes da biblioteca Gym, é possível definir:

$$A(t) = f_a(O(t), r(t)) \quad (5)$$

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

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

$$A(t) = f_a(r(t)) = 1, \forall t < T \quad (6)$$

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

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.

No Capítulo 2.2, foi visto que os valores iniciais das variáveis de estado são aleatórios e restritos a uma faixa específica. Caso o cálculo da aptidão de um indivíduo seja realizado com base em apenas uma simulação, é possível que o valor atribuído não reflita a real capacidade da lei de controle em relação à todas as situações iniciais possíveis. Realizar a simulação em diversos episódios auxilia na convergência da busca por um 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, é possível considerar, por exemplo, a utilização de 10 episódios para a avaliação de um indivíduo. Nesse caso, a aptidão de uma solução será uma média, conforme indica a Equação 7.

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

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

(7)

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 $O(t)$. Por exemplo, a Equação 8 representa uma função de controle genérica de um indivíduo, para o problema do pêndulo invertido.

$$f_c(t) = f_c(O(t)) = f_c(s, \dot{s}, \theta, \dot{\theta}) \quad (8)$$

$O(t)$ 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 *gp* 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 evolutivo, para cada indivíduo.

3.3 SELEÇÃO DE INDIVÍDUOS

Ao longo do Capítulo 1 foi mencionado o papel da seleção realizada na população. De forma geral, esse processo busca inspiração na seleção natural descrita por Darwin. Ao longo deste capítulo, o objetivo é analisar as principais formas de implementar este procedimento na população da PG.

Um dos principais conceitos referentes ao método de seleção é a *pressão de seleção*, que indica o quão favorecidos os indivíduos mais aptos serão. Isto é, quanto maior a pressão de seleção, mais discriminante será o ambiente, o que pode levar a 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 que será utilizado.

O método chamado de *torneio* foi descrito brevemente na Capítulo 1. São escolhidos aleatoriamente dois ou mais indivíduos na população e suas aptidões são comparadas. É possível selecionar n indivíduos; dessa forma, o torneio apresenta diversos níveis e o indivíduo de melhor desempenho será 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 exerce baixa pressão de seleção, gerando populações diversificadas.

O DEAP promove o método descrito acima com o controle de um processo indesejável, chamado de *bloat*. Esse fenômeno é caracterizado por um crescimento excessivo do comprimento dos indivíduos, sem retorno significativo em termos de desempenho [16]. O método de seleção *selDoubleTournament* promove dois torneios: um baseado na aptidão e outro no tamanho dos indivíduos, de forma que os membros de menor comprimento tenham chances maiores de serem selecionados para participar do torneio de aptidão. A ordem em que cada tipo de comparação ocorre aparenta não ter significância [15]. Segue

uma descrição da função selDoubleTournament:

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

Seleciona k indivíduos de uma lista $indList$ a partir de um torneio com $fitSize$ indivíduos. Caso $fitFirst$ seja falso, participantes do campeonato de aptidão serão selecionados após um a comparação de dois indivíduos aleatórios da população, em que o menor será selecionado com uma probabilidade $lenSize/2$, quando $lenSize \in [1,2]$

Seguindo o mesmo conceito da implementação da função de avaliação, a função de seleção é registrada na caixa de ferramentas, sob o nome *select*.

3.4 OPERADORES GENÉTICOS

Este capítulo discute a aplicação dos operadores genéticos (cruzamento, mutação e replicação) nos indivíduos selecionados. Além disso, são 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 [9], 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 no Capítulo 1 de busca global e local, respectivamente.

3.4.2 Mutação

A mutação é um operador genético eficiente e sua importância foi demonstrada por Kotzing et al. [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 utilizada começa selecionando um ponto aleatório dentro da árvore

de representação do indivíduo. Tal ponto pode ser uma função ou variável terminal. Ocorre a substituição do nó por um ramo gerado aleatoriamente por uma função.

A função utilizada para gerar essas expressões, descrita a seguir, é igual à utilizada para inicializar indivíduos; a mudança reside nos comprimentos 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*.

A partir da classe Toolbox, abordada no Capítulo 3.1.4, é possível 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 as iterações do ciclo evolutivo, é proveitoso nomeá-la *mutate*.

Como uma ferramenta adicional para o controle de *bloat*, é possível alterar o funcionamento da função descrita acima para descartar automaticamente 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, portanto, eliminar indivíduos mutantes cujo comprimento máximo ultrapasse um limite 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], este 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 Figura 13, é implementado na biblioteca através da seguinte 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.

Através do mesmo princípio de implementação da mutação, a operação de cruzamento é adicionada à caixa de ferramentas, sob o nome de *mate*. O controle de bloat, com o mesmo limite estático de comprimento dos indivíduos, também é implementado.

3.4.4 Seleção Aleatória do Operador

Não foi mencionado até o momento, em termos práticos, 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 à aplicação dos operadores. A primeira, objeto de estudo deste trabalho, funciona exatamente como descrito na Figura 15: 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 apenas 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 alteração.

Para a abordagem *varOr* 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 a probabilidade de replicação será $1 - P_c - P_m$.

3.5 CICLO ITERATIVO

Ao longo deste capítulo 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. Este capítulo trata da interação desses componentes, em um processo iterativo, com o objetivo de implementar o ciclo evolutivo completo.

Basicamente, o propósito é implementar o processo descrito no fluxograma da Figura 5. A biblioteca fornece algoritmos que realizam o processo iterativo evolutivo, utilizando as funções registradas na classe Toolbox. Esses algoritmos se encontram no módulo *Algorithms* e permitem a simplificação do código criado. A função que se assemelha ao ciclo descrito na Figura 15 é a *eaMuPlusLambda*, descrita a seguir:

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* (variação do indivíduo selecionado com uma das operações genéticas).

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, através dos argumentos *cxbp* e *mutpb*, que indicam, respectivamente, as chances de ocorrência das operações de cruzamento e mutação.

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*) auxiliam a obtenção de estatísticas relacionadas ao processo evolutivo artificial.

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

4 ESTUDOS DE CASO

Ao longo do Capítulo 3 foram abordados os principais aspectos da implementação da PG, através da biblioteca DEAP. Principalmente, a implementação da: representação e inicialização de indivíduos, avaliação, seleção e alteração dos membros da população. O que se espera de cada iteração da programação genética são valores crescentes de aptidão.

Diversos problemas são abordados ao longo deste capítulo. Para cada caso, uma tabela é apresentada 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, é 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 indica que o objetivo é maximizar a função de aptidão. O valor -1 aponta o contrário.
- *Número de entradas (n_entradas)*: número de variáveis de estado.
- *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 torneio de aptidão (camp_apt)*: número de indivíduos que irão compor o torneio de aptidão.

- *Tamanho do torneio de comprimento* (*camp_d*): pressão de seleção sobre indivíduos aspirantes ao torneio de aptidão (valor entre 1 e 2).
- *Operações*: operações que compõe os nós não-terminais da árvore.
- *Comprimento mínimo e máximo de inicialização* (*d_min, d_max*): comprimentos mínimos e máximos possíveis dos indivíduos inicializados.
- *Comprimento máximo de mutação* (*max_d_mut*): A função geradora de expressões será diferente para a mutação. O comprimento mínimo será sempre zero, entretanto o máximo irá variar, dependendo da complexidade da solução esperada.
- *Limite de comprimento dos indivíduos* (*limite_d*): comprimento máximo dos indivíduos gerados através das operações de mutação e cruzamento (controle de bloat).

As seguintes estatísticas relacionadas à evolução da população e à execução do programa são obtidas:

- a) Em relação à aptidão: este 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*: valor médio de aptidão da população.
- b) Em relação ao comprimento: Medida que indica o maior comprimento (ou profundidade) da árvore, conforme aponta a Figura 25. As medidas relacionadas a 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: esta medida indica o número total de nós em um indivíduo (operadores e variáveis terminais), em 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.

Foi realizada uma comparação da PG com dois algoritmos de aprendizagem profunda 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 exemplo, para a aplicação da PG, foi o pêndulo invertido. Os parâmetros utilizados apresentam-se na Tabela 5.

Tabela 5: Parâmetros da PG 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, 1)
Número de Simulações	10
Tamanho do Torneio de Aptidão	6
Tamanho do Torneio de Comprimento	1,2
Operações	$+, -, \times, \div, \sqrt{ }, \sin, >, \text{sgn}$
Comprimento Mínimo e Máximo de Inicialização	(1, 3)
Comprimento Máximo de Mutação	5
Limite de Comprimento dos Indivíduos	17

A função de cálculo de aptidão, é tal como descrita na Equação 6, isto é, a média do tempo total em que o bastão permanece equilibrado, em vários episódios, respeitando os limites estabelecidos na Tabela 1. Já que a simulação tem uma duração limite de 500 passos de tempo, a aptidão máxima possível é 500. Como as condições iniciais são aleatórias, a medida de desempenho é realizada através da aptidão média obtida em vários episódios (parâmetro representado pelo número de simulações).

Utilizando os dados da Tabela 5, o algoritmo foi executado 10 vezes em sequência e a média das estatísticas foram obtidas. O algoritmo completo encontra-se no apêndice A.

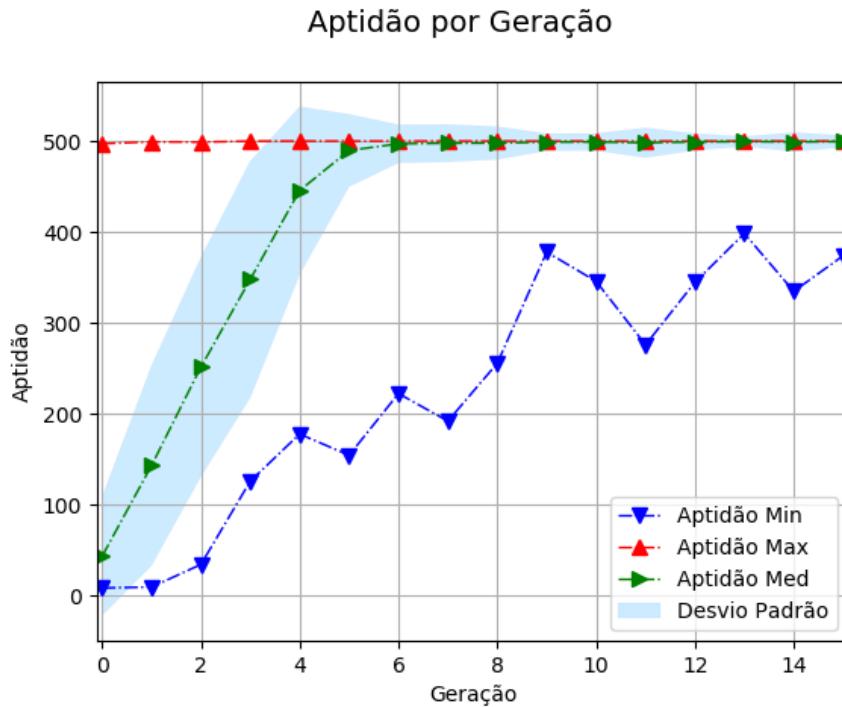


Figura 26: Média da aptidão de todos os indivíduos por geração (verde). Valor máximo de aptidão observado em cada geração (vermelho). Menor aptidão observada em cada geração (azul).

Observa-se no gráfico da Figura 26, que na primeira geração alguns indivíduos já são capazes de obter a aptidão máxima. Isto se deve, em parte, ao grande número de indivíduos (500) gerados aleatoriamente, o que funciona, de certa forma, como uma busca exaustiva.

Entretanto, ainda é possível observar o aumento da aptidão média e mínima ao longo das gerações. O gráfico da Figura 27 mostra o número de indivíduos que atingiram uma determinada faixa de aptidão, em algumas gerações.

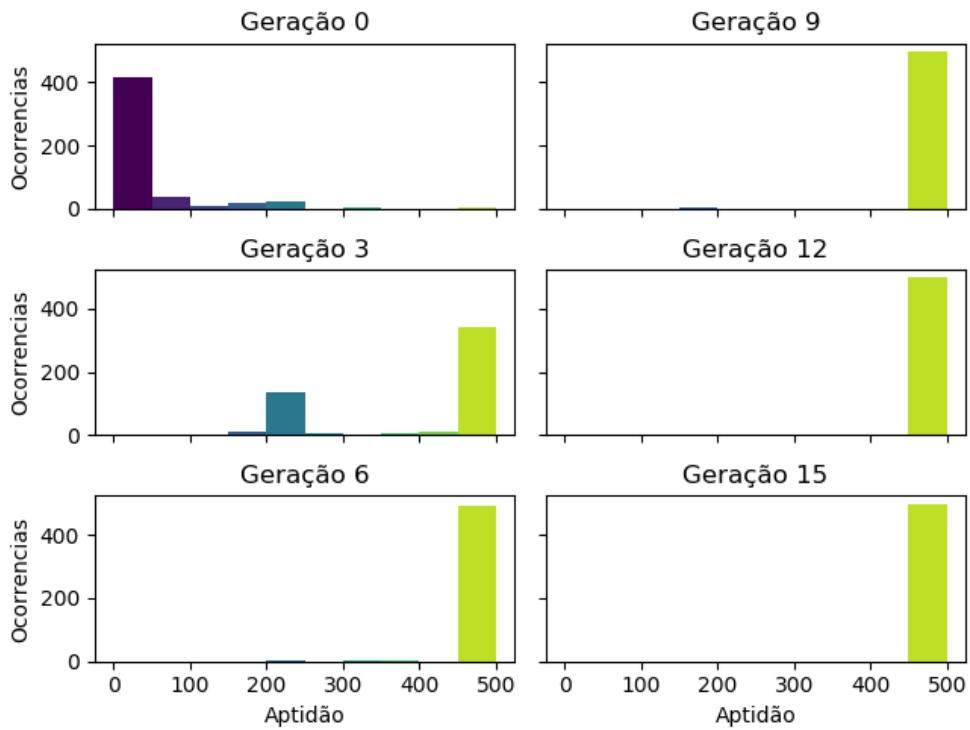


Figura 27: Número de indivíduos, em algumas gerações, que obtiveram cada faixa de aptidão.

Conforme visto no Capítulo 3.4, existe uma tendência de aumento do comprimento dos indivíduos ao longo do processo. O gráfico da Figura 28 mostra a média do comprimento de cada indivíduo em cada geração, assim como os valores mínimos e máximos observados. Já que todas as estatísticas foram obtidas através do valor médio em 10 execuções do algoritmo, o gráfico contém valores fracionários.

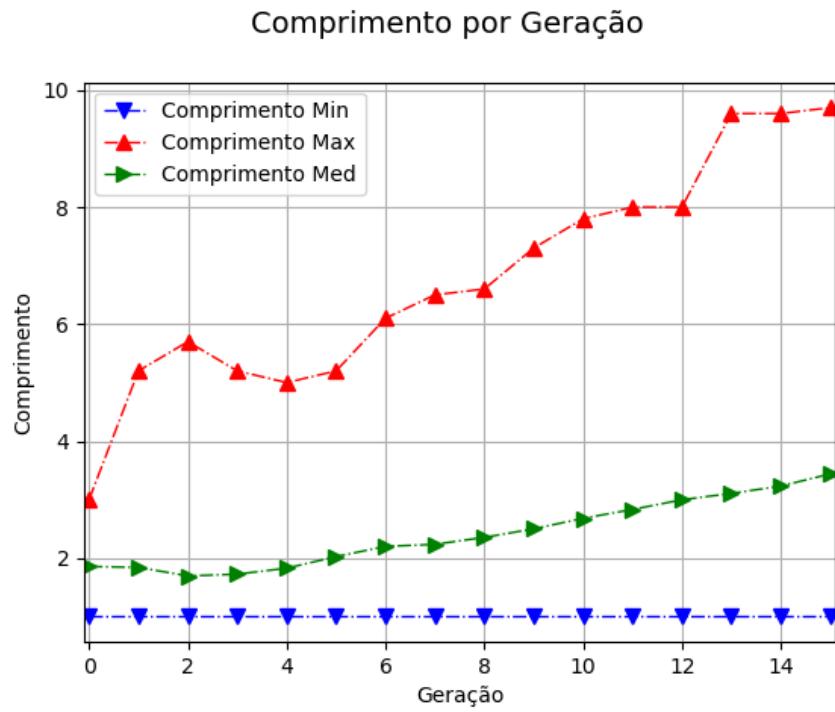


Figura 28: Comprimento dos indivíduos por geração.

A complexidade dos indivíduos ao longo do processo pode ser observada no gráfico da Figura 29.

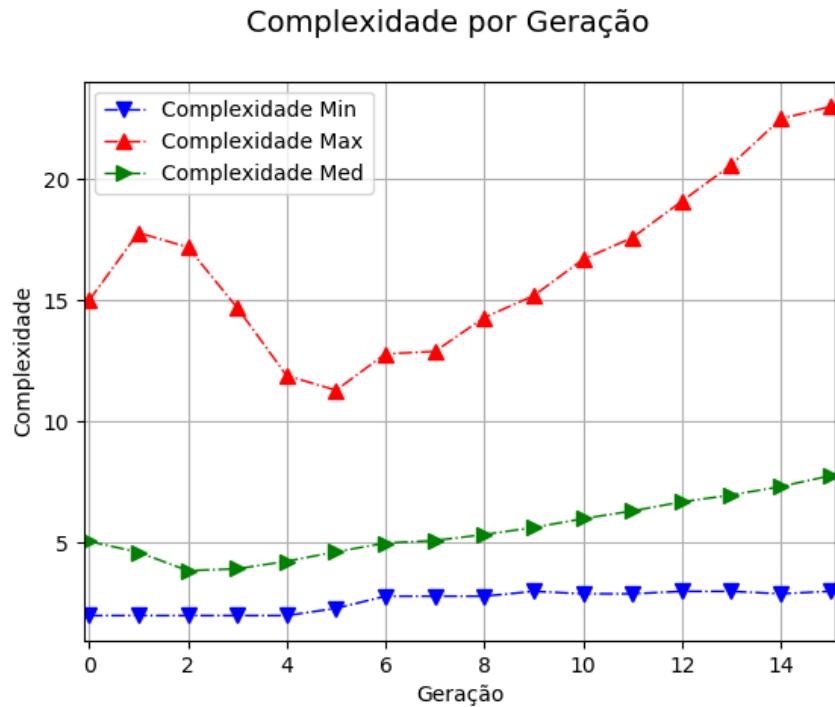


Figura 29: Complexidade dos indivíduos em cada geração.

Foi possível observar que, alguns operadores e variáveis, pertencentes ao conjunto

primitivo, são mais eficientes para a resolução 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 pertencente à geração final, o resultado pode ser verificado no gráfico da Figura 30.

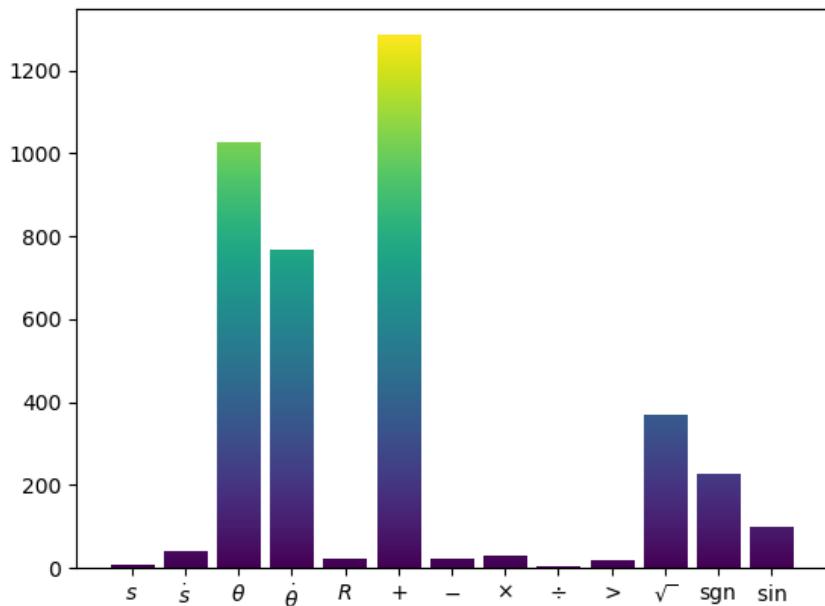


Figura 30: Número de ocorrências dos operadores e variáveis terminais, nos indivíduos da última geração.

Através do objeto *hall da fama*, foi 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 31, 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 31 tem prioridade sobre as outras. O pequeno comprimento do indivíduo indica que, de fato, pertence às primeiras gerações.

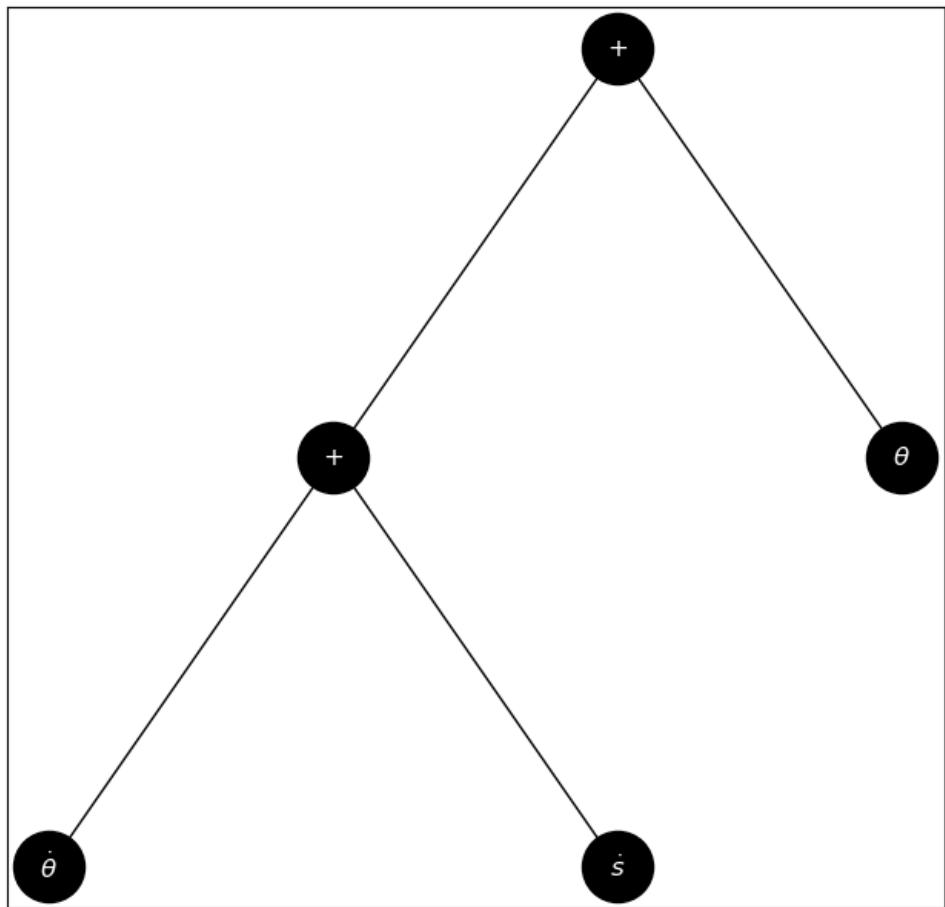


Figura 31: Primeiro indivíduo do hall da fama, na primeira execução do algoritmo.

É possível perceber indivíduos de maior comprimento, nas posições finais do hall da fama, como, por exemplo, na Figura 32.

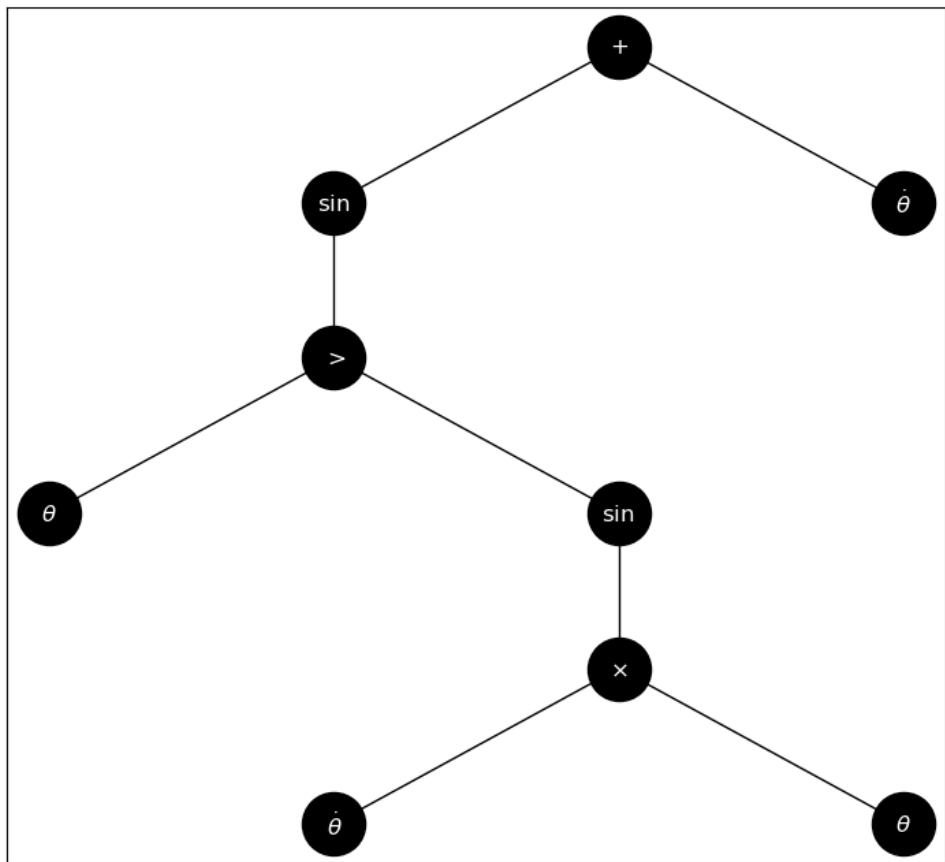


Figura 32: Vigésimo terceiro indivíduo do hall da fama, na primeira execução do algoritmo.

Já que múltiplos indivíduos obtiveram um bom desempenho em cada geração, incluindo as iniciais, é interessante notar o comprimento desses indivíduos mais aptos, especificamente. Pode-se observar que a tendência de aumento do comprimento dos indivíduos ao longo das gerações é preservada.

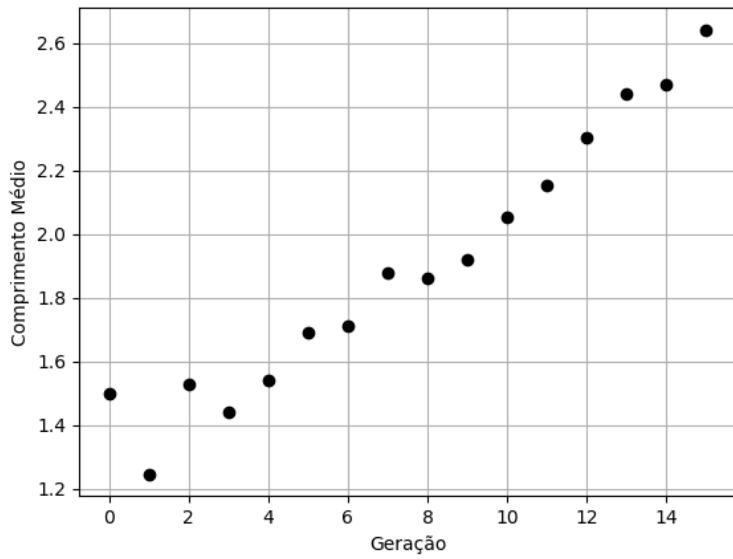


Figura 33: Comprimento médio dos indivíduos que obtiveram aptidão maior que 400.

A Figura 34 mostra os gráficos relacionados à atuação do indivíduo da Figura 31, em um único episódio. O eixo *ação* indica o controle aplicado no sistema, a partir do *resultado* obtido no cálculo da expressão matemática que representa um indivíduo.

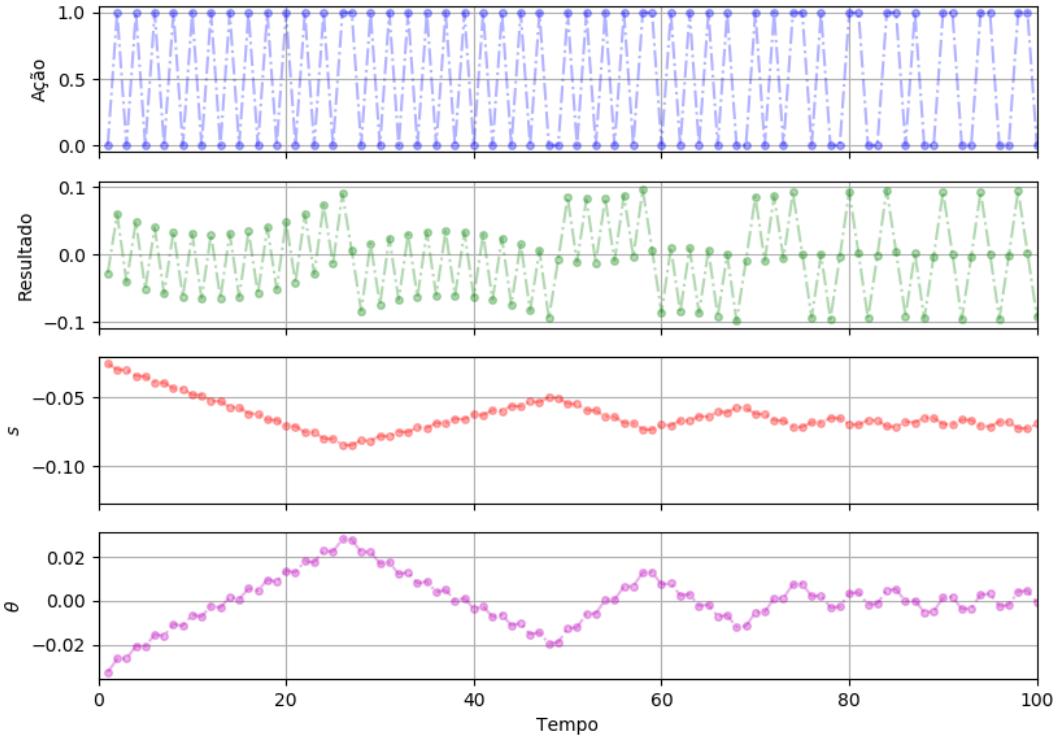


Figura 34: Controle do sistema pelo indivíduo da Figura 31. O pêndulo é levado rapidamente a um estado de baixa oscilação.

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 levava 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 um 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, busca-se o indivíduo mais apto ao avaliar cada membro do hall da fama em 100 simulações.

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 a comparação desses resultados com a abordagem proposta pelo algoritmo DQN, uma vez que o critério de desempenho é o mesmo, isto é, a média de recompensa acumulada em vários episódios. O tempo médio de execução do algoritmo de PG foi 334s. O agente DQN é treinado, aproximadamente, pelo mesmo tempo. Em seguida, as recompensas médias acumuladas por episódio, em 100 simulações são comparadas. Destaca-se que a avaliação da PG foi realizada pelo indivíduo mais apto do hall da fama, quando cada membro é submetido à 100 episódios.

Utilizando a biblioteca *stable-baselines* [22], o agente DQN foi treinado por 334s (código encontra-se no apêndice). O gráfico da Figura 35 mostra a recompensa obtida pelo agente ao longo do treinamento.

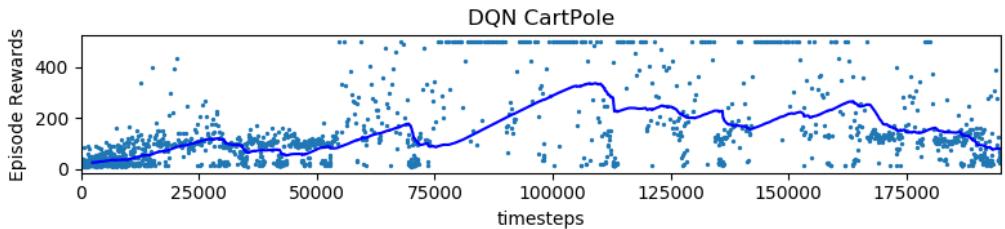


Figura 35: Recompensa média acumulada em função do número de passos de simulação. A linha contínua representa a média móvel.

Na Figura 35 é possível notar a degradação da recompensa média acumulada a partir dos 110000 passos de simulação. Já que esse comportamento foi observado, o algoritmo foi executado novamente com o número de passos totais reduzido. O resultado pode ser verificado na Figura 36.

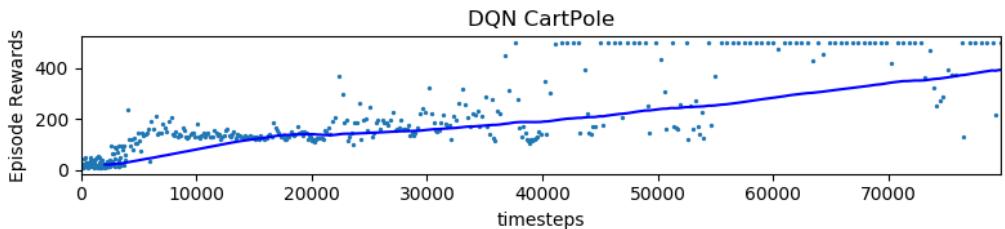


Figura 36: Recompensa média ao longo dos passos de simulação. A linha azul contínua indica a média móvel.

A Tabela 6 sumariza uma breve comparação entre o desempenho da programação genética com o algoritmo DQN, em termos de custos computacionais.

Tabela 6: Comparação entre a programação genética e DQN, para o problema do 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 37 mostra a atuação do agente DQN, ao longo de um episódio.

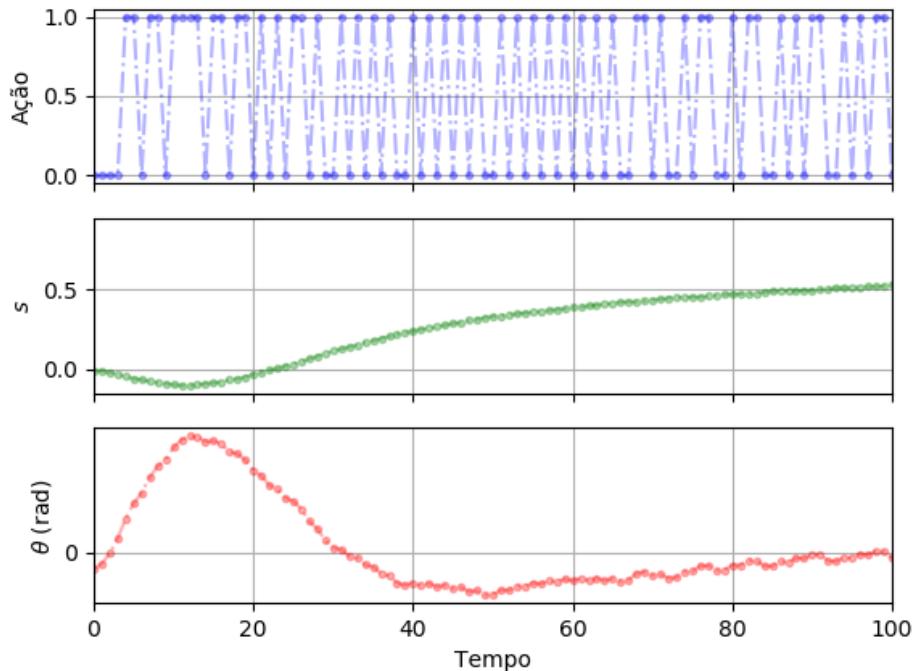


Figura 37: Posição do carrinho e ângulo do bastão, em função do tempo, com a atuação do agente DQN.

Observa-se, 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.

¹Valor estimado.

4.2 Pêndulo *Swing-up*

O próximo sistema abordado está disponível na biblioteca Gym, na seção *classic control*, direcionada à implementação de ambientes de simulações para problemas clássicos de controle. Conforme a abordagem do problema anterior, são 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 é abordada.

A dinâmica envolve um bastão com uma de suas extremidades fixas, 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. A Figura 38 ilustra o problema.

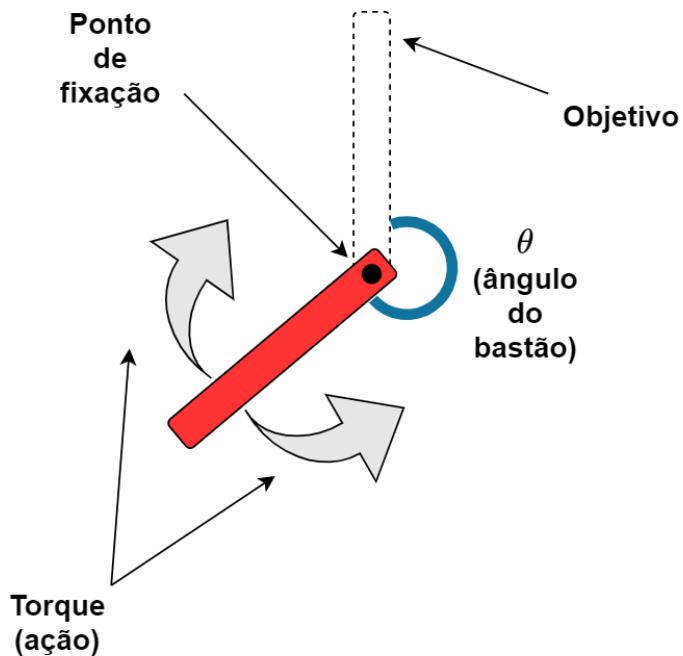


Figura 38: Pêndulo *Swing-up*.

A seguir são enumerados alguns aspectos que aumentam a complexidade do problema, além de 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 permite 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 7 apresenta as variáveis que indicam o estado do sistema e 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], isto ocorre após 200 passos de simulação.

Tabela 7: Variáveis que compõem a observação do 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 do Capítulo 3.2, é criada uma função de aptidão para o indivíduo. A recompensa a cada instante de tempo é uma função custo, segundo a Equação 9.

$$r(t) = - \left[(\theta(t))^2 + 0,1(\dot{\theta}(t))^2 + 0,001(a(t))^2 \right] \quad -\pi \leq \theta \leq \pi \quad (9)$$

$$- 2 \leq a(t) \leq 2$$

A variável $a(t)$ representa a ação (torque) do agente, no instante de tempo t . Foi mencionado no Capítulo 1 a possibilidade de projetar a função de aptidão utilizando uma penalização por desvios de um estado de referência. Desta forma, o desempenho de um indivíduo é dado pela soma dos custos (implementados como recompensas negativas) em vários episódios:

$$A(t) = r(t) = - \left[[\theta(t)]^2 + 0,1 [\dot{\theta}(t)]^2 + 0,001 [a(t)]^2 \right]$$

$$A_{tot}^{ep} = \sum_{t=0}^T A(t) = - \sum_{t=0}^T \left[[\theta(t)]^2 + 0,1 [\dot{\theta}(t)]^2 + 0,001 [a(t)]^2 \right] \quad T \leq 200 \quad (10)$$

$$\bar{A} = \frac{1}{nep} \sum_{ep=1}^{nep} A_{tot}^{ep} = - \frac{1}{nep} \sum_{ep=1}^{nep} \sum_{t=0}^T \left[[\theta(t)]^2 + 0,1 [\dot{\theta}(t)]^2 + 0,001 [a(t)]^2 \right]$$

De forma maneira similar ao problema anterior, o objetivo é maximizar a aptidão média na Equação 10.

Buscando demonstrar a robustez da PG, poucas mudanças foram realizadas nos parâmetros da Tabela 5, mais especificamente, foram alterados: o número de entradas, comprimentos de inicialização e o comprimento máximo de mutação.

Tabela 8: 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, 1)
Número de Simulações	10
Tamanho do Campeonato de Aptidão	6
Tamanho do Campeonato de Comprimento	1,2
Operações	+, -, ×, ÷, √, sin, >, 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 do Capítulo 4.1 utilizava ações discretas.

Logo, bastava mapear os valores que resultavam da compilação das árvores em ações: números positivos produziam uma força para a direita no carrinho, enquanto os negativos geravam uma ação no 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 aos limites de torque, definidos na Equação 9. Isto pode ser concebido pela utilização da função *clip*, cujo comportamento é demonstrado na Figura 39.

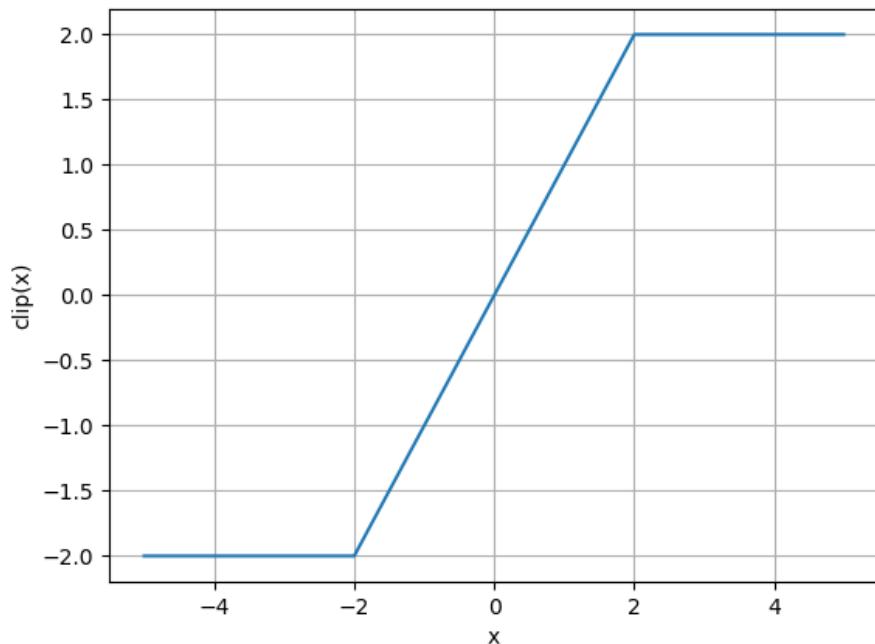


Figura 39: Função *clip*.

Essa função funcionará como *wrapper*, isto é, um mapeamento de um valor numérico em uma ação admissível.

Novamente, o algoritmo foi executado 10 vezes e a média das estatísticas 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 pôde obter um resultado satisfatório, como no problema anterior. As próximas gerações encontram uma solução eficaz para o pêndulo, conforme pode ser visto nas Figuras 40 e 41.

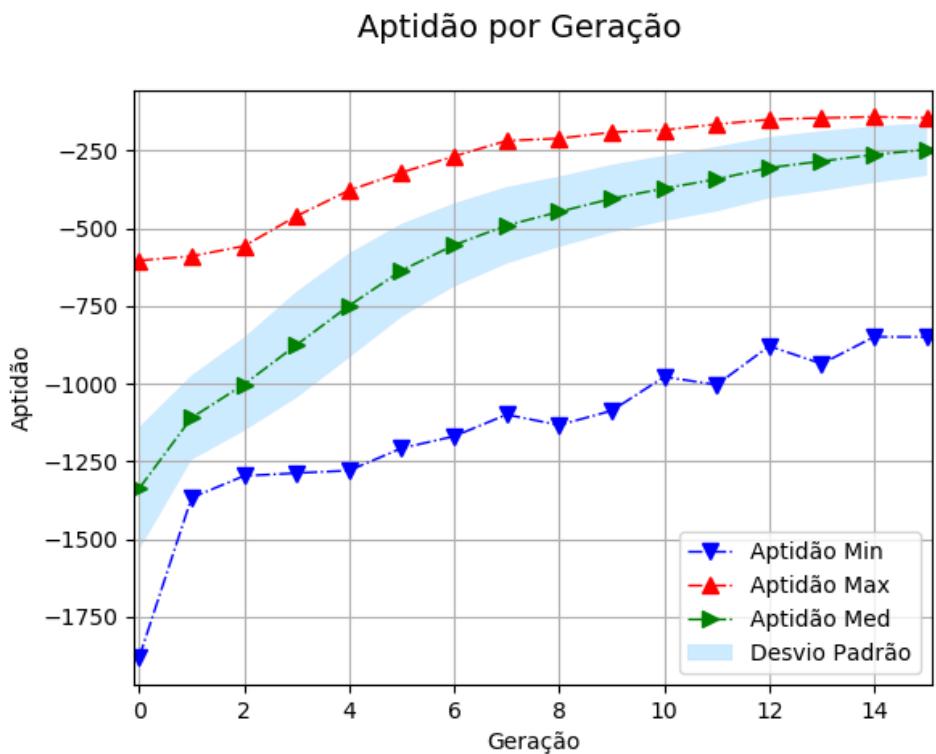


Figura 40: Aptidão dos indivíduos, ao longo das gerações.

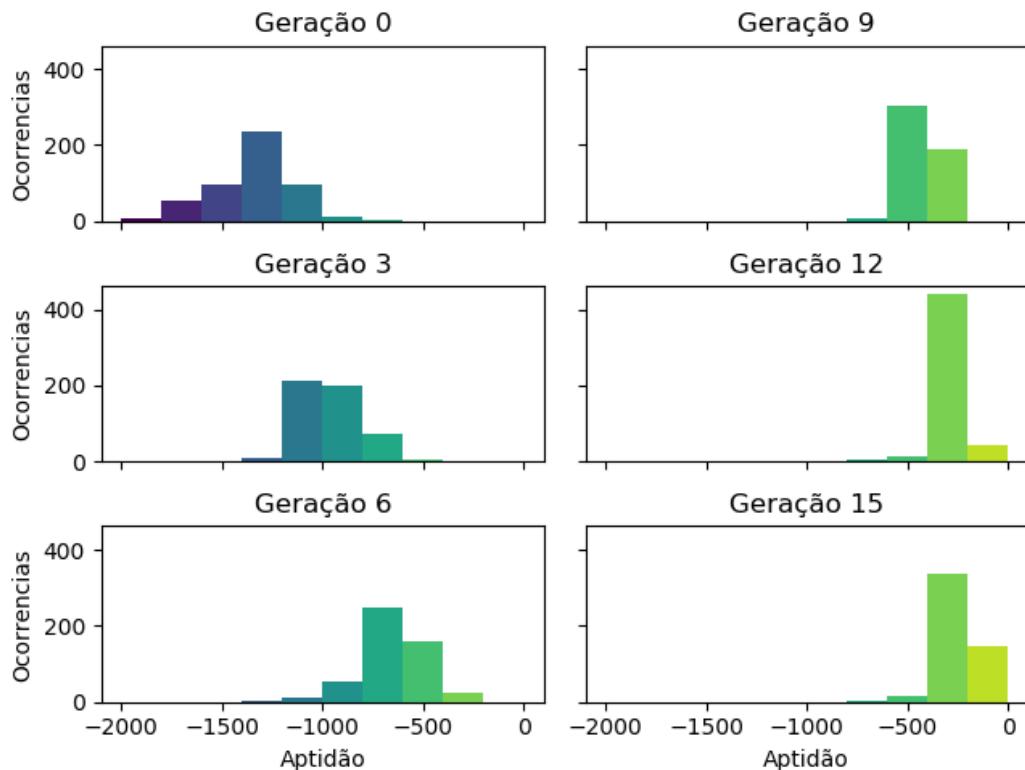


Figura 41: Histograma da aptidão dos indivíduos.

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

gerações, podem ser vistos nas Figuras 42 e 43, respectivamente.

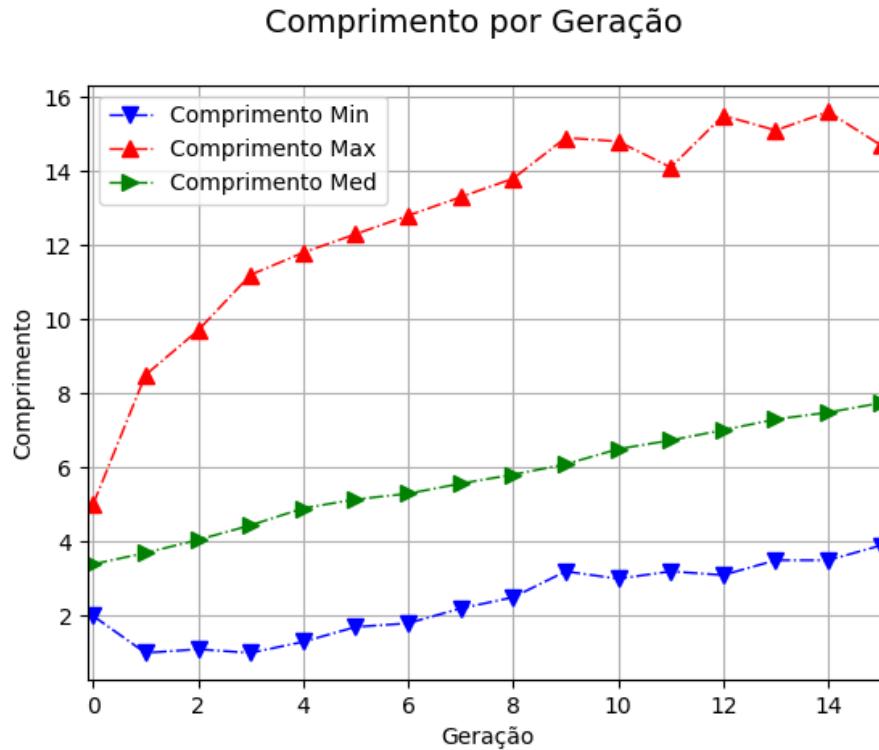


Figura 42: Comprimento dos indivíduos.

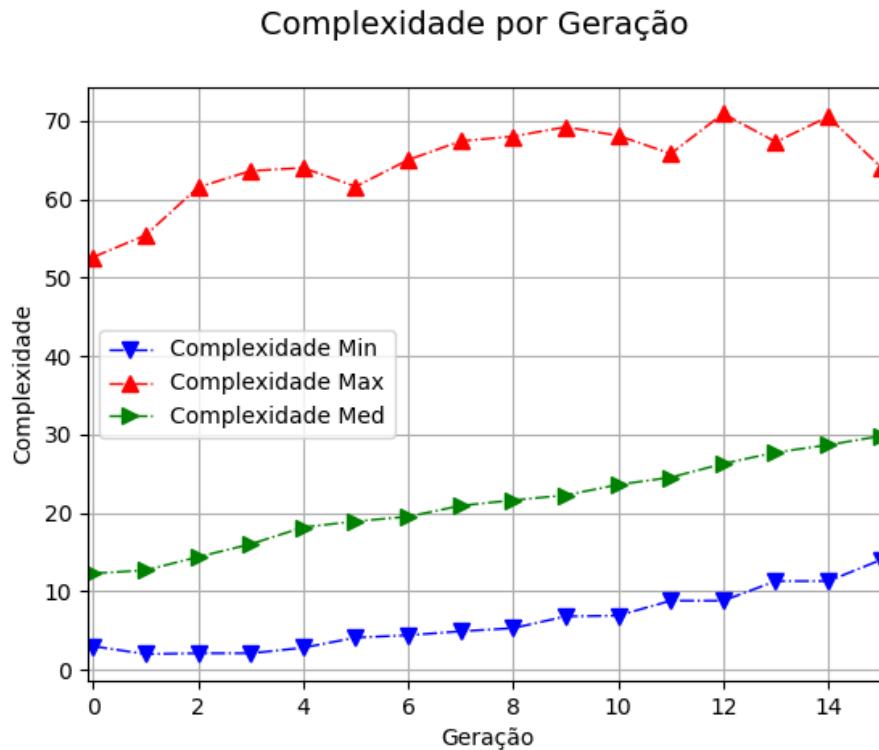


Figura 43: Complexidade dos indivíduos.

A Figura 44 mostra o número de ocorrências dos operadores e variáveis terminais, na população da última geração.

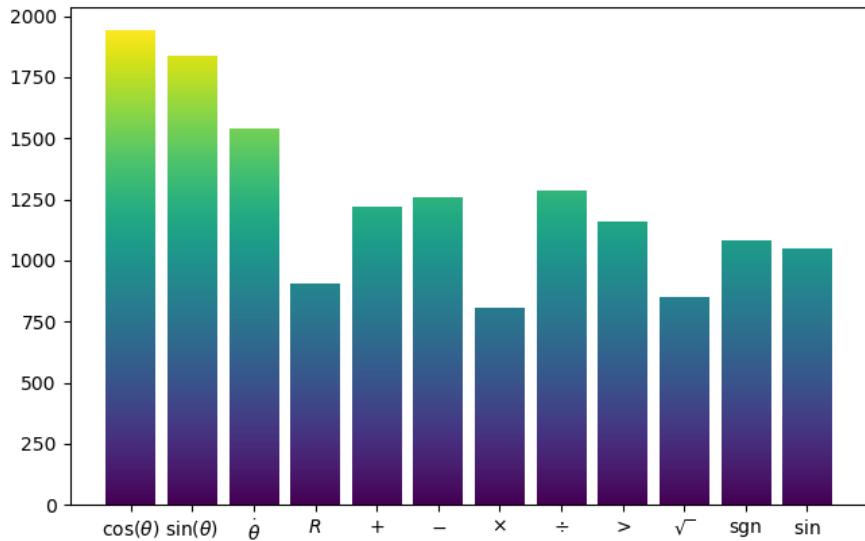


Figura 44: Histograma de operadores e variáveis terminais, na última geração.

O indivíduo de maior aptidão, na primeira execução, é mostrado na Figura 45.

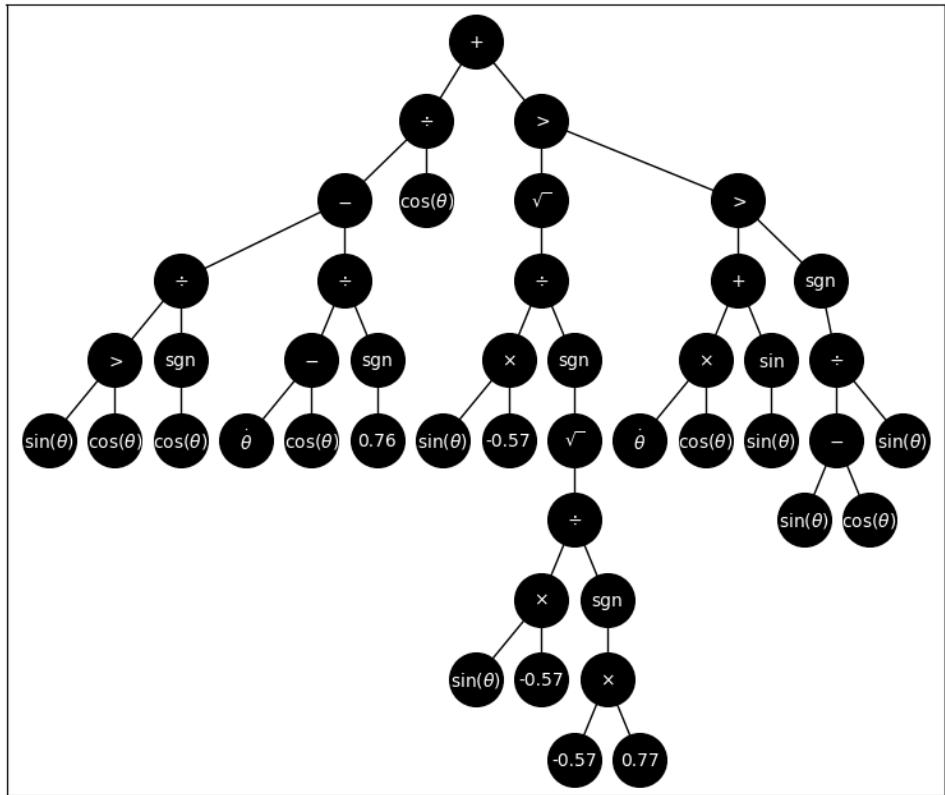


Figura 45: Primeiro integrante do hall da fama.

O segundo integrante do hall da fama, na primeira execução, é mostrado na Figura 46.

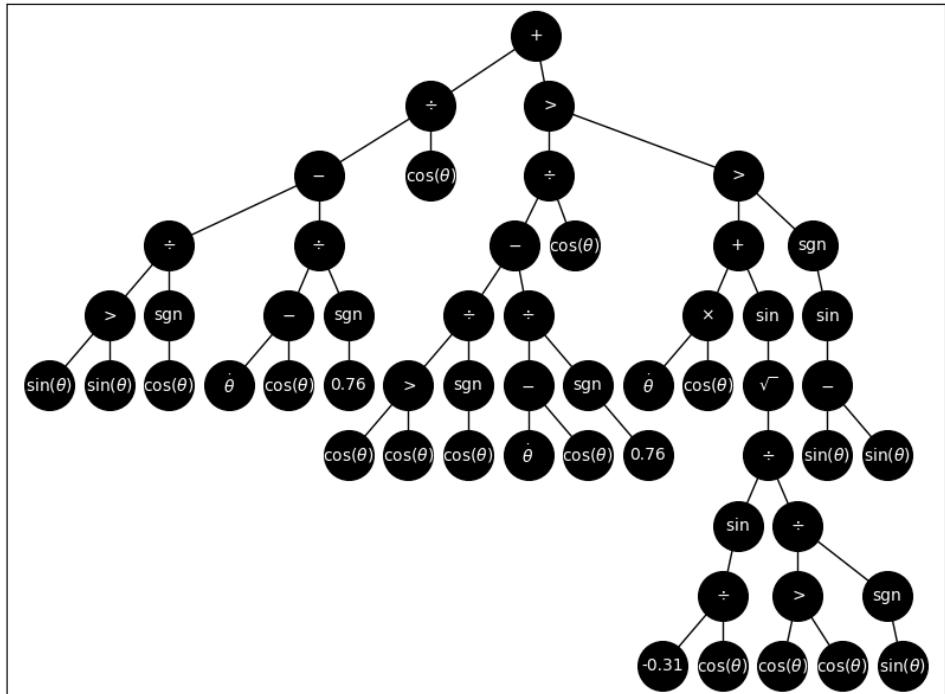


Figura 46: Segundo integrante do hall da fama.

A Figura 47 mostra as ações, o ângulo e velocidade angular do bastão, quando o sistema é submetido ao controle do indivíduo da Figura 45, em um único episódio.

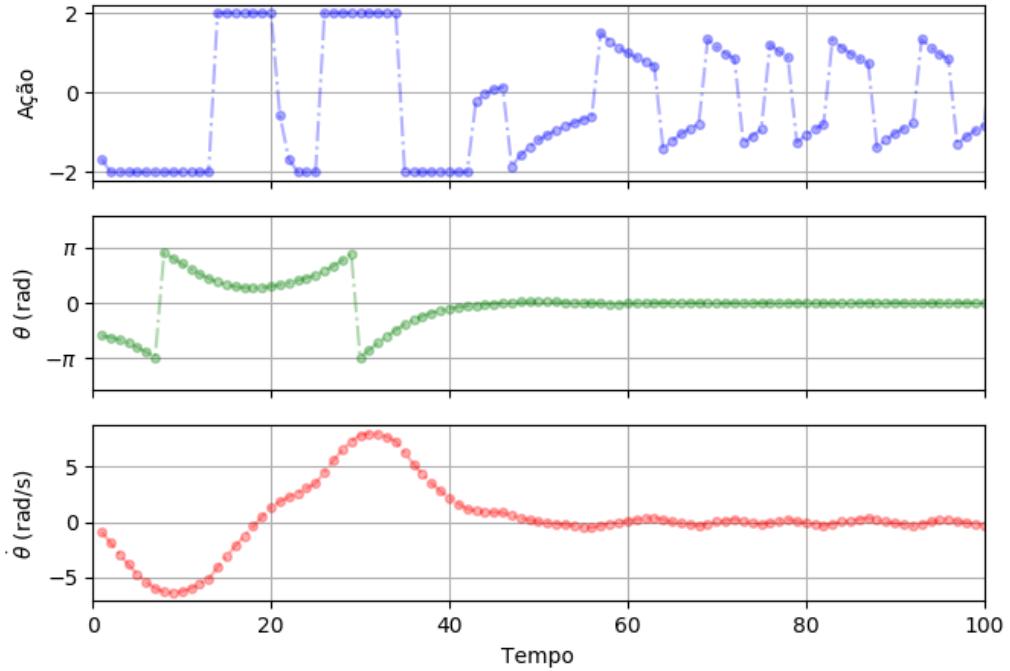


Figura 47: Controle exercido pelo indivíduo da Figura 45.

As transições bruscas no ângulo θ indicam a passagem da extremidade livre do pêndulo no ponto mais baixo, onde a função adquiriu o maior valor absoluto. Após alguns instantes, o ângulo do bastão oscila em torno do zero de forma estável.

Utilizando a mesma metodologia do Capítulo 4.1, é feita a avaliação em 100 episódios dos indivíduos do hall da fama. O maior valor de aptidão obtido é 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 é obtida através função de recompensa disponibilizada pelo ambiente de simulação. Com isso, é possível realizar uma comparação direta entre as duas abordagens.

A Figura 48 mostra o agente DDPG sendo treinado em 100000 passos de tempo.

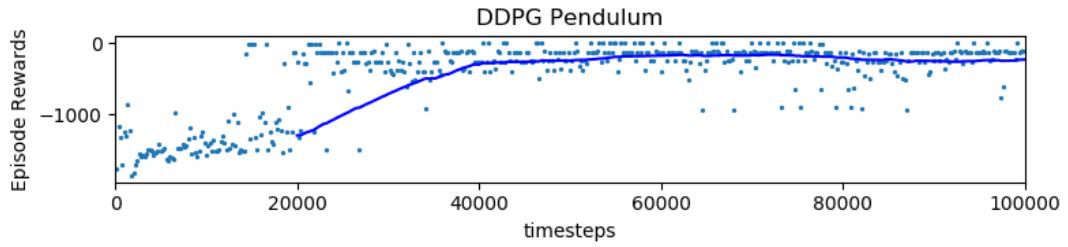


Figura 48: Evolução da recompensa acumulada para o agente DDPG no pêndulo swing-up.

A atuação do agente em um episódio pode ser vista na Figura 49.

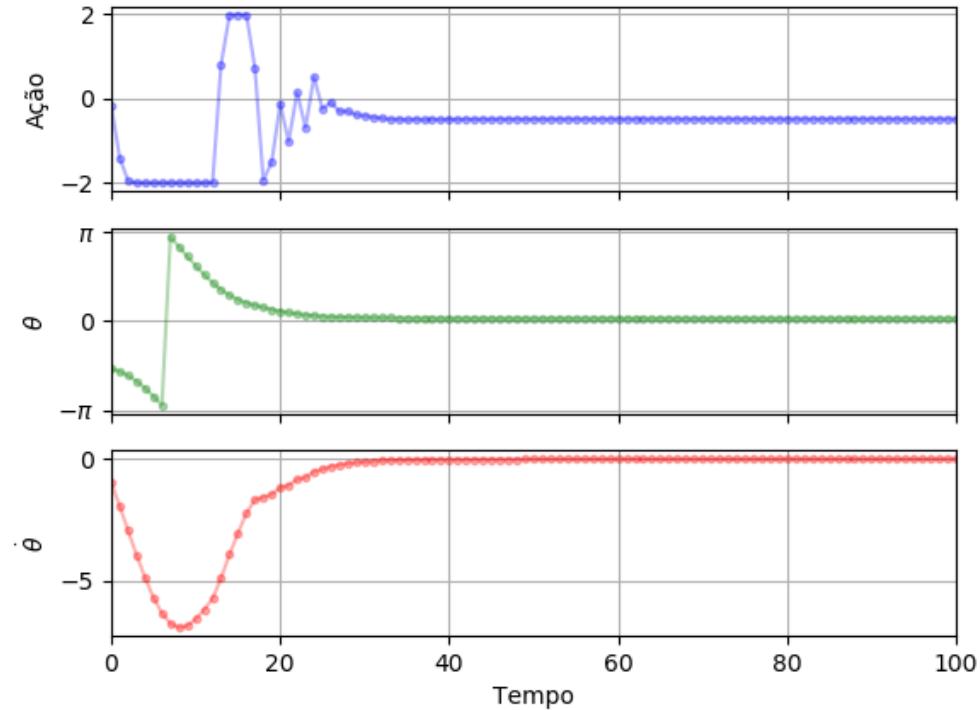


Figura 49: Dinâmica do pêndulo swing-up sob ação do agente DDPG.

Tabela 9: 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 9 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 veículo (que contém o ponto de fixação) sobre uma trilha. A diferença reside na existência de um outro bastão, fixado na extremidade antes livre, aumentando consideravelmente a complexidade do problema. As Figuras 50 e 51 mostram o sistema proposto.

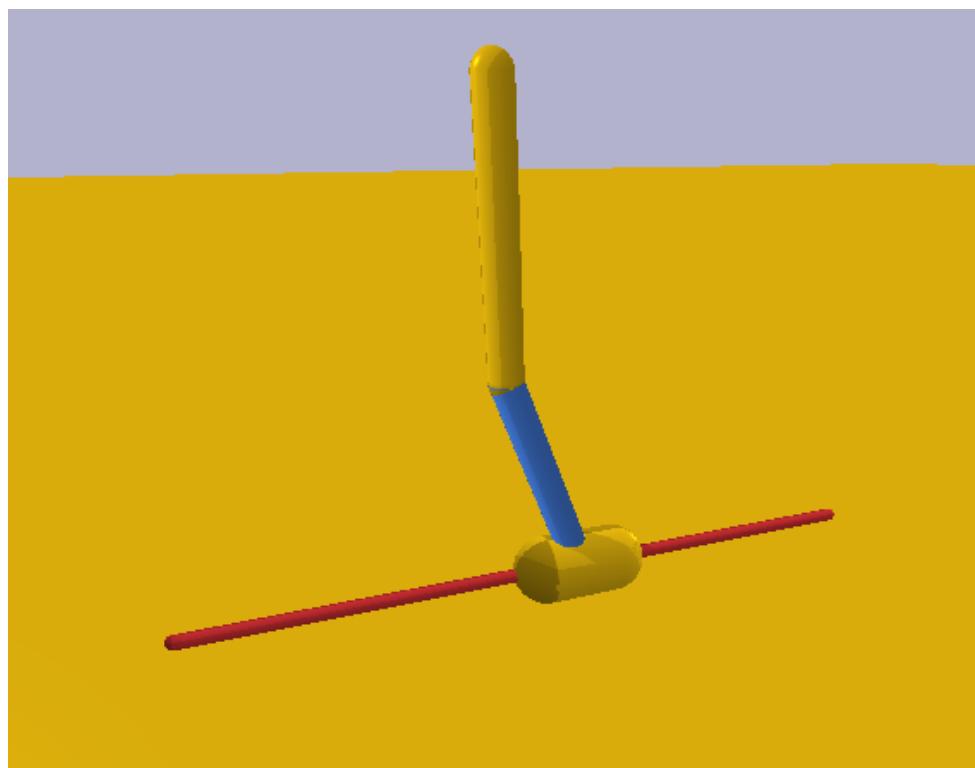


Figura 50: Renderização 3D do sistema pêndulo duplo invertido.

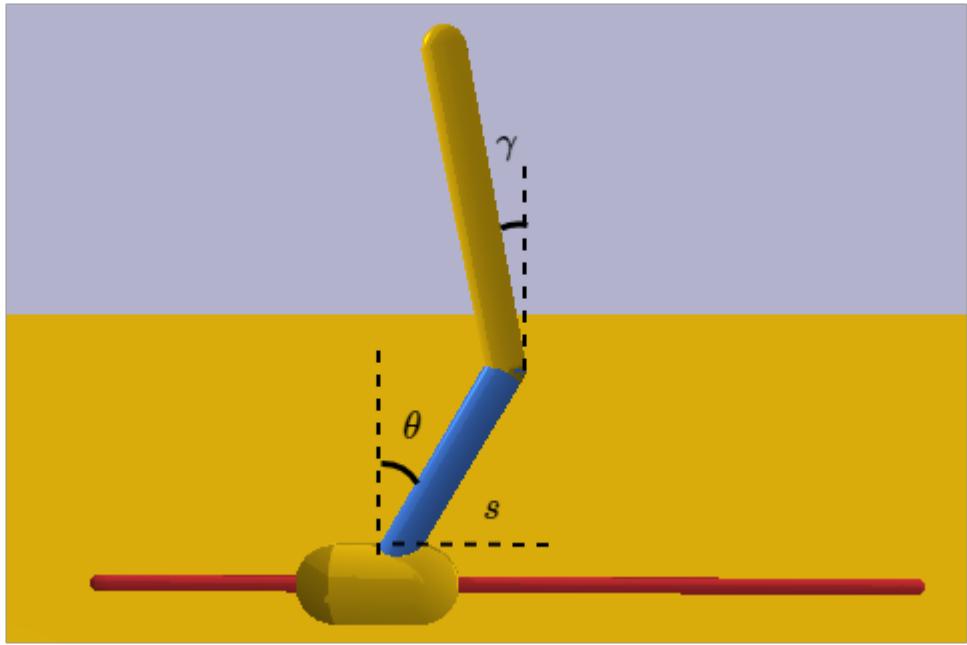


Figura 51: Ângulos θ , γ e a posição do veículo (s).

A observação do sistema é composta de 11 variáveis:

Tabela 10: Variáveis disponibilizadas como observação do 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 10 fazem parte do conjunto primitivo. Os outros parâmetros foram mantidos iguais ao do problema anterior (Tabela 8).

A função de recompensa do ambiente de simulação, a cada instante de tempo, pode ser vista na Equação 11.

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

$$d_p(t) = 0,01 [s(t)]^2 + [y(t) - 2]^2 \quad (11)$$

$$v_p(t) = 0,001 [\dot{\theta}(t)]^2 + 0,005 [\dot{y}(t)]^2$$

Na Equação 11, $d_p(t)$ 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 no 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 função de recompensa mostrada na Equação 11 será utilizada para o cálculo da aptidão de um indivíduo, utilizando a mesma formulação da Equação 10.

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

$$A_{tot}^{ep} = \sum_{t=0}^T A(t) = - \sum_{t=0}^T [10 - d_p(t) - v_p(t)] \quad T \leq 1000 \quad (12)$$

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

Como o espaço de ações é contínuo, foi utilizada a função clip, da Figura 39, como wrapper para o resultado produzido pelas árvores. Entretanto, os valores extremos serão -1 e 1 , e não -2 e 2 .

As Figuras 52 a 55 mostram os resultados obtidos, através da média de 10 execuções do algoritmo.

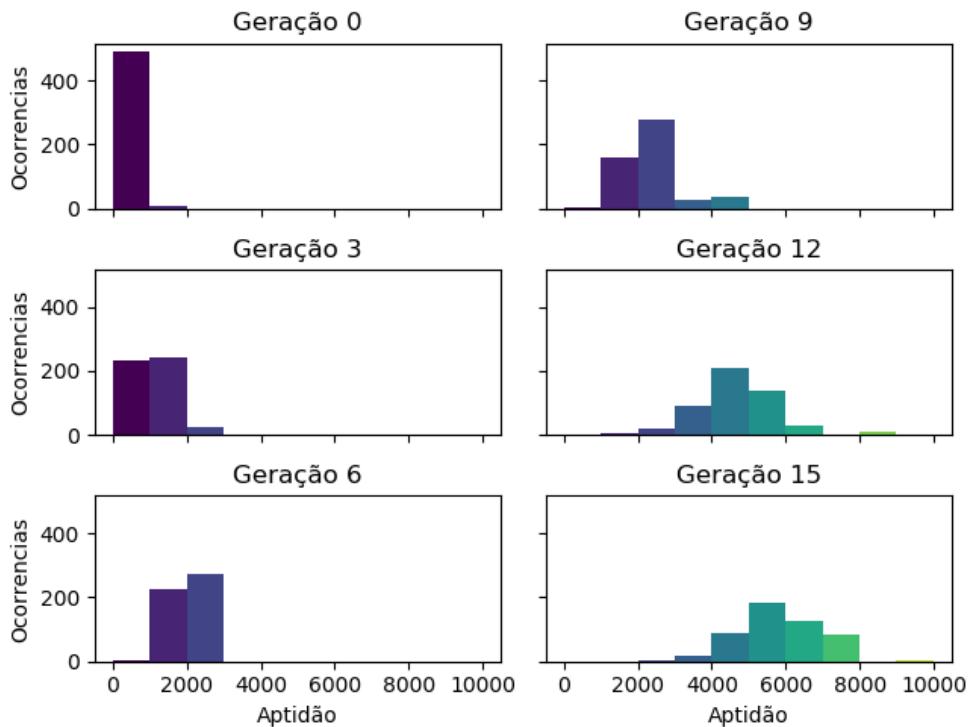


Figura 52: Histograma da aptidão dos indivíduos, em algumas gerações, para o pêndulo duplo invertido.

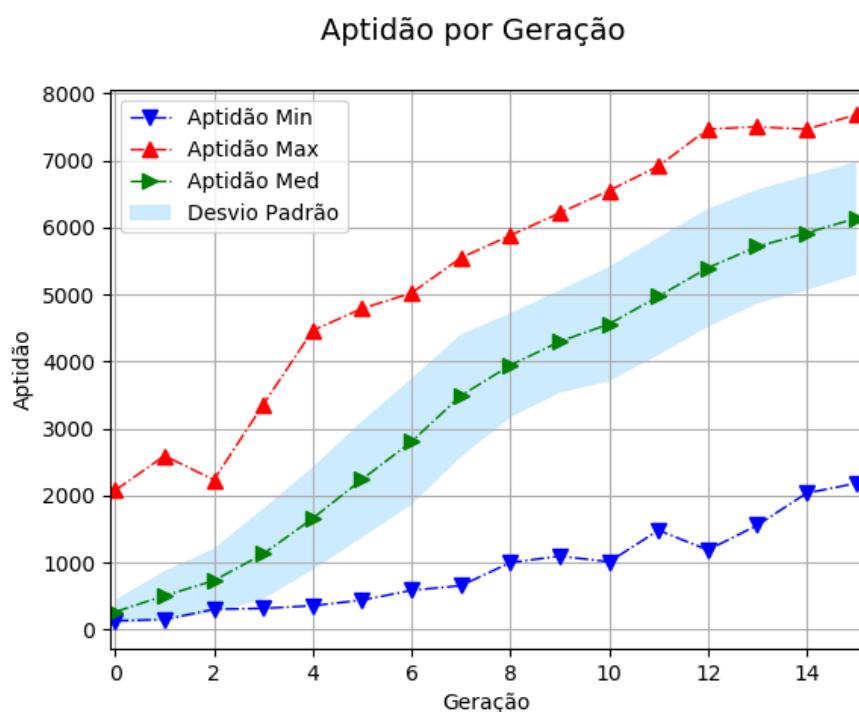


Figura 53: Medidas de aptidão da população, em cada geração.

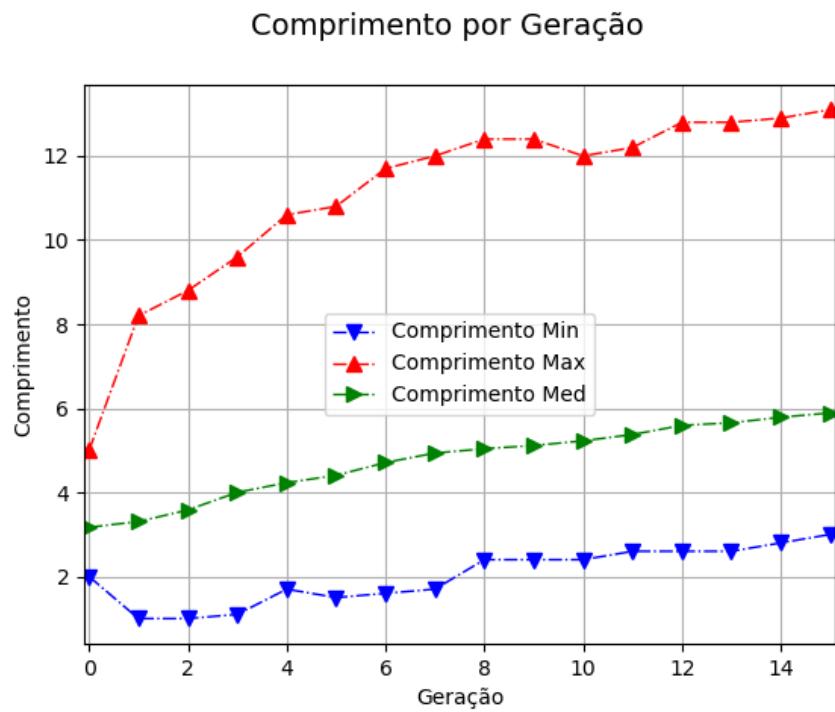


Figura 54: Medidas de comprimento da população, ao longo das gerações.

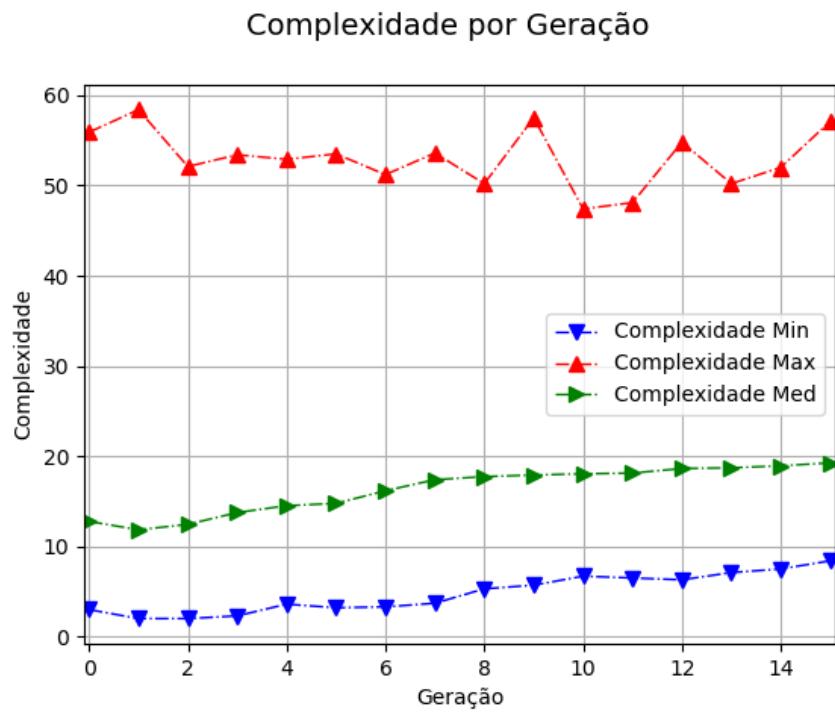


Figura 55: Medidas de complexidade dos indivíduos, em função das gerações.

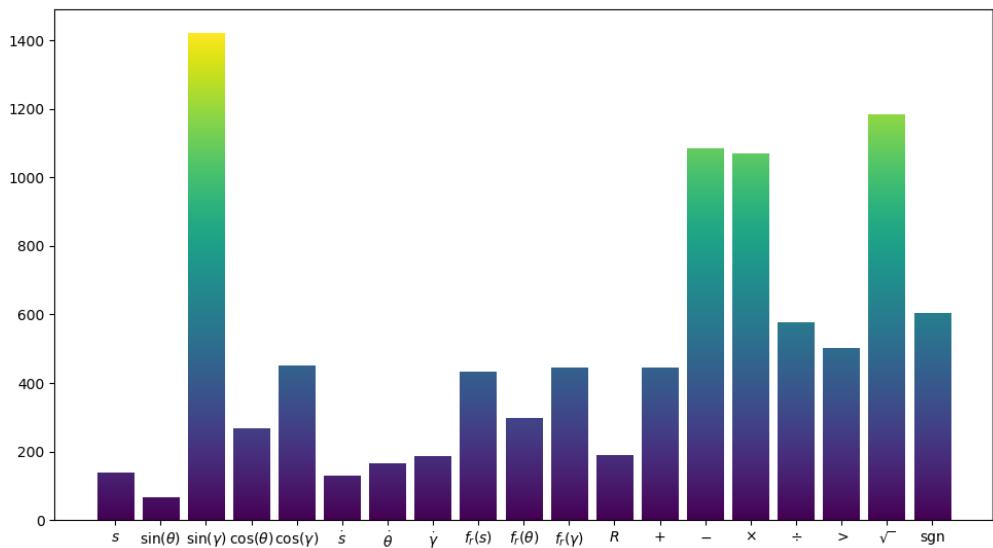


Figura 56: Número de ocorrências dos operadores e variáveis terminais, na última geração.

Para cada indivíduo do hall da fama foram realizadas 100 simulações, com o intuito de verificar a solução mais apta e generalista. O indivíduo da Figura 57 obteve uma aptidão média de 7794.

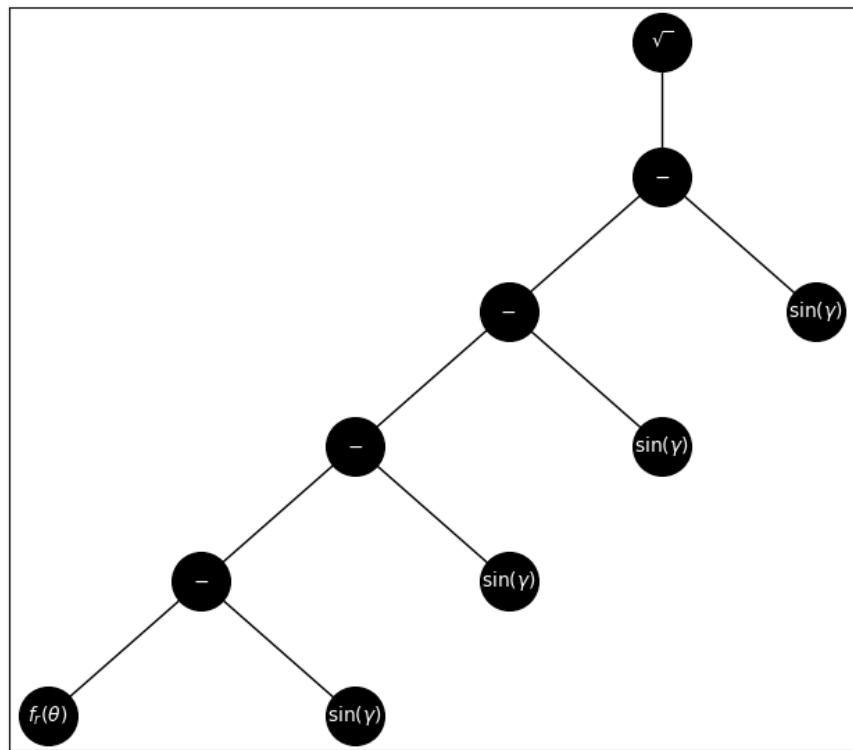


Figura 57: Indivíduo mais apto da primeira execução do algoritmo.

Os gráficos da Figura 58 mostram os ângulos e a posição do carrinho, quando a solução da Figura 57 atua em um episódio.

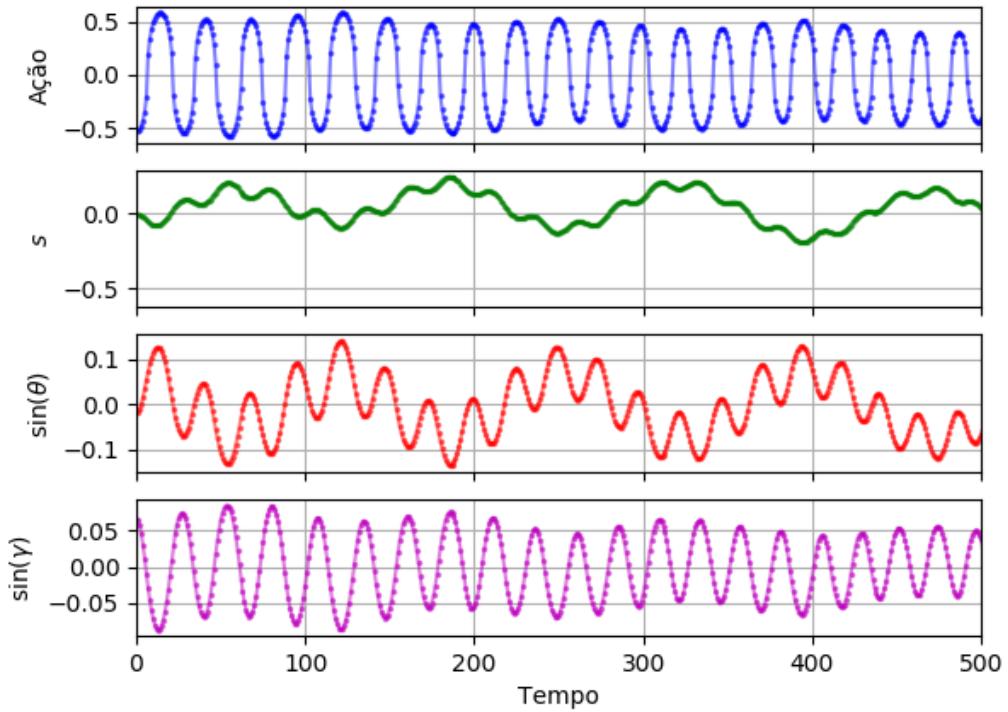


Figura 58: Avaliação do melhor indivíduo observado na primeira execução.

O gráfico da Figura 59 mostra um agente sendo treinado com o algoritmo DDPG, por 400000 passos de simulação. A recompensa média acumulada pelo agente, em 100 episódios, pode ser vista na Tabela 11, junto à outras estatísticas relacionadas ao custo computacional.

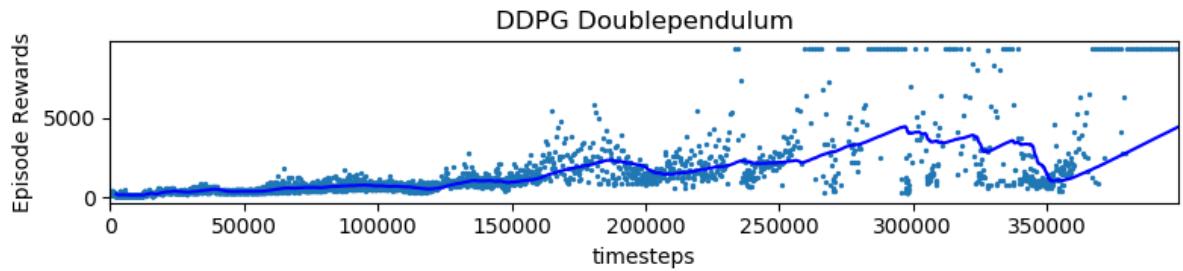


Figura 59: Evolução da recompensa acumulada do agente DDPG. A linha azul representa a média móvel.

Tabela 11: 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 ao se locomover na direção contrária. A Figura 60 ilustra o problema.

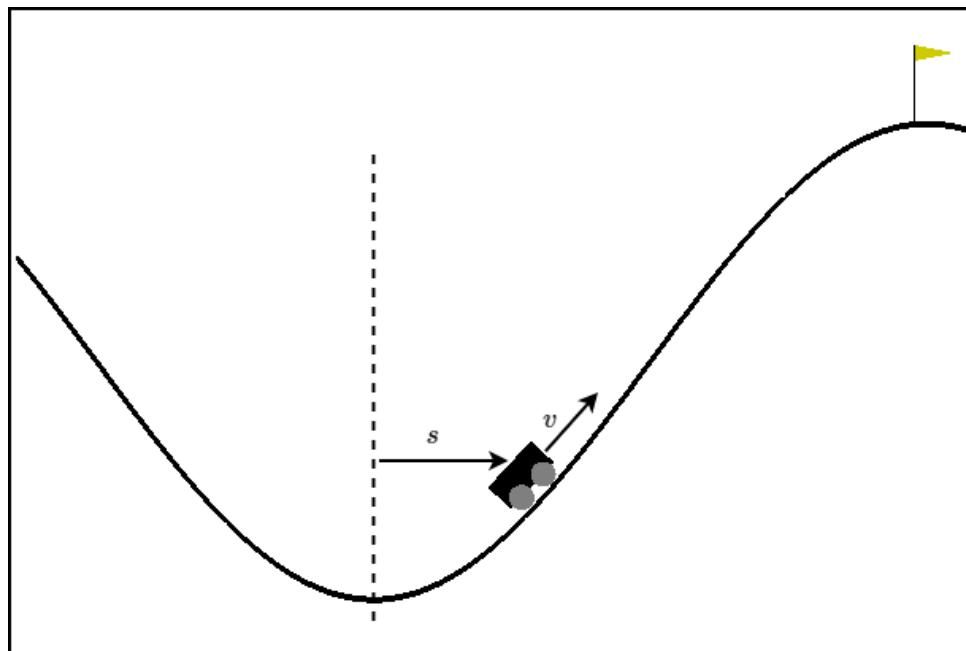


Figura 60: Problema do carro na ladeira.

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 60.

A observação do ambiente é simples, e consiste apenas na posição e velocidade do carro, conforme a Tabela 12 indica. O agente recebe uma penalização unitária, conforme indica a Equação 13, a cada instante de tempo. Desta forma, a busca é por indivíduos que cheguem ao objetivo no menor tempo possível.

Tabela 12: Variáveis de estado para o problema do carro na ladeira.

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

$$r(t) = -1 \quad (13)$$

O espaço de ações é discreto e indica uma força que atua no veículo, de acordo com a Equação 14.

$$a(t) = \begin{cases} 0 & \rightarrow \text{Força aplicada à esquerda} \\ 1 & \rightarrow \text{Não aplica força} \\ 2 & \rightarrow \text{Força aplicada à direita} \end{cases} \quad (14)$$

No capítulo 4.1 foi visto como transformar um número no espaço contínuo, retornado pela função de controle de um indivíduo, em uma ação mapeada em números inteiros. Com um conjunto de duas ações possíveis, bastava mapear números positivos à uma ação e negativos à outra. No problema atual, existem três ações possíveis, portanto, é definida uma faixa de valores para a situação em que não se aplica força , isto é, quando $a(t)$ vale 1. Este artifício pode ser conferido na Equação 15, onde $num(t)$ representa a saída da função de controle do indivíduo, no instante t .

$$a(t) = \begin{cases} 0, & \text{se } num(t) \leq -0,5 \\ 1, & \text{se } -0,5 < num(t) \leq 0,5 \\ 2, & \text{se } 0,5 < num(t) \end{cases} \quad (15)$$

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

$$\bar{A} = \frac{1}{nep} \sum_{ep=1}^{nep} \sum_{t=0}^T r(t) \quad (16)$$

A Tabela 13 mostra os parâmetros utilizados na execução da PG.

Tabela 13: 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, 1)
Número de Simulações	10
Tamanho do Campeonato de Aptidão	6
Tamanho do Campeonato de Comprimento	1,2
Operações	+, -, ×, ÷, √, sin, >, 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 61 a 65.

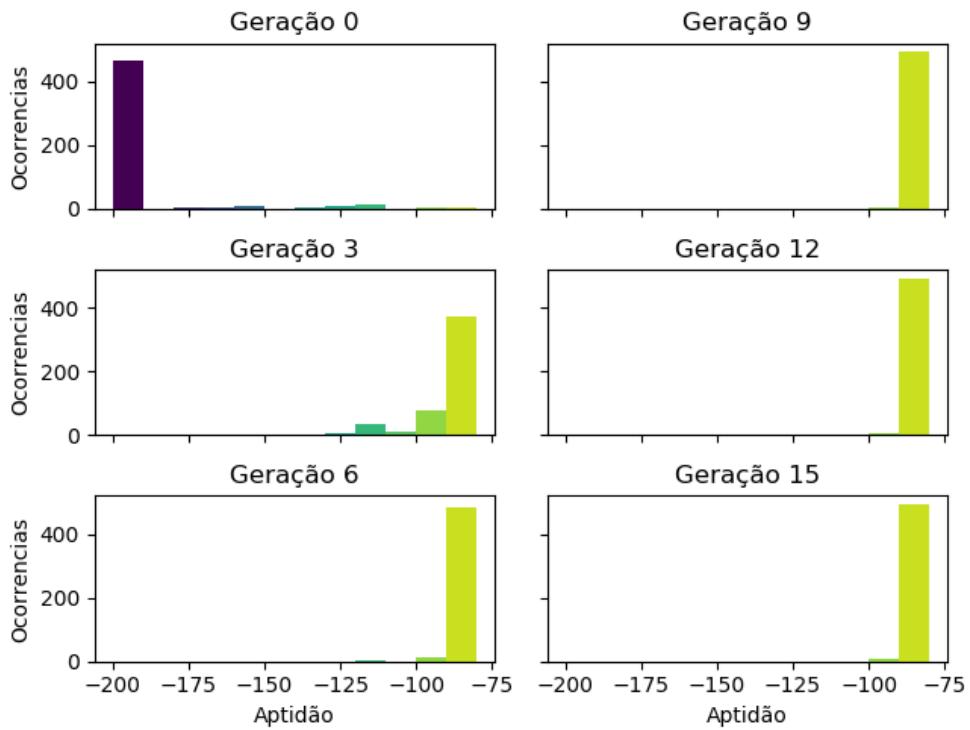


Figura 61: Histograma da aptidão dos indivíduos, em algumas gerações, para o pêndulo duplo invertido.

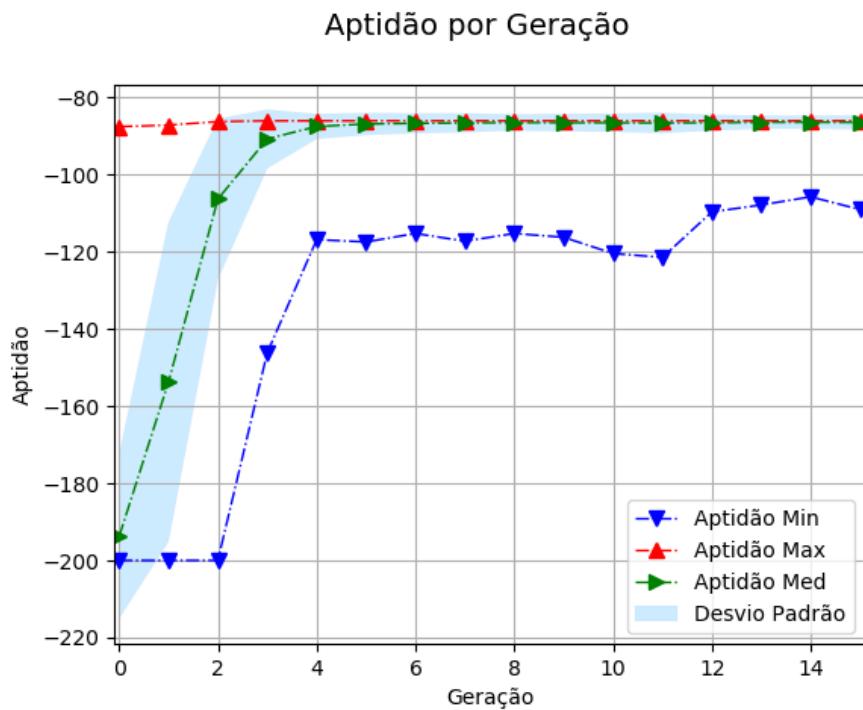


Figura 62: Aptidão da população.

Comprimento por Geração

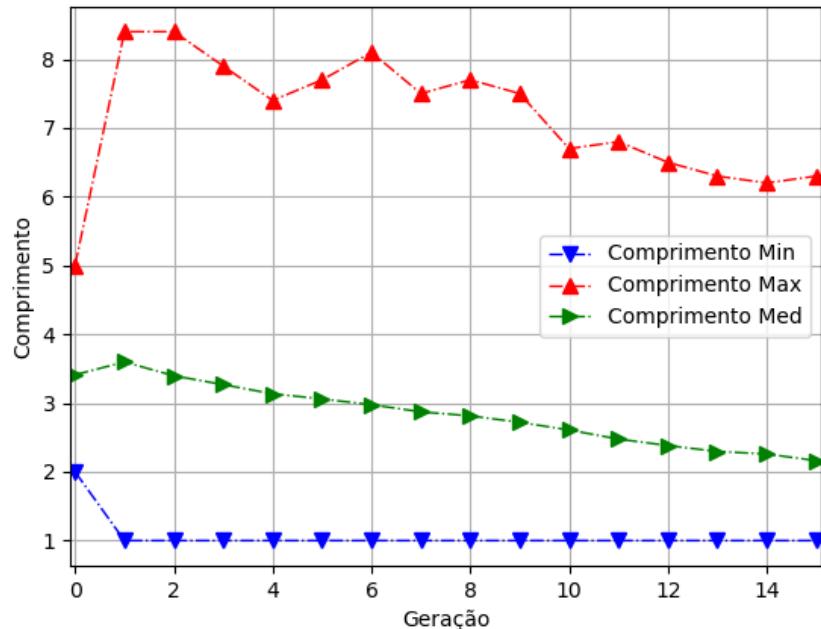


Figura 63: Comprimento dos indivíduos.

Complexidade por Geração

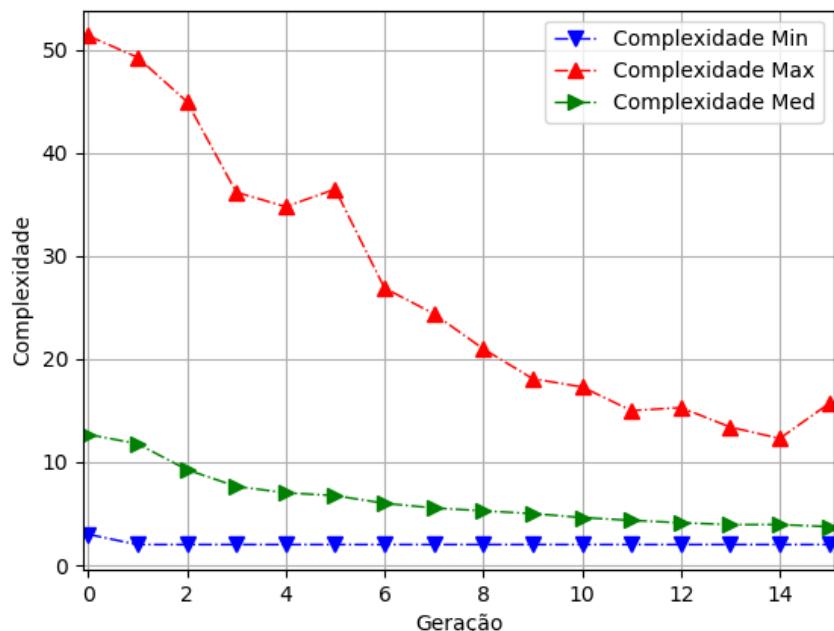


Figura 64: Complexidade da população.

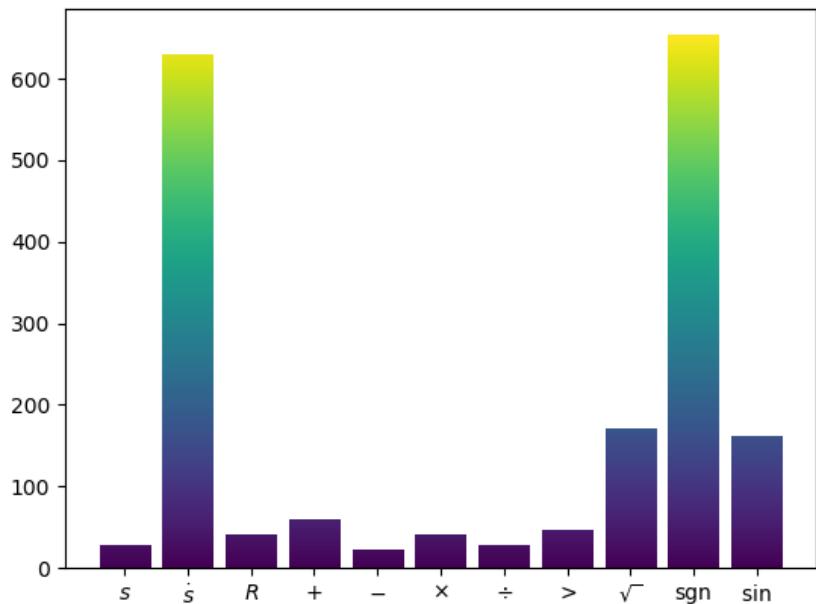


Figura 65: Ocorrências dos operadores e variáveis terminais, na última geração.

O melhor indivíduo foi obtido ao simular a atuação de todas as soluções do hall da fama, em 100 episódios. A Figura 66 mostra a composição do indivíduo.

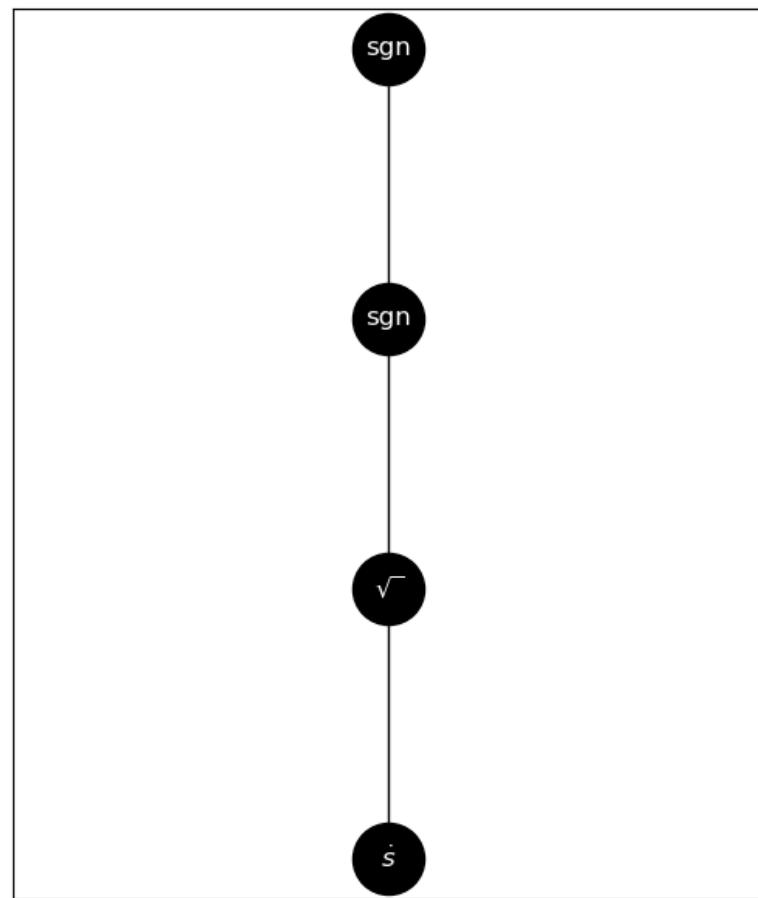


Figura 66: Melhor indivíduo da primeira execução.

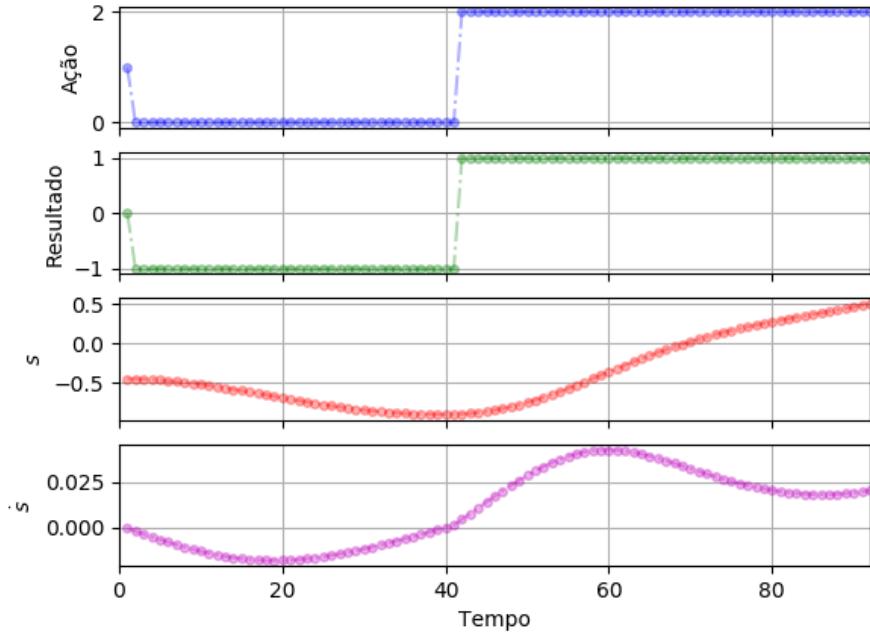


Figura 67: Atuação do indivíduo da Figura 66 em um episódio aleatório.

Como o espaço de ações é discreto, um agente foi treinado utilizando o algoritmo DQN. A Figura 68 mostra a recompensa acumulada pelo agente DQN ao longo do tempo de execução. A Tabela 14 sumariza os resultados encontrados com a PG, comparado à outra abordagem.

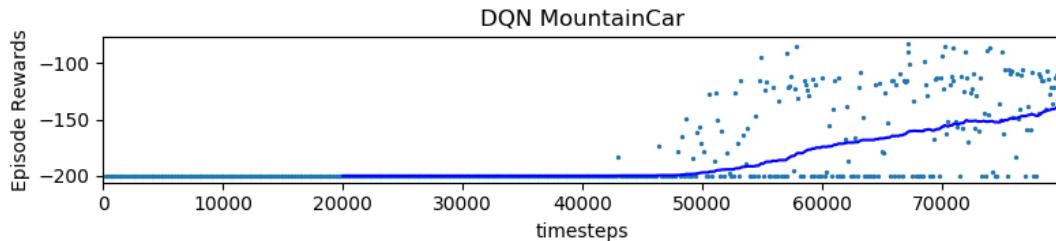


Figura 68: Agente DQN e a recompensa acumulada ao longo dos passos de tempo.

Tabela 14: 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.

4.5 Veículo Terrestre Autônomo

O último problema abordado trata do planejamento de trajetória para um veículo autônomo semelhante a um automóvel de passeio. O veículo é propelido através do torque aplicado às rodas dianteiras, as quais também dão direção ao movimento permitindo a mudança de orientação do veículo em trajetórias curvas.

A cinemática do veículo é dada pela Equação 17, nas quais a velocidade v e a orientação ϕ das rodas dianteiras são os sinais de controle do movimento, enquanto x , y e θ representam as coordenadas do centro de gravidade do veículo (origem do sistema de coordenadas fixo ao veículo) com relação a um sistema de coordenadas fixo de referência e a orientação entre ambos, respectivamente. L é a distância entre os eixos traseiro e dianteiro.

$$\begin{aligned}\dot{x}_c &= v \cos(\theta + \beta) \\ \dot{y}_c &= v \sin(\theta + \beta) \\ \dot{\theta} &= \frac{v \cos \beta \tan \phi}{L} \\ \beta &= \arctan\left(\frac{\tan \phi}{2}\right)\end{aligned}\tag{17}$$

A Figura 69 apresenta o modelo cinemático de “bicicleta” em que se baseia a Equação 17. Neste modelo, é realizada uma simplificação da cinemática de veículos de passeio, ao considerar que as rodas dos eixos frontais e traseiros se unem no centro geométrico de cada eixo. Dessa forma, considera-se a existência de apenas duas rodas no veículo, equiparando-se a uma bicicleta, conforme indica a Figura 69. Apesar de sua simplicidade, o modelo se mostra eficiente para veículos com baixa aceleração [24]. Supõe-se, na Equação 17, que o centro de gravidade do veículo coincide com o centro geométrico do mesmo.

²Valor aproximado.

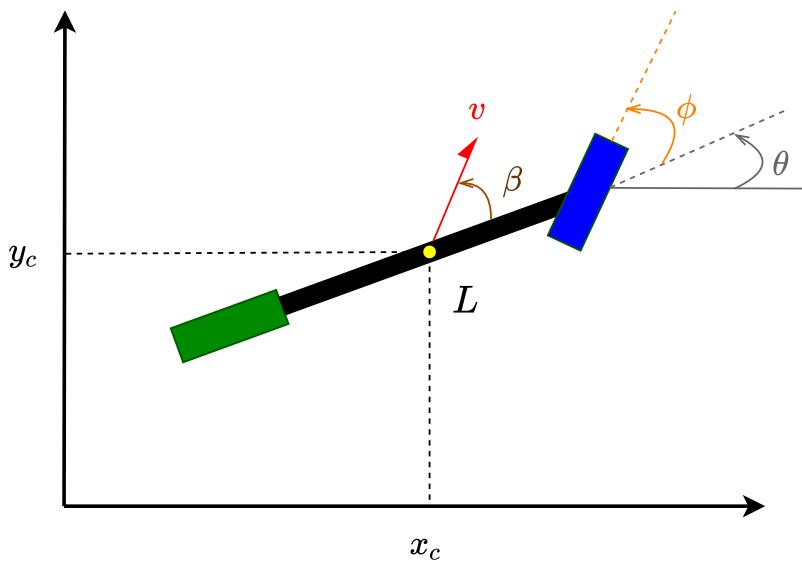


Figura 69: Modelo de bicicleta com o ponto de referência no centro de gravidade.

O objetivo do experimento é verificar a capacidade de gerar funções temporais para os sinais de controle que levem o veículo de uma pose inicial para uma pose final, através da programação genética.

O problema foi implementado a partir da biblioteca Gym. Portanto, foi necessário definir:

- a) As variáveis que são fornecidas como observação do sistema:

A Tabela 15 apresenta as variáveis que compõe a observação do sistema e que são utilizadas como variáveis terminais.

Tabela 15: Variáveis que fornecem informações sobre o sistema para o agente.

Variável	Significado
x	Projeção horizontal da distância entre o veículo e o alvo
y	Projeção vertical da distância entre o veículo e o alvo
θ	Orientação do robô
\dot{x}	Componente de velocidade linear em x
\dot{y}	Componente de velocidade linear em y
$\dot{\theta}$	Taxa de variação de orientação do veículo
v	Velocidade do veículo
ϕ	Orientação do eixo frontal

Nota-se na Tabela 15 que, diferentemente da Equação 17, as coordenadas x e y representam distâncias em relação ao centro geométrico do veículo. Isto é, o agente recebe uma informação, a cada passo de simulação, que não inclui precisamente as variáveis de estado que implementam a simulação cinemática do veículo. As variáveis da Tabela 15 podem ser visualizadas na Figura 70.

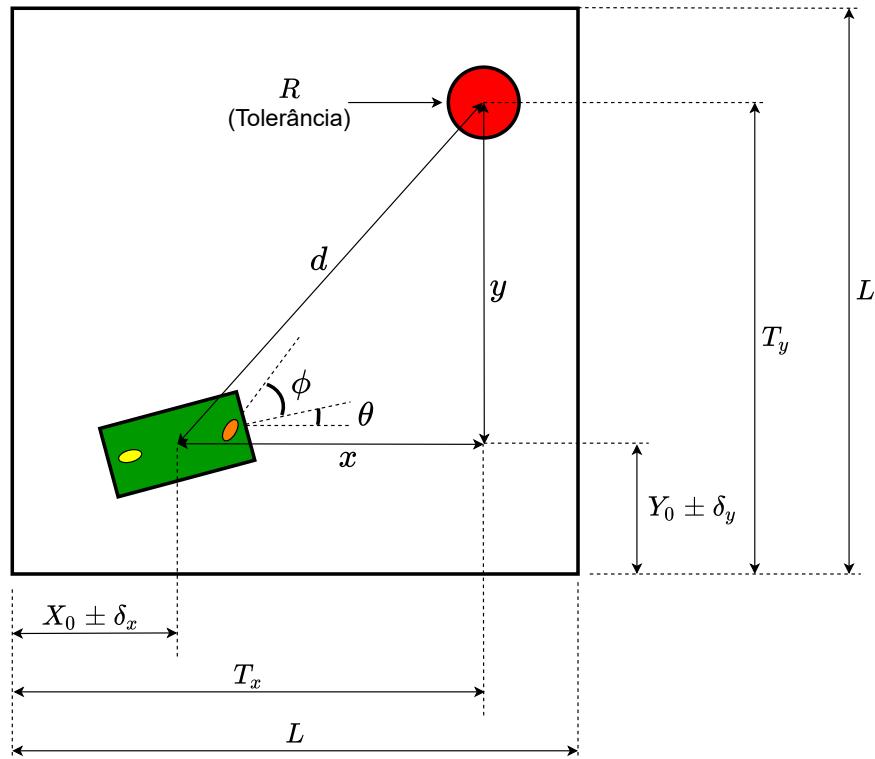


Figura 70: Visão geral das variáveis fornecidas ao agente e outras informações sobre a implementação da simulação do veículo autônomo.

b) Os limites de cada variável:

Ao definir os limites de cada variável terminal da Tabela 15, foi possível implementar critérios de término para o episódio, simplificando o custo computacional associado à avaliação dos indivíduos.

Na Tabela 16, L é um limite espacial, definido para cada caso particular. Isto é, a criação do ambiente permite impor uma distância máxima em relação ao alvo, impedindo a continuação da simulação para os indivíduos que se distanciem muito do objetivo. Não se mostrou necessário impôr limites para as outras variáveis que compõe a observação.

Tabela 16: Limites definidos para algumas variáveis da Tabela 15.

Variável	Mínimo	Máximo
x	-L m	L m
y	-L m	L m

c) O limite de cada variável de controle:

Os limites estabelecidos para as variáveis de controle são aproximações do que foi observado em um protótipo do veículo terrestre autônomo, apresentado na Figura 71, construído por estudantes do curso de Engenharia Elétrica na Universidade do Estado do Rio de Janeiro. A aceleração observada é condizente com a suposição da Equação 17. Além de permitir uma estimativa das restrições de um veículo autônomo de pequeno porte, o protótipo pode servir futuramente como um veículo de testes para algoritmos de planejamento de trajetória.

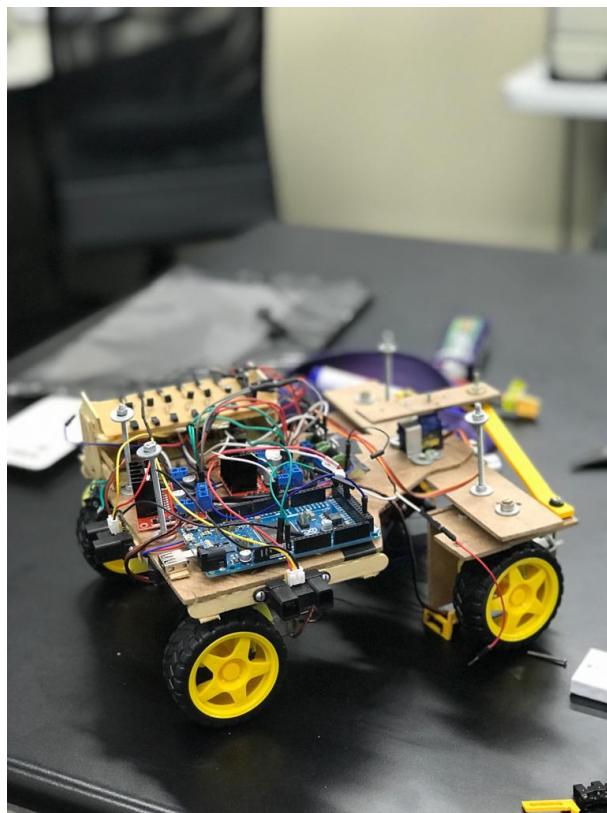


Figura 71: Protótipo do veículo autônomo.

As Tabelas 17 e 18 apresentam as restrições definidas para a simulação, quando as variáveis de controle são v e ϕ ou \dot{v} e $\dot{\phi}$, respectivamente.

Tabela 17: Limites para as variáveis de controle v e ϕ .

Variável	Mínimo	Máximo
v	-0,1 m/s	0,1 m/s
ϕ	-0,5 rad/s	0,5 rad/s

Tabela 18: Restrições ao controle exercido por meio das variáveis \dot{v} e $\dot{\phi}$.

Variável	Mínimo	Máximo
\dot{v}	-0,1 m/s	0,1 m/s
$\dot{\phi}$	-0,5 rad/s	0,5 rad/s

d) A função de recompensa:

Para os casos em que se deseja apenas chegar ao alvo, independentemente da orientação do robô no local, a função de recompensa é:

$$r(t) = \begin{cases} -1000, & O(t) \notin \mathcal{S}_O \\ 2000, & d \leq R \\ -d \end{cases} \quad (18)$$

Quando deseja-se considerar a orientação do robô na posição do alvo, a função de recompensa será:

$$r(t) = \begin{cases} -1000, & O(t) \notin \mathcal{S}_O \\ 2000 \cdot |\theta_{ref} - \theta|, & d \leq R \\ -d \end{cases} \quad (19)$$

Nota-se que \mathcal{S}_O representa o espaço de observação, isto é, o conjunto de valores das variáveis de estado que não causam o término do episódio, conforme estabelecido na Tabela 16.

e) Os critérios de término do episódio:

Naturalmente, deve haver um tempo limite para cada simulação. Além disso, caso o veículo não obedeça os limites estabelecidos na Tabela 16 ou atinja o alvo, dentro da tolerância estabelecida, o episódio de simulação é terminado automaticamente.

A partir dessas características, os métodos da biblioteca Gym (Capítulo 2.2) foram implementados, incluindo a função *render*, que permite visualizar a simulação em tempo real, através de um vídeo.

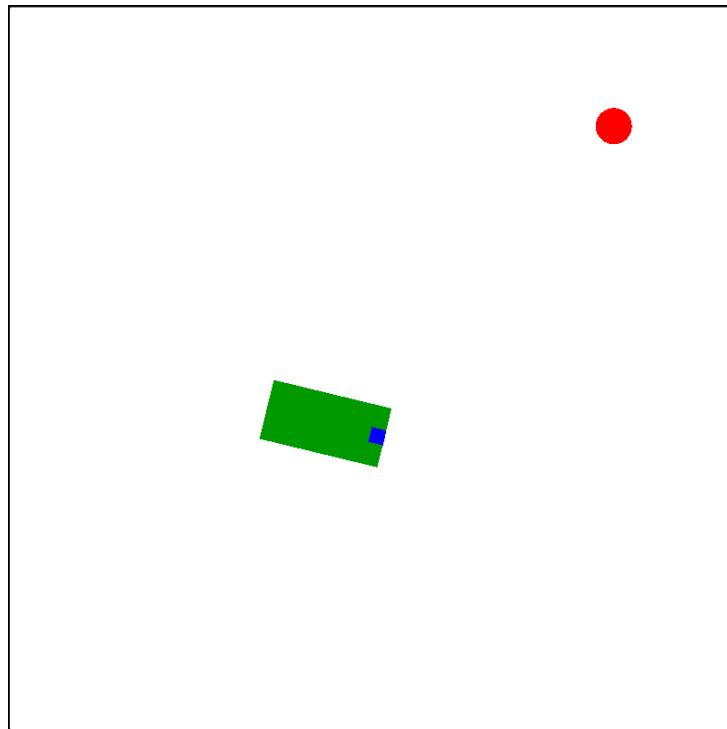


Figura 72: *Frame* do vídeo em um episódio de simulação.

Existe uma particularidade na aplicação da PG neste problema: cada indivíduo deve produzir dois valores de controle. Nos casos anteriores abordados, havia uma única ação, o que tornou a aplicação da PG imediata, já que cada árvore produz um único resultado numérico.

Para contornar este problema, é utilizada a programação genética *multigênica*, onde cada indivíduo é composto de duas árvores: uma responsável pelo controle de v (ou \dot{v}) e outra para a variável ϕ (ou $\dot{\phi}$). As operações genéticas são aplicadas em cada árvore do indivíduo. Especificamente, para a operação de cruzamento, a recombinação ocorre entre árvores que controlam a mesma variável.

Para que os valores de controle produzidos pelos indivíduos respeitem os limites estabelecidos nas Tabelas 17 e 18, o valor resultante da avaliação da árvore é truncado com a função *clip* (Figura 39).

Algumas situações específicas relacionadas a este problema foram abordadas, em ordem de complexidade.

Problema 1) A partir de uma posição inicial **fixa**, chegar ao alvo com **qualquer** orientação ao controlar as entradas v e ϕ . Admite-se a possibilidade de variação instantânea dessas variáveis.

Problema 2) A partir de uma posição inicial **fixa**, chegar ao alvo com uma orientação **definida** ao controlar as entradas v e ϕ . Admite-se a possibilidade de variação instantânea dessas variáveis.

Problema 3) A partir de uma posição inicial **aleatória**, chegar ao alvo com **qualquer** orientação ao controlar as entradas \dot{v} e $\dot{\phi}$.

Problema 4) A partir de uma posição inicial **aleatória**, chegar ao alvo com uma orientação **definida** ao controlar as entradas \dot{v} e $\dot{\phi}$.

Para os problemas 3 e 4, a aleatoriedade da posição inicial é estabelecida a partir dos parâmetros δ_x , δ_y e δ_θ , apresentados na Figura 70.

4.5.1 Problema 1

Neste caso, não há um objetivo associado à orientação do veículo e a pose inicial é fixa. Portanto, não é necessário que a avaliação de um indivíduo ocorra em vários episódios. A visão geral do problema pode ser vista na Figura 73.

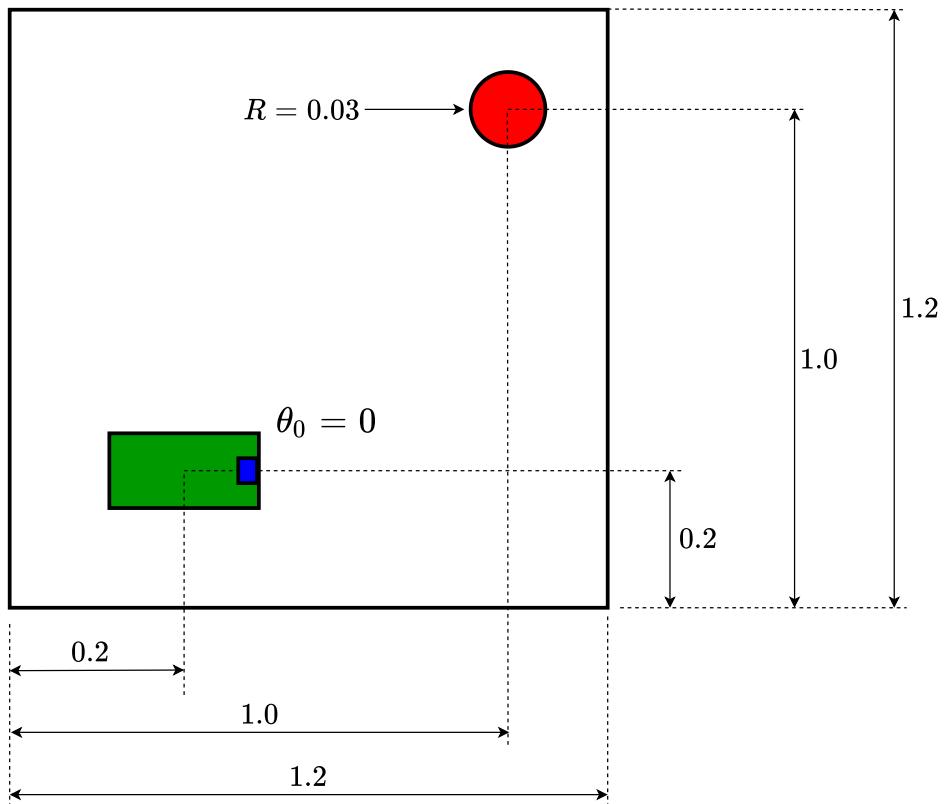


Figura 73: Primeira situação.

Os parâmetros utilizados para a programação genética são apresentados na Tabela 19.

Tabela 19: Parâmetros utilizados para o problema 1 do carro robô.

Parâmetro	Valor
Tamanho da População	500
Probabilidade de Cruzamento	0,75
Probabilidade de Mutação	0,05
Número de Gerações	15
Número de Entradas	6
Faixa para Constante Efêmera	(-1, 1)
Número de Simulações	1
Tamanho do Campeonato de Aptidão	6
Operações	+, -, ×, ÷, √, sin, >, sgn
Comprimento Mínimo e Máximo de Inicialização	(2, 5)
Comprimento Máximo de Mutação	7
Limite de Comprimento dos Indivíduos	17

Em conformidade com a abordagem utilizada até o momento, a aptidão é calculada a partir de uma função de recompensa (Equação 18).

$$A(t) = r(t), \forall t < T$$

$$A_{tot} = \sum_{t=0}^T r(t), \quad r(t) = \begin{cases} -1000, & O(t) \notin \mathcal{S}_O \\ 2000, & d \leq R \\ -d & \end{cases} \quad (20)$$

$$\bar{A} = A_{tot}$$

Os gráficos relacionados à aptidão dos indivíduos, ao longo das gerações, podem ser visto nas Figuras 74 e 75. As aptidões positivas indicam, necessariamente, que o indivíduo chegou ao local alvo. Dentre esses, os que alcançam o objetivo no menor tempo devem apresentar um desempenho melhor.

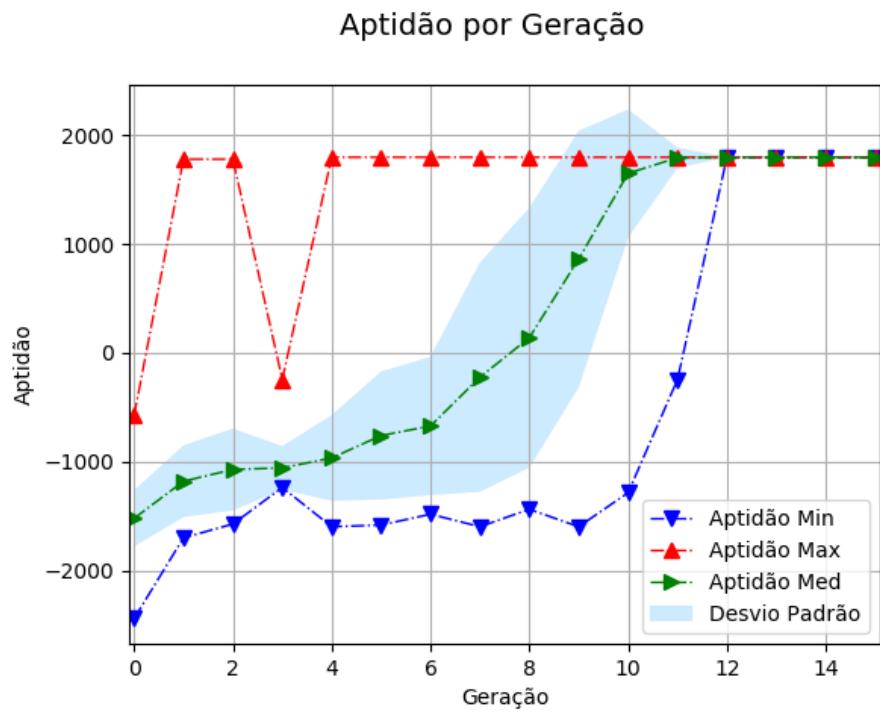


Figura 74: Aptidão dos indivíduos.

É possível perceber, na Figura 74, que na terceira geração alguns indivíduos já são capazes de alcançar o local alvo.

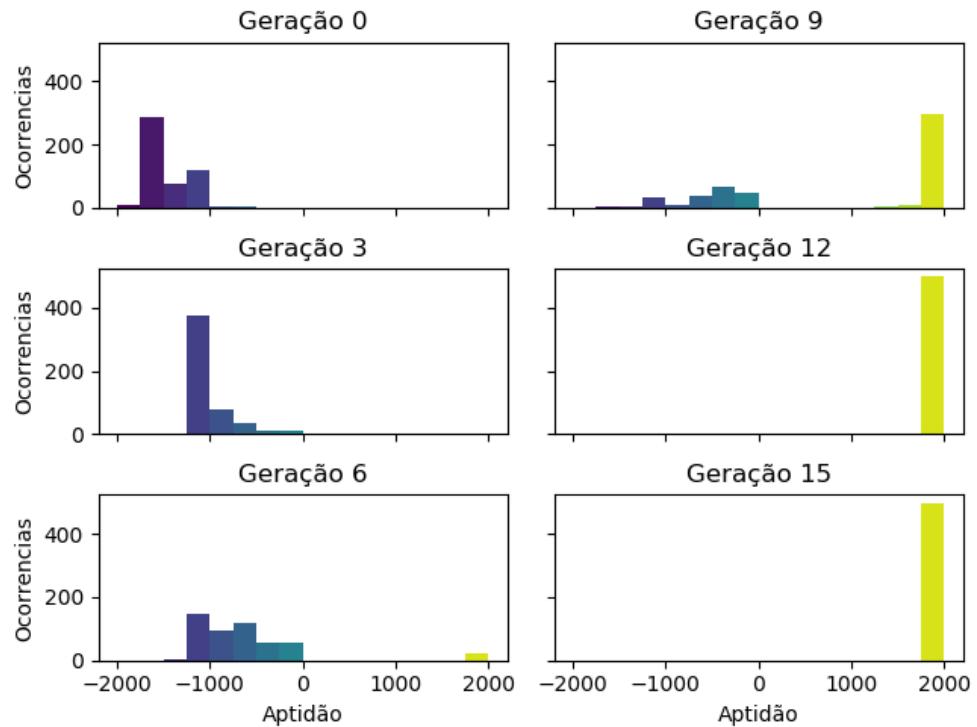


Figura 75: Histograma das aptidões.

O indivíduo que obteve melhor desempenho pode ser visto na Figura 76. A tra-

jetória deste indivíduo é apresentada na Figuras 77.

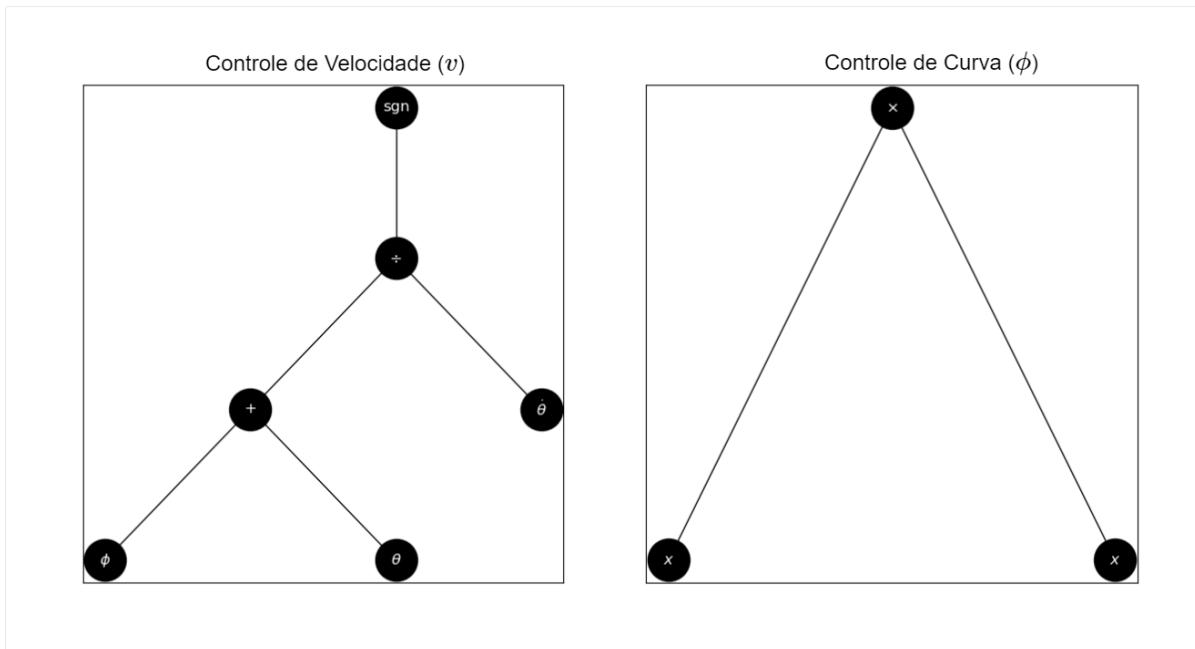


Figura 76: Árvores de controle do melhor indivíduo encontrado.

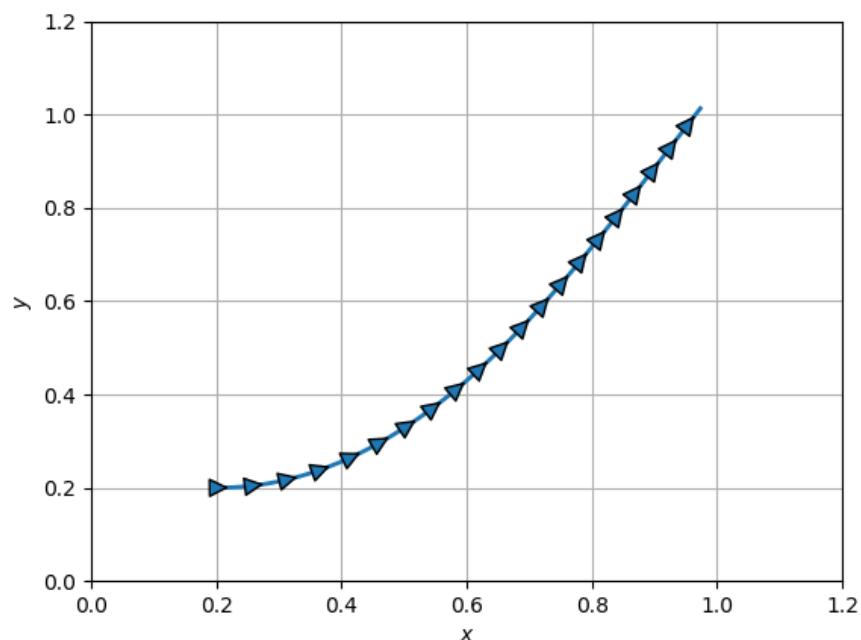


Figura 77: Trajetória do indivíduo de melhor desempenho. As setas indicam a orientação do veículo ao longo da trajetória.

A Figura 78 ilustra o controle exercido pelo melhor indivíduo durante a simulação.

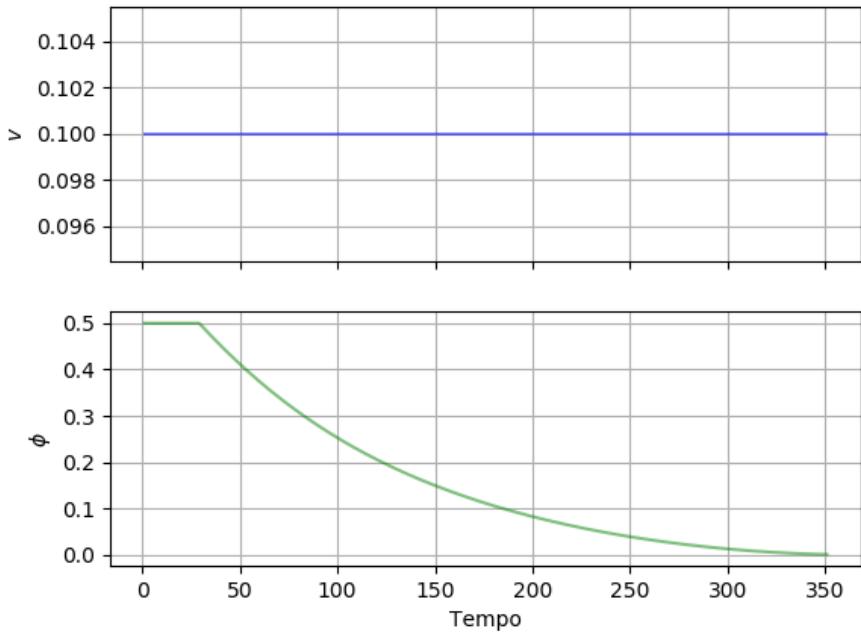


Figura 78: Saída das funções de controle ao longo da simulação.

Os dados relativos à execução do algoritmo podem ser vistos na Tabela 20.

Tabela 20: Problema 1 - custo computacional.

Tempo de execução (h:m:s)	00:34:26
Passos de simulação	5062016
Número de episódios	6503

Para este problema inicial, fica claro que a programação genética multigênica pode gerar um indivíduo capaz de guiar o veículo até o alvo.

4.5.2 Problema 2

Neste problema é tratado o caso em que o robô deve chegar ao lugar desejado com uma determinada orientação. A única diferença na implementação do algoritmo, para este caso, reside na mudança da função de recompensa (Equação 19). Será considerado um ângulo desejado igual ao de partida, isto é, zero.

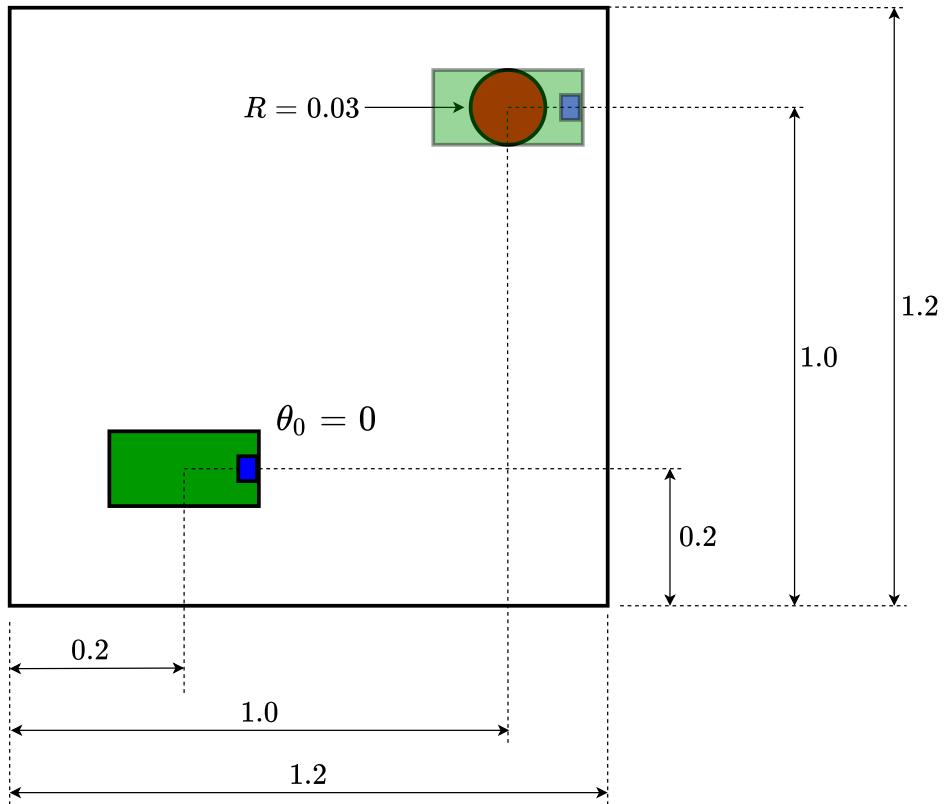


Figura 79: Visão geral do problema 2.

A função de recompensa deve levar em conta o ângulo de orientação, quando o robô chega ao alvo. Portanto, utiliza-se a Equação 19, com $\theta_{ref} = 0$. A aptidão de um indivíduo é tal como descrita na Equação 21

$$A(t) = r(t), \forall t < T$$

$$A_{tot} = \sum_{t=0}^T r(t), \quad r(t) = \begin{cases} -1000, & O(t) \notin \mathcal{S}_O \\ 2000 \cdot |\theta|, & d \leq R \\ -d & \end{cases} \quad (21)$$

$$\bar{A} = A_{tot}$$

Foram utilizados os mesmos parâmetros do caso anterior (Tabela 19) para a obtenção dos resultados a seguir:

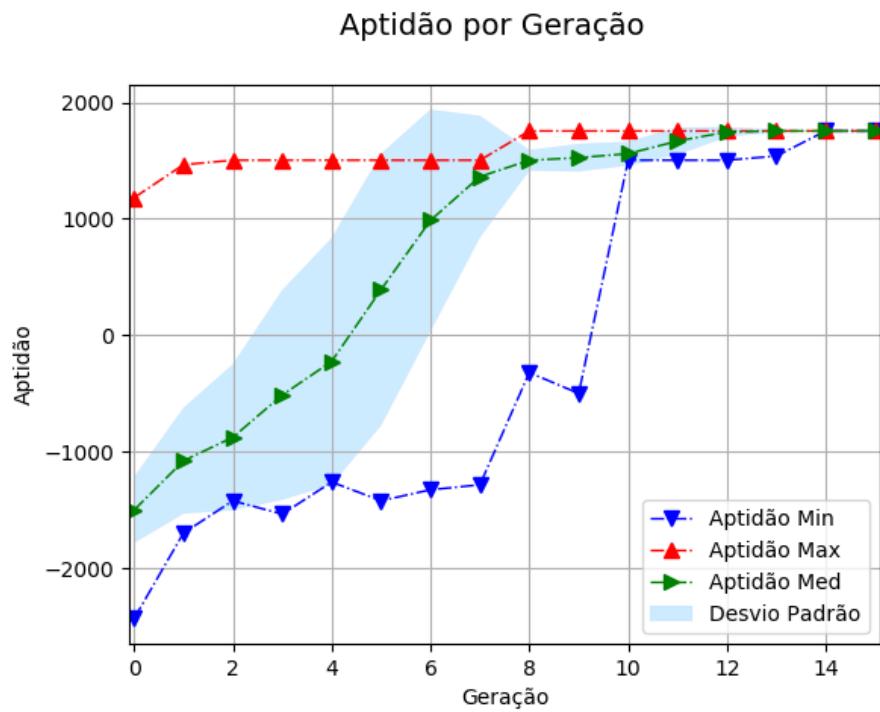


Figura 80: Aptidão dos indivíduos.

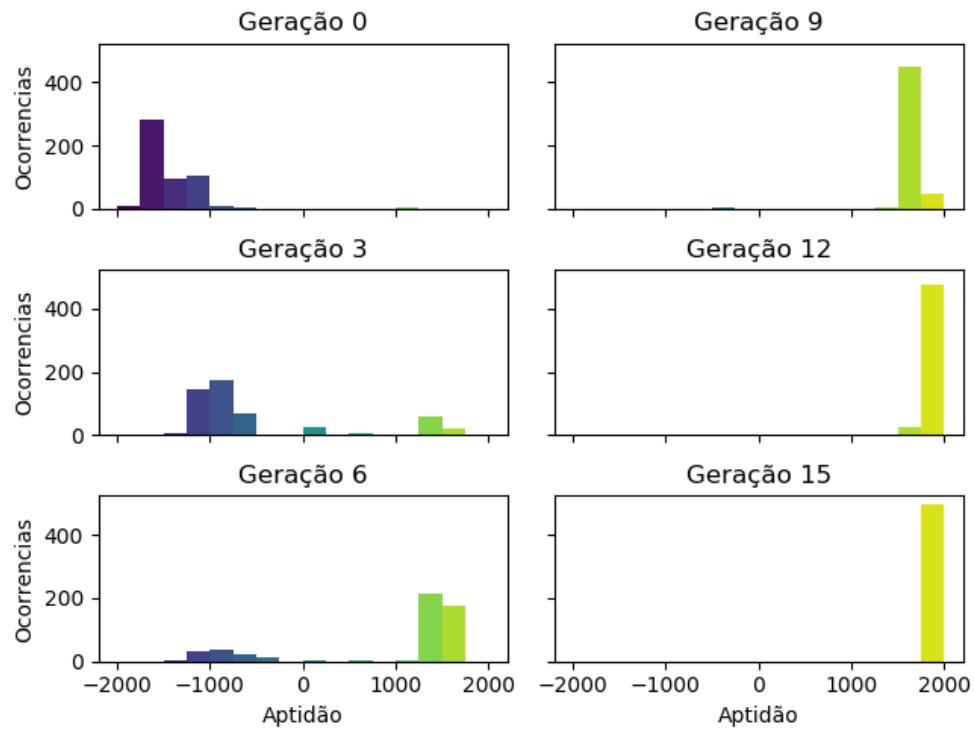


Figura 81: Histograma das aptidões.

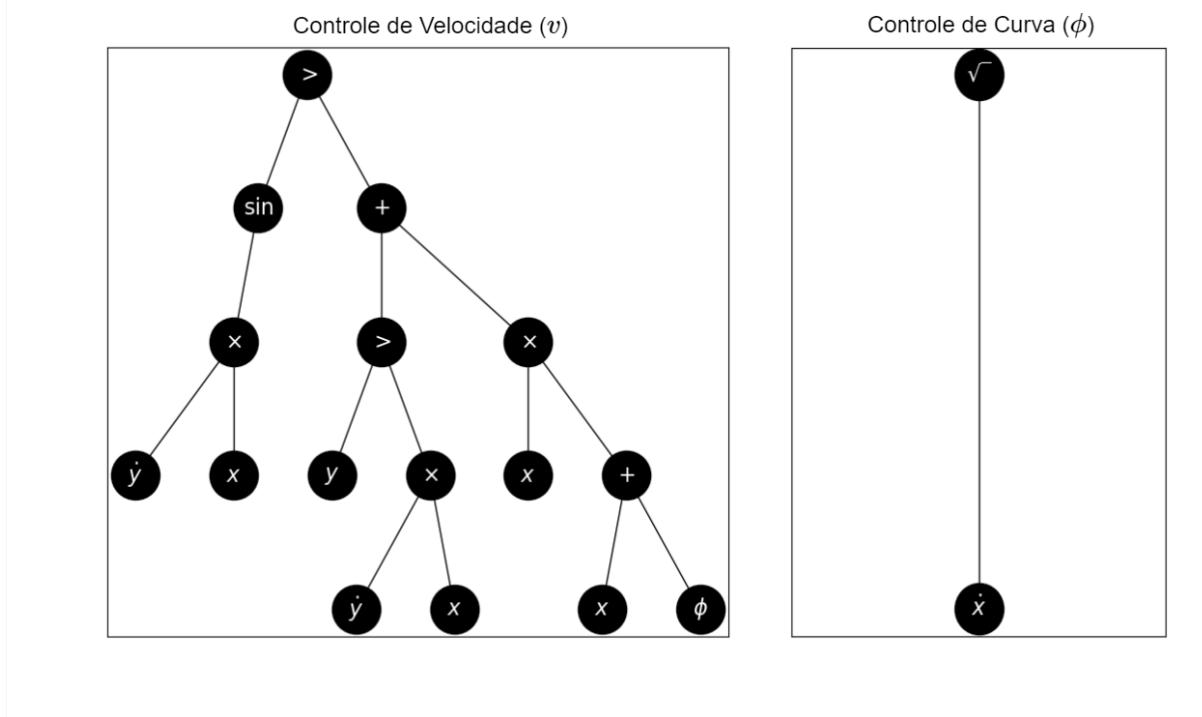


Figura 82: Indivíduo que obteve melhor desempenho.

É possível perceber nas Figuras 83 e 84 que o indivíduo, quando o veículo está próximo ao objetivo, gera valores para v e ϕ que alternam entre o mínimo e máximo permitidos, causando uma mudança brusca na orientação do veículo sem grandes alterações na sua posição.

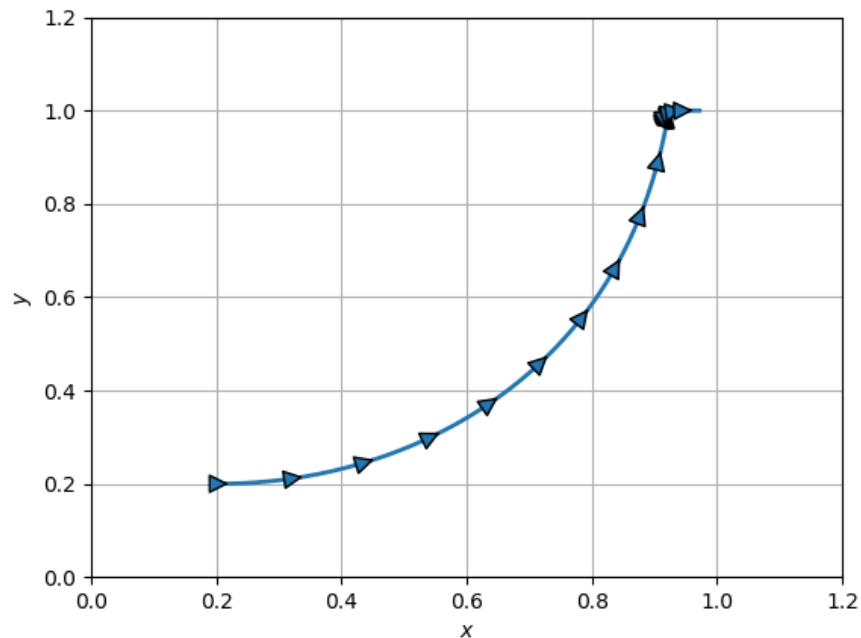


Figura 83: Trajetória do indivíduo da Figura 82.

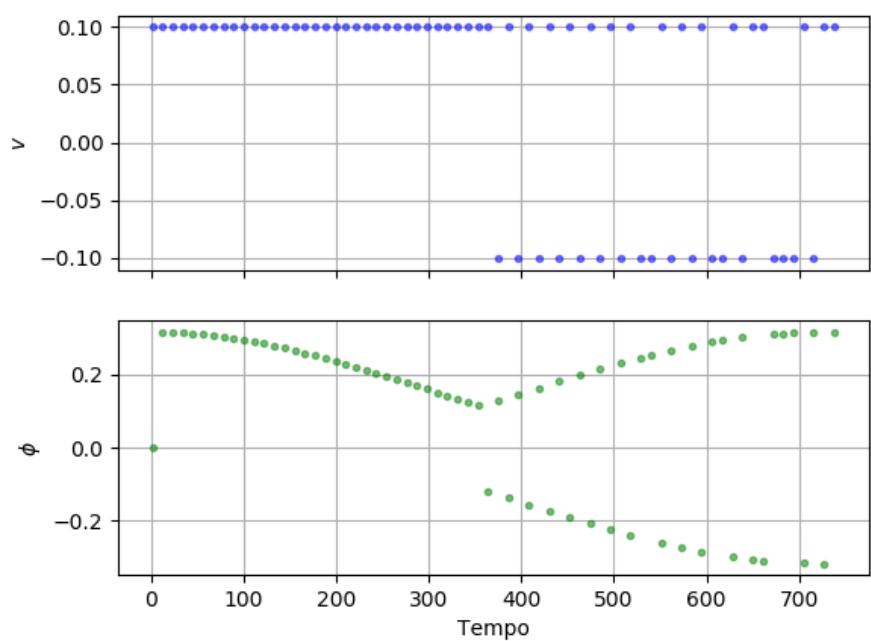


Figura 84: Funções temporais geradas.

O custo computacional, associado à execução do algoritmo, pode ser visto na Tabela 21.

Tabela 21: Problema 2: custo computacional.

Tempo de execução (h:m:s)	00:35:43
Passos de simulação	5290938
Número de episódios	6452

Na Figura 83 é possível notar que o objetivo proposto foi alcançado.

4.5.3 Problema 3

Neste problema é considerado o caso em que a pose inicial do robô é aleatória, tornando-o semelhante aos problemas iniciais da biblioteca Gym. Não há um objetivo associado à orientação do robô, ou seja, a função de recompensa utilizada é a Equação 18. O controle é exercido por meio das variáveis v e $\dot{\phi}$, portanto, a simulação possui uma maior conformidade com a situação real.

Devido à aleatoriedade da pose inicial, é necessário aumentar o número de simulações para cada indivíduo, o que garante que os indivíduos selecionados tenham uma boa capacidade de generalização. A Figura 85 mostra o problema proposto.

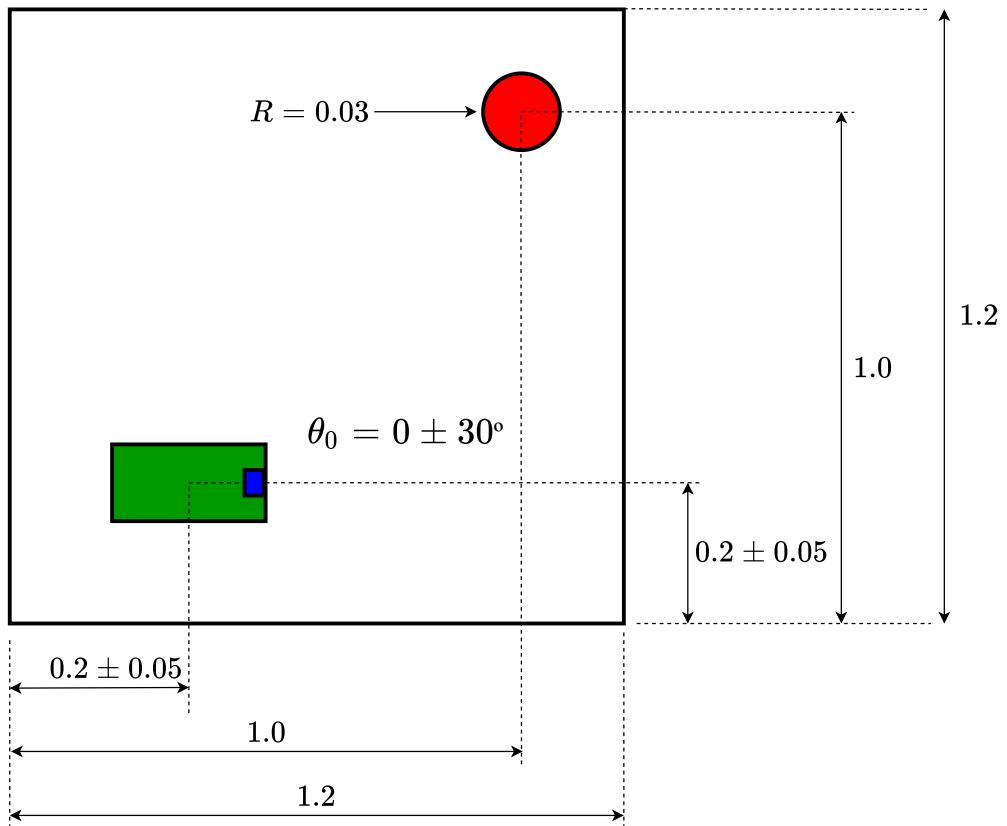


Figura 85: Visão geral do problema 3.

Pode-se perceber na Figura 85 os valores iniciais possíveis para a posição e orientação do robô.

Os parâmetros da programação genética são os mesmos da Tabela 19, com exceção do **número de simulações** definido como **30** e o **número de gerações, 30**.

A função de aptidão deve levar em conta todos os episódios de simulação do indivíduo:, conforme indica a Equação 22.

$$A(t) = r(t), \forall t < T$$

$$A_{tot}^{ep} = \sum_{t=0}^T r(t), \quad r(t) = \begin{cases} -1000, & O(t) \notin \mathcal{S}_O \\ 2000, & d \leq R \\ -d & \end{cases} \quad (22)$$

$$\bar{A} = \sum_{ep=1}^{40} A_{tot}^{ep}$$

As Figuras 86 e 87 mostram as estatísticas relacionadas à aptidão da população.

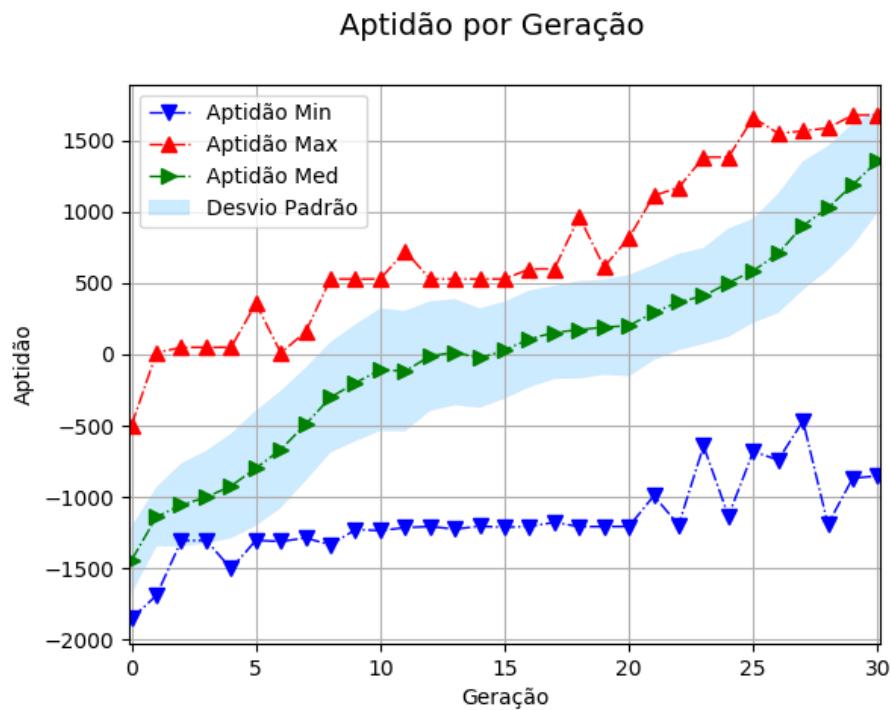


Figura 86: Aptidão da população ao longo de 30 gerações.

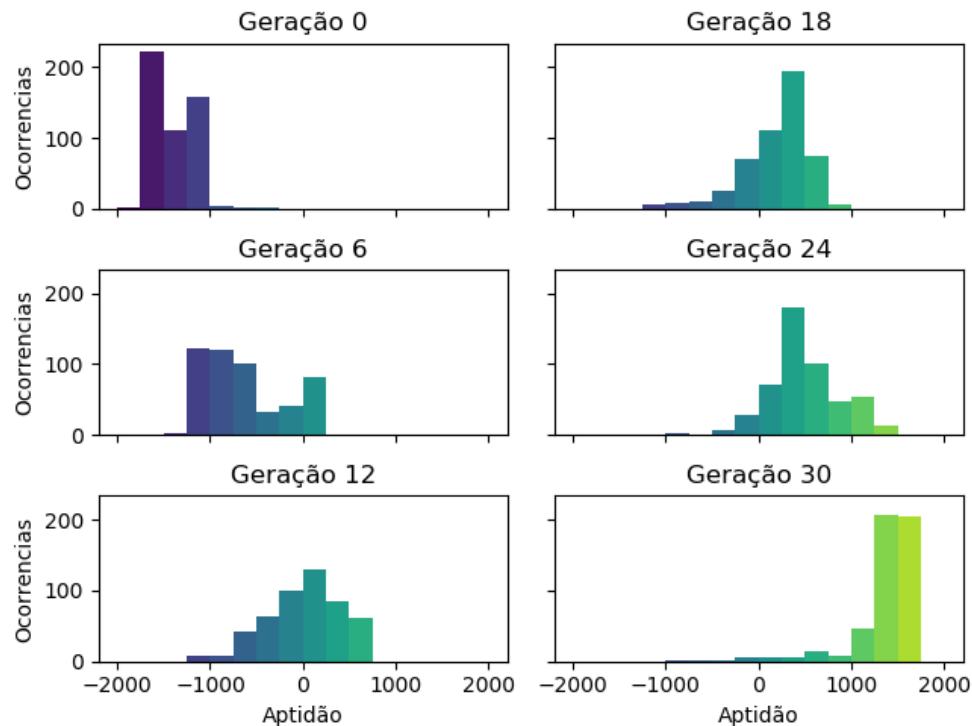


Figura 87: Histograma de aptidões.

O melhor indivíduo encontrado pode ser visto na Figura 88.

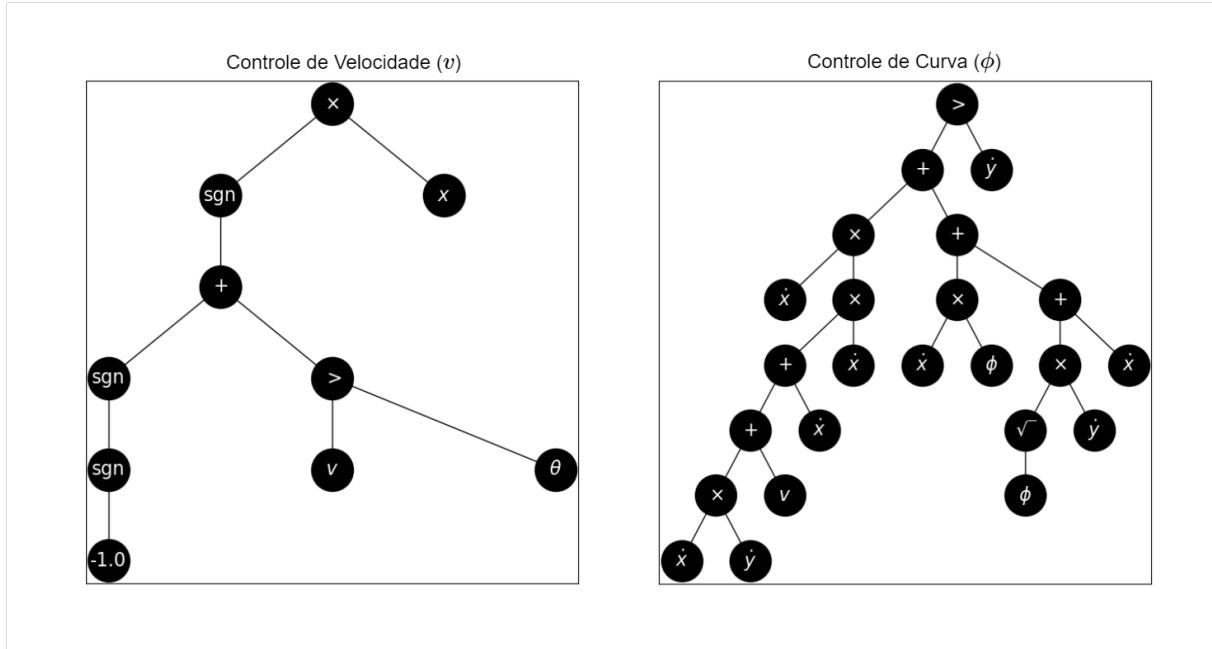


Figura 88: Árvores de controle do indivíduo que apresentou melhor desempenho.

Algumas trajetórias exibidas pelo indivíduo da Figura 88, com os valores extremos da condição inicial, podem ser vistas nas Figuras 89, 90 e 91. Na parte superior esquerda de cada figura, é possível observar a aptidão obtida pelo agente.

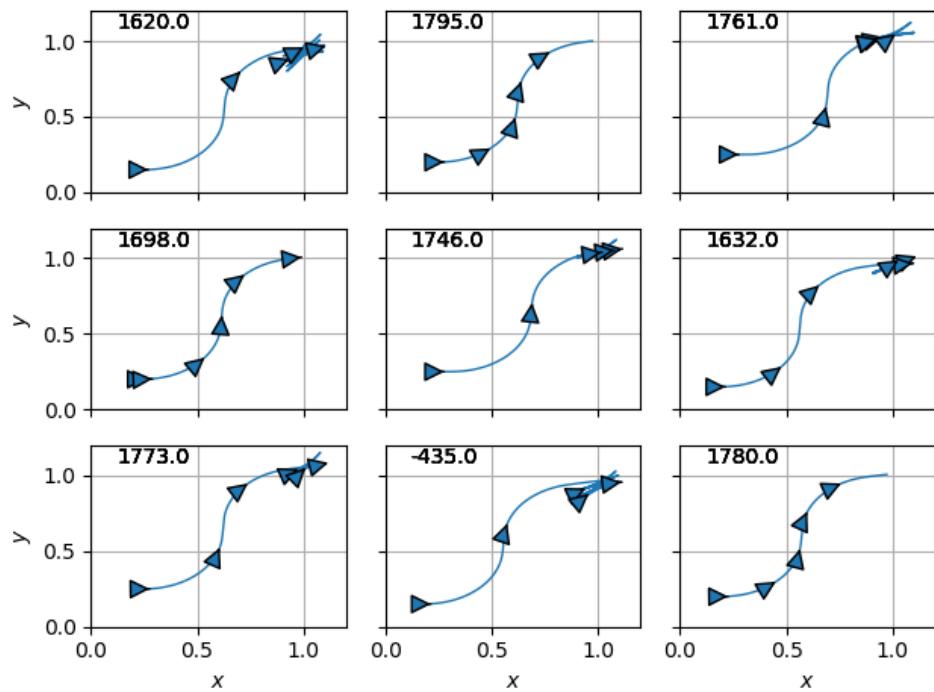


Figura 89: Trajetórias exibidas pelo melhor indivíduo com ângulo de orientação inicial igual a 0. As coordenadas X_0 e Y_0 são os valores extremos possíveis (0.15 e 0.20) e o valor médio (0.20).

Considerando os mesmos valores de X_0 e Y_0 iniciais, porém com o ângulo de orientação inicial igual a 30° e -30° , foram obtidos os gráficos da Figura 90 e 91, respectivamente.

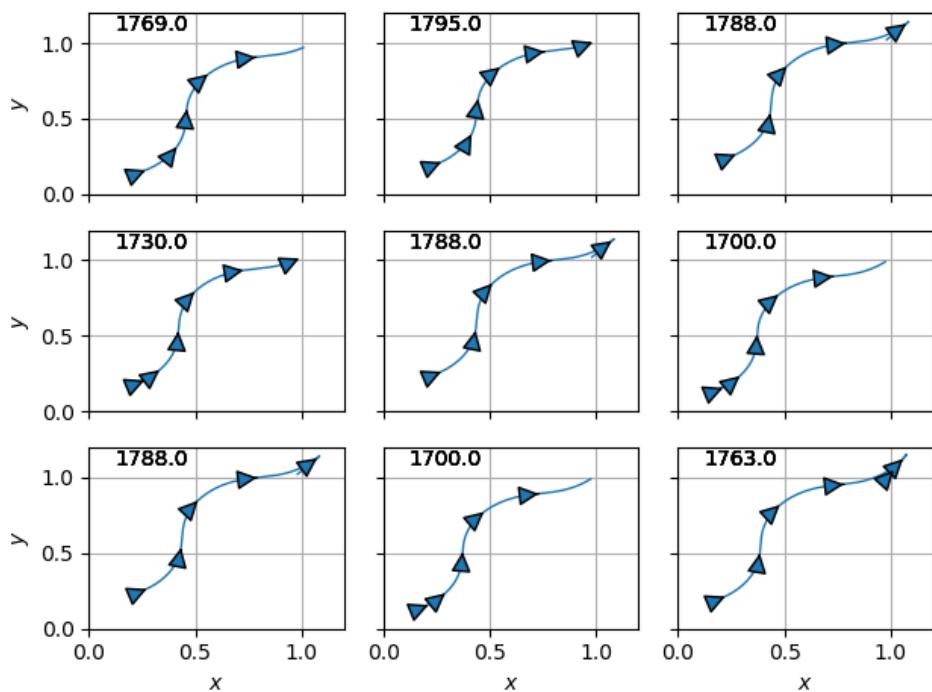


Figura 90: $\theta_0 = 30^\circ$

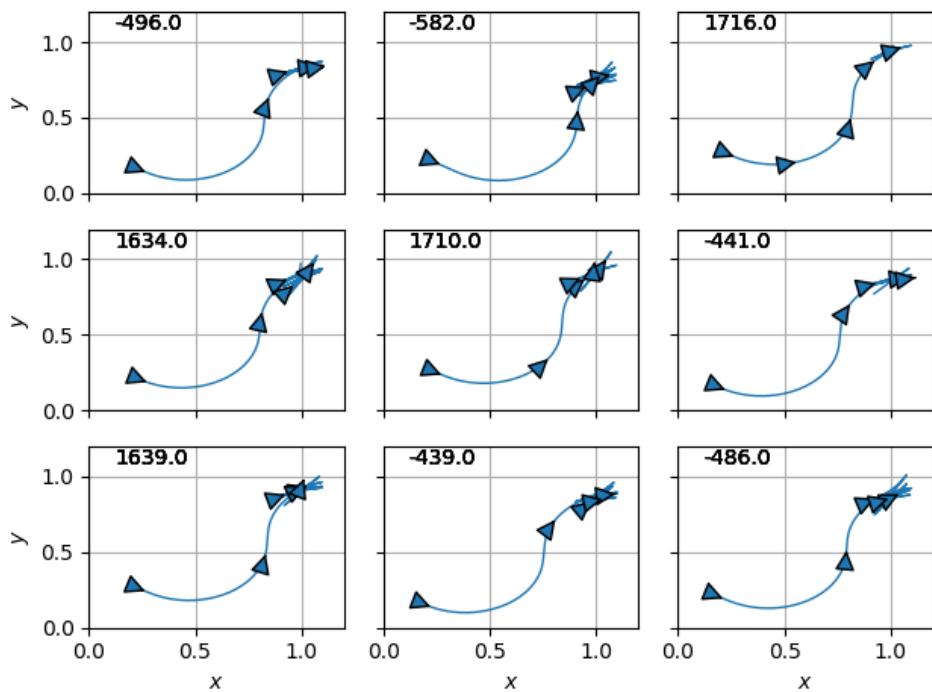


Figura 91: $\theta_0 = -30^\circ$

Observa-se que as aptidões positivas indicam que o robô chegou ao local alvo (com tolerância de 3 cm).

Para verificar a taxa de sucesso do robô em atingir o local alvo, foram consideradas as situações iniciais formadas pela combinação de 10 valores igualmente espaçados para cada variável (X_0 , Y_0 e θ_0), dentre os limites estabelecidos, totalizando **1000 simulações** com condições iniciais diferentes.

O alvo foi alcançado em 770 episódios, indicando uma taxa de sucesso aproximada de 77%. O custo computacional pode ser observado na Tabela 22.

Tabela 22: Custo computacional para o problema 3.

Tempo de execução (h:m:s)	37:14:08
Passos de simulação	333557152
Número de episódios	374760

4.5.4 Problema 4

O último problema se assemelha ao caso anterior, entretanto, deseja-se também que o robô atinja uma determinada orientação ao fim da trajetória. Isto é, a única diferença reside na função de recompensa, onde será utilizada a Equação 19, para o cálculo da aptidão, conforme indica a Equação 23.

$$A(t) = r(t), \forall t < T$$

$$A_{tot}^{ep} = \sum_{t=0}^T r(t), \quad r(t) = \begin{cases} -1000, & O(t) \notin \mathcal{S}_O \\ 2000 \cdot |\theta|, & d \leq R \\ -d & \end{cases} \quad (23)$$

$$\bar{A} = \sum_{ep=1}^{40} A_{tot}^{ep}$$

A situação proposta pelo problema 4 está esquematizada na Figura 92.

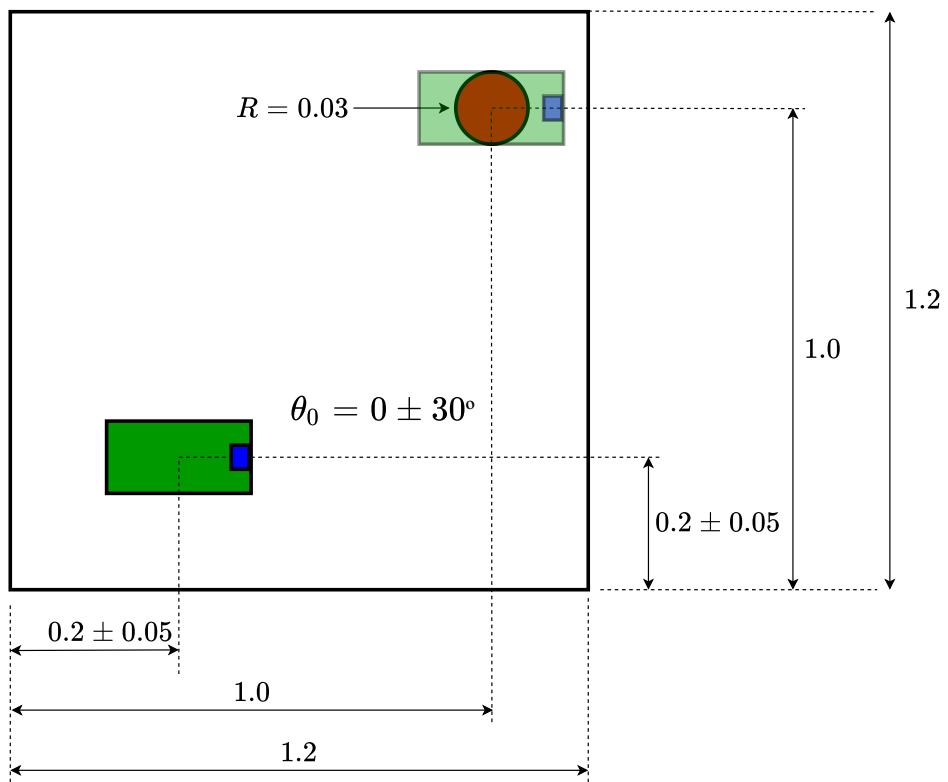


Figura 92: Visão geral do problema 4.

Os resultados obtidos podem ser verificados nas Figuras 93 a 96.

Aptidão por Geração

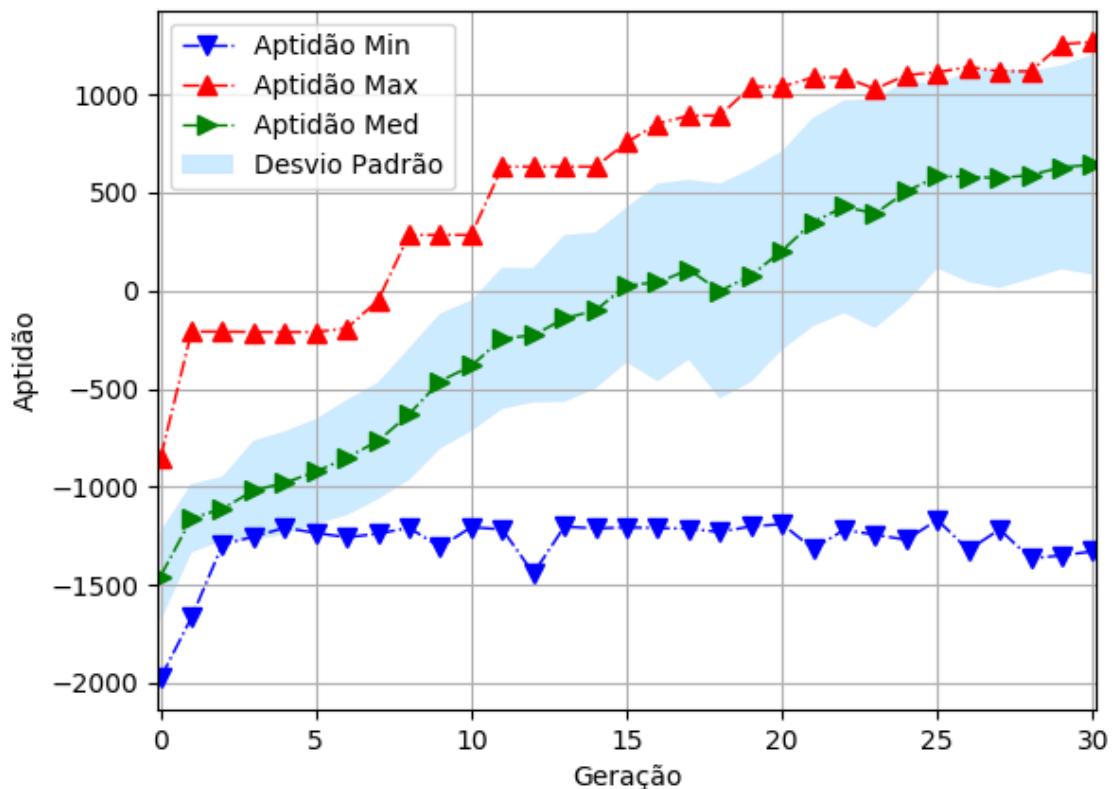


Figura 93: Aptidão ao longo das gerações para o problema 3.

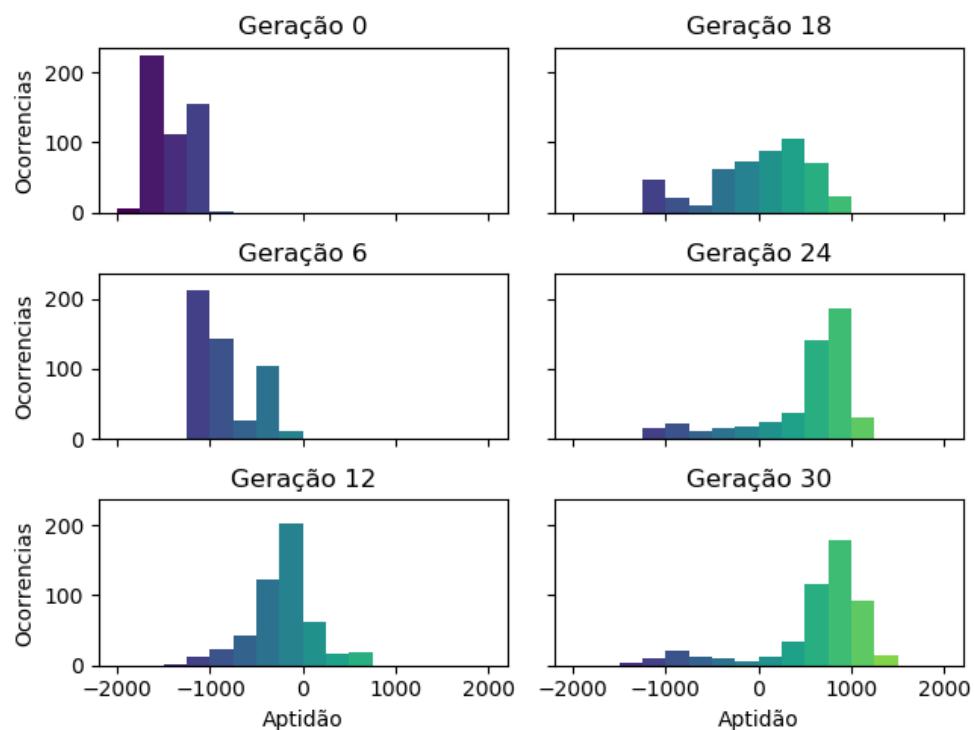


Figura 94: Histograma das aptidões.

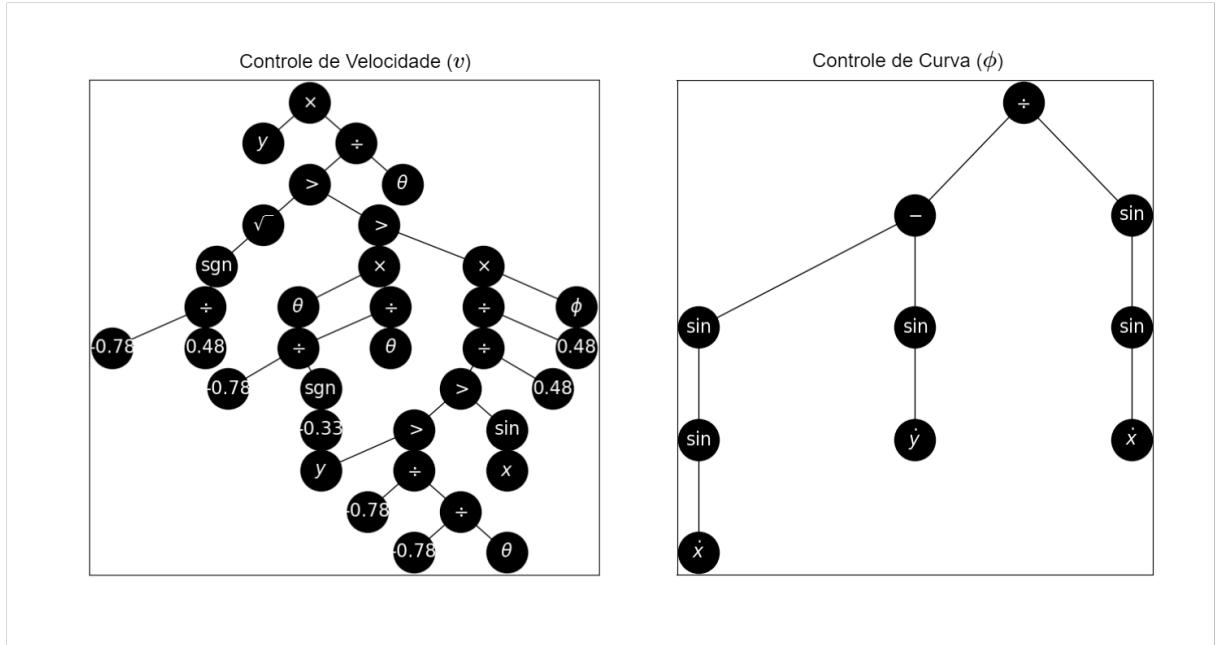


Figura 95: Árvores de controle do indivíduo de melhor desempenho encontrado.

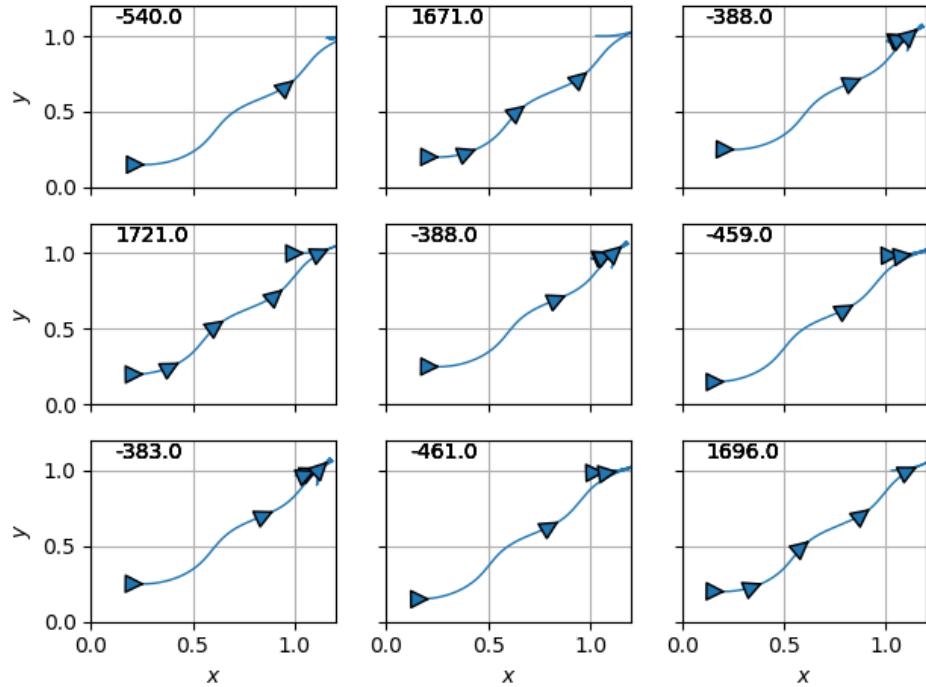


Figura 96: Trajetórias exibidas pelo melhor indivíduo com ângulo de orientação igual a 0. As coordenadas X_0 e Y_0 são os valores extremos (0.15 e 0.20) e o valor médio (0.20).

Novamente consideramos os mesmos X_0 e Y_0 , com diferentes ângulos de orientação inicial. As Figuras 90 e 98, mostram os resultados obtidos.

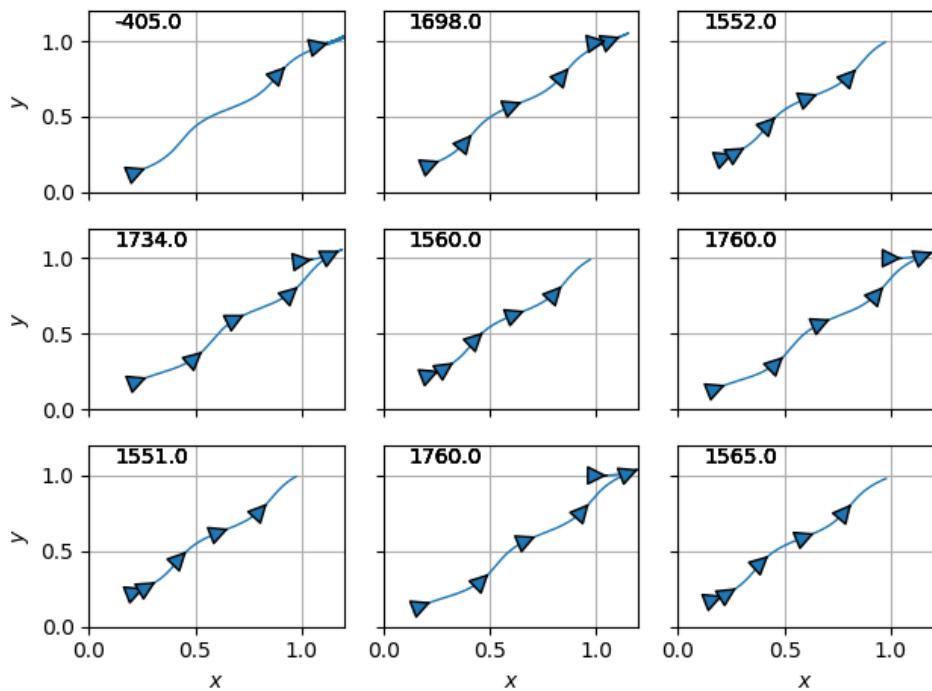


Figura 97: $\theta_0 = 30^\circ$

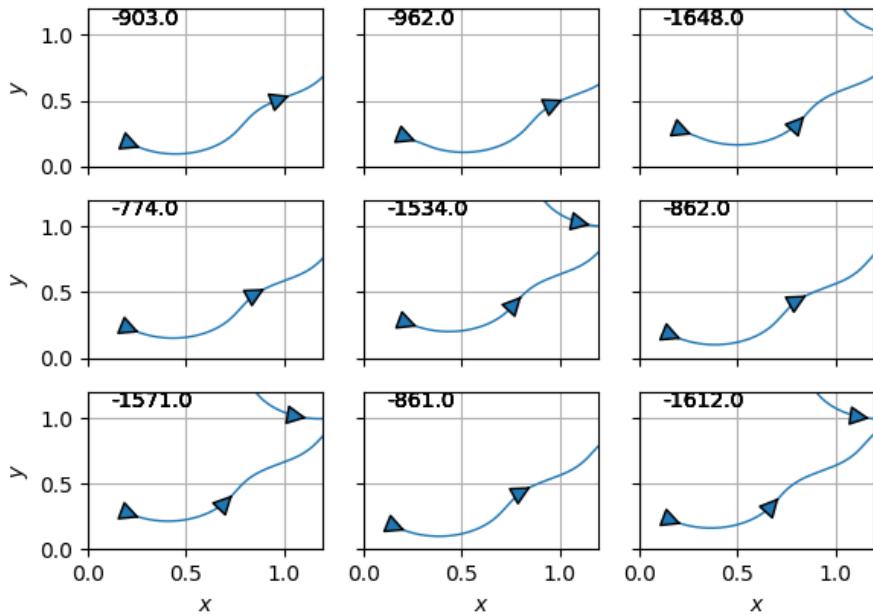


Figura 98: $\theta_0 = -30^\circ$

Ao considerar, de forma semelhante ao problema anterior, 1000 simulações com condições iniciais distintas, a taxa de sucesso obtido foi de 51.9%. Como o problema em questão formula um objetivo ao redor do ângulo de chegada, foi obtida a média dos ângulos de orientação do carro. O valor encontrado foi 20° .

Tabela 23: Problema 4: custo computacional.

	PG
Tempo de execução (h:m:s)	33:47:56
Passos de simulação	289651497
Número de episódios	377340

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.

As operações de cruzamento, mutação e replicação, possuem diferentes propósitos na busca por soluções. Esta busca é guiada, em grande parte, pela função de aptidão; isto é, busca-se 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 a situação atual, influenciada, em parte, pelo agente.

Os algoritmos de aprendizagem por reforço utilizam uma função de recompensa, que fornece um retorno ao agente, à cada instante de tempo. Muitas vezes essa função de recompensa é similar a uma função custo, logo, pode ser utilizada como critério de 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 outra área de estudo, a 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” simples, o que permite a comparação eficaz de algoritmos de aprendizagem de máquinas. Apesar de ser direcionada a algoritmos de aprendizagem por reforço, é possível utilizar a abordagem evolucionária. A implementação dos agentes foi feita a partir da biblioteca DEAP, que permite criar todas as etapas do ciclo evolutivo 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. Entretanto, o tempo de execução diminui consideravelmente com o 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 os 500 passos de simulação, em cada um dos 10 episódios com diferentes estados iniciais. 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 são membros importantes do conjunto primitivo, 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. Os resultados obtidos através da programação genética foram similares aos alcançados 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 45 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 às 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-

dades de ocorrência dos operadores genéticos, tenham um efeito positivo no desempenho do algoritmo. Os hiperparâmetros do algoritmo DDPG não foram alterados, a 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.

Nos problemas da biblioteca Gym, a função de recompensa foi utilizada como medida de aptidão dos indivíduos. Devido à relação intrínseca dessa função com os processos de decisão de Markov, a recompensa deve ser, a cada instante de tempo, uma função de três argumentos: estado atual, ação e estado futuro. Isto impõe algumas restrições no direcionamento do comportamento do agente. Por exemplo, no problema do carro na ladeira, supondo que a busca inicial da PG, promovida pela inicialização da população, não seja capaz de produzir um indivíduo que atinja o objetivo, a evolução seria lenta ou não existente, já que todos os indivíduos apresentariam a menor aptidão possível. Contudo, a PG não impõe as mesmas restrições na avaliação do agente. Uma solução simples seria determinar como critério de aptidão a maior distância percorrida montanha acima, o que não poderia ser feito de forma simples na abordagem de aprendizagem por reforço. Pode-se concluir que na programação genética há um grau de liberdade maior na definição do critério de desempenho.

Em relação ao carro robô, foi possível notar que a programação genética multigênica pôde resolver o problema da trajetória de um veículo autônomo, que parte de um ponto fixo e alcança uma determinada posição e orientação. Os problemas 3 e 4 mostraram que é possível, muitas vezes, alcançar o objetivo partindo de posições e orientações iniciais

aleatórias. Alguns testes foram realizados considerando uma variabilidade maior da *pose* inicial, entretanto, não foram obtidos bons resultados. Em algumas situações iniciais foi possível perceber o movimento de *zig-zag*, para alcançar o objetivo em um pequeno espaço físico. Vale notar que os algoritmos DQN e DDPG não são eficazes na resolução deste problema, e se mostra necessário a utilização de extensões ao processo de decisão de Markov e outros artifícios que buscam facilitar a exploração do espaço de observação. Além disso, a programação genética apresenta uma viabilidade maior de implementação, tendo em vista um veículo autônomo microcontrolado de pequeno porte. Enquanto os algoritmos de aprendizagem por reforço atuais geram políticas representadas por redes neurais, a programação genética produz expressões algébricas que são mais facilmente interpretadas [25].

Em suma, este trabalho mostrou como integrar a formulação de problemas de aprendizagem por reforço com a aplicação da programação genética. Python é uma das principais linguagens de programação para o estudo e desenvolvimento de aplicações na área de aprendizagem de máquinas e as bibliotecas DEAP e Gym são referências em suas áreas, e buscam facilitar a implementação de novos algoritmos, assim como padronizar a verificação do desempenho de diferentes métodos.

Trabalhos futuros incluem a possibilidade de melhorar a implementação do ambiente criado para o carro robô, buscar novas implementações que integrem a programação genética com a aprendizagem por reforço e verificar o desempenho da PG em outros ambientes propostos pela biblioteca Gym.

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.
- [24] POLACK, P. et al. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In: IEEE. *2017 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.], 2017. p. 812–818.

- [25] HEIN, D.; UDLUFT, S.; RUNKLER, T. A. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 76, p. 158–169, 2018.

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 24: 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 25: 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 26: Condições iniciais do ambiente *Pendulum-v0*.

Variável	Valor Mínimo	Valor Máximo
s	-0,6	-0,4
\dot{s}	0	0

Apêndice C Hardware e Software Utilizados

Tabela 27: Características do computador e dos programas utilizados para a execução do algoritmo.

Componente / Programa	
Processador	Ryzen 5 1600
Memória	Corsair Vengeance LPX 2x8GB DDR4 3000Mhz
Placa-Mãe	Asrock A320M-HD
Interpretador de Python	Anaconda
IDE	Pycharm Community