Aprendizado por Reforço utilizando Algoritmos Evolucionários

Antonioni Barros Campos
Programa de Engenharia Elétrica - COPPE
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
antonioni.campos@ufrj.br

***Abstract*—O *Framework* de Aprendizado por Reforço vem, nos últimos anos, sendo estudado no contexto de controle e tem como ideia central realizar um processo de aprendizado através da interação de um agente com o ambiente. Uma das abordagem para o problema seria trabalhar no espaço das políticas utilizando Algoritmos Evolucionários na busca por uma política ótima para a interação agente-ambiente. O objetivo do estudo é de comparar algumas classes de Algoritmos Evolucionários no âmbito do problema de Aprendizado por Reforço.**

***Index Terms*—algoritmos evolucionários, aprendizado por reforço, estratégias de evolução, programação evolucionária**

I. Introdução

O Aprendizado por Reforço é um campo da inteligência computacional que tem como ideia desenvolver algoritmos que aprendam através da interação com um ambiente com um objetivo de atingir um certo objetivo predefinido. Mais especificamente, problemas de Aprendizado por Reforço envolvem aprender o que fazer (como mapear observações a ações) para maximizar um sinal numérico de recompensa [1].

Existem duas classes distintas de métodos de Aprendizagem por Reforço. Métodos que buscam no espaço das funções de valor e métodos que buscam nos espaços de políticas. Para a primeira classes, existem os métodos de diferença temporal (TD, *Temporal Difference*) e para o segundo tipo de classes, pode-se utilizar uma abordagem utilizando algoritmos evolucionários. Métodos de busca no espaço de políticas mantém representações explicitas de políticas e as modificam através de vários operadores de busca [2].

O objetivo principal desse trabalho é comparar diferentes Algoritmos Evolucionários, mais precisamente os algoritmos de Estratégia de Evolução (ES, *Evolution Strategy*) e Programação Evolucionária (EP, *Evolutionary Programming*), para a busca de uma política ótima π^* representada por uma rede neural densamente conectada com uma única camada oculta em um problema de pouso de uma nave simulada. Além disso, um objetivo secundário é de realmente conseguir, com o processo de evolução, fazer com que a nave tenha um comportamento esperado de controle e consiga pousar com sucesso.

II. Teoria

A. Aprendizado por Reforço

Todos os problemas de Aprendizado por Reforço "envolvem interações entre um agente decisório ativo e seu ambiente, dentro do qual o agente busca atingir um objetivo apesar da incerteza sobre seu ambiente. As ações dos agentes podem afetar o estado futuro do ambiente, afetando assim as opções e oportunidades disponíveis para o agente em momentos posteriores" [1]. Definindo as entidades da estrutura de um problema de Aprendizado por Reforço, o aprendiz e tomador de decisão é chamado de **agente**. Tudo com o que o agente interage é chamado de **ambiente**. Essas interações ocorrem continuamente, o agente tomando decisões e o ambiente respondendo e apresentando novas situações.

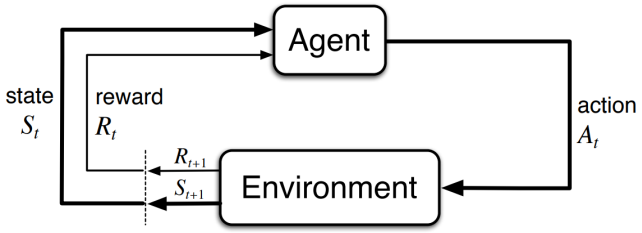

Fig. 1: Interação Agente-Ambiente [1]

Para este trabalho, a interação entre agente e ambiente foi considerada em tempo discreto, $t = 0, 1, 2, 3, \ldots$. Para cada tempo t, o agente recebe uma representação do *estado* do ambiente, $S_t \in S$, onde S é o conjunto de todos os possíveis estados, e baseado nisso seleciona uma *ação*, $A_t \in A(S_t)$, onde $A(S_t)$ é o conjunto de ações possíveis para o estado S_t. Um passo de tempo depois, em consequência da ação tomada, o agente recebe uma *recompensa* numérica, $R_{t+1} \in R \subset \mathbb{R}$ e situa-se num novo estado S_{t+1} [1]. Pode-se ver essas relações, na forma de fluxograma, na Figura 1.

Para cada instante de tempo, o agente toma uma decisão, a partir de todas as possibilidades de ações, de qual ação vai tomar de acordo com o seu estado atual. Este mapeamento entre estados e possíveis ações é chamado de política e é denotado por π_t, onde $\pi_t(a|s)$ é a probabilidade de $A_t = a$ se $S_t = s$ [1].

Problemas de aprendizado por reforço podem ser classificados em uma na qual a interação agente-ambiente naturalmente se decompõe em uma sequência de episódios separados (tarefas episódicas) e outra em que não ocorre (tarefas contínuas). Para o trabalho, foi adotado um problema de

tarefas episódicas, que possuem uma quantidade finita de períodos de tempo (Problema descrito na seção III).

B. Algoritmos Evolucionários

Algoritmos Evolucionários são uma classe de algoritmos inspirados no processo de evolução natural. Uma descrição inicial e abstrata da base dos algoritmos evolucionários é dada a seguir. Para um dado ambiente preenchido por uma população de indivíduos que lutam pela sobrevivência e reprodução. A aptidão desses indivíduos é determinada pelo ambiente e descreve o quão bem eles tiveram sucesso em atingir o objetivo [3]. Na prática, o que temos é uma função a ser maximizada, criamos aleatoriamente um conjunto de soluções candidatas (população). Com isso, aplicamos a função para essas possíveis soluções como uma medida de aptidão. Com base nesses resultados, são escolhidos os melhores candidatos para semear a próxima geração através de recombinação (*crossover*) e mutação. A partir do resultado dessas operações, é gerado, a partir dos pais, um conjunto de novos candidatos (os filhos). Esse processo é repetido até atingir-se indivíduos com aptidão suficiente (solução) ou atingido um limite computacional previamente estabelecido. O pseudocódigo é descrito na Figura 2.

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Fig. 2: Pseudocódigo de um típico Algoritmo Evolucionário [3]

No trabalho, foram utilizadas duas classes de Algoritmos Evolucionários chamados Estratégias de Evolução (Evolution Strategies, ES) e Programação Evolucionária (Evolutionary Programming, EP).

Ambos utilizam o conceito de auto-adaptação que tem como ideia de que o tamanho do passo da mutação não é definido pelo usuário. Ao invés disso, o valor do distúrbio σ coevolui com a solução $\bar{x}$ [3]. Para atingir esse comportamento, é essencial alterar o valor de σ a priori e, em seguida, modificar os x_i utilizando os novos valores de σ [3]. No trabalho, foi utilizado a mutação não correlacionada com n tamanhos de passo que tem como ideia tratar cada dimensão da representação do indivíduo separadamente, ou seja, cada dimensão tem seu tamanho de passo resultando em $\langle x_1, ..., x_n, \sigma_1, ..., \sigma_n \rangle$. O modo ao qual a mutação é aplicada é descrita pelas equações a seguir.

$$\sigma_i' = \sigma_i \cdot e^{\tau' \cdot \mathcal{N}(0,1) + \tau \cdot \mathcal{N}_i(0,1)} \quad (1)$$

$$x_i' = x_i + \sigma_i' \cdot \mathcal{N}_i(0,1) \quad (2)$$

Onde $\tau' \propto 1/\sqrt{2n}$, e $\tau \propto 1/\sqrt{2\sqrt{n}}$. No trabalho, os valores adotados para τ' e τ foram exatamente os descritos pelos lados direitos das proporcionalidades. Também foi adotado σ' mínimo de $\varepsilon_0 = 0.0001$.

A seguir, um resumo das etapas adotadas para os algoritmos típicos de Estratégias de Evolução e Programação Evolucionária utilizados no trabalho (Tabela I). Especificamente para a Programação Evolucionária, também foi implementado a mutação utilizando a distribuição de Cauchy no lugar da Gaussiana (Fast Evolutionary Programming, FEP).

Tabela I: Resumo dos Algoritmos Evolucionários utilizados no trabalho

	Estratégias de Evolução	**Programação Evolucionária**
Representação	Representação Contínua	Representação Contínua
Seleção dos Pais	Aleatória e Uniforme	Determinístico (todos os pais sofrem mutação gerando um filho)
Recombinação	Discreta e Intermediária	NA
Mutação	Gaussiana não correlacionada com n tamanhos de passos	Gaussiana (EP) e Cauchy (FEP) não correlacionada com n tamanhos de passos
Seleção dos Sobreviventes	Determinística (Elitismo) a partir dos filhos	Determinística (Elitismo) a partir dos pais e filhos

III. Experimento

Para a estrutura de Aprendizado por Reforço utilizada no trabalho, foi adotado o ambiente Gym [4] que é uma interface para Aprendizado por Reforço implementada em Python que possui uma coleção de diversos ambientes de referência. Para o trabalho, foi adotado o ambiente *LunarLander-v2*[1] que simula o pouso de uma aeronave (Figura 3).

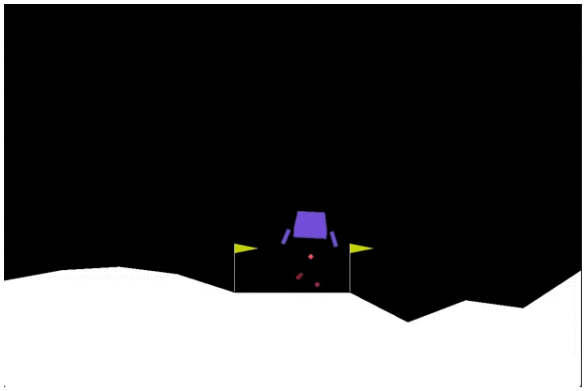

Fig. 3: Módulo de aterrissagem lunar (Lunar Lander)

[1]Documentação: https://www.gymlibrary.ml/environments/box2d/lunar_lander/

A. Espaço das Ações

Existem 4 (quatro) opções de ações discretas disponíveis para o agente. Não fazer nada (0), ativar motor esquerdo (1), ativar motor principal (2) e ativar motor direito (3).

B. Espaço dos Estados

Existem 8 (oito) estados: as coordenadas do módulo de pouso $x, y \in [-1.5, 1.5]$, suas velocidades lineares $v_x, v_y \in [-5.0, 5.0]$, seu ângulo $\theta \in [-3.14, 3.14]$, sua velocidade angular $\omega \in [-5.0, 5.0]$ e dois valores booleanos que representam se cada perna está em contato com o solo ou não. O módulo de pouso começa no centro superior da janela de visualização com uma força inicial aleatória aplicada ao seu centro de massa.

C. Recompensa

A recompensa por se mover do topo da tela para a plataforma de pouso e parar é de cerca de 100 a 140 pontos. Se o módulo de pouso se afastar da plataforma de pouso, ele perde a recompensa. Se o módulo de pouso cair, ele recebe -100 pontos adicionais. Se parar, recebe +100 pontos adicionais. Cada perna com contato com o solo é +10 pontos. O disparo do motor principal é de -0,3 pontos a cada quadro. O disparo do motor lateral é de -0,03 pontos a cada quadro. Resolvido é 200 pontos.

D. Fim do episódio

O episódio finaliza quando:

1) O módulo de pouso cai (o corpo do módulo de pouso entra em contato com a lua)
2) o módulo de pouso sai da janela de visualização (coordenada x é maior que 1)
3) O módulo de pouso não está acordado (não se move e não colide com nenhum outro corpo)

E. Política

A política definida para o agente é baseada em uma Rede Neural densamente conectada com uma camada oculta com 128 unidades. Com o vetor de entrada dos estados com 8 dimensões e com as quatro dimensões possíveis para o vetor de saídas que representam as ações, os Algoritmos Evolucionários trabalharam com indivíduos de dimensão 1.668 ($8 \times 128 + 128 + 128 \times 4 + 4$) considerando os *bias*.

F. Algoritmos Evolucionários

Como descrito na seção II-B, os algoritmos usados no trabalho são as Estratégias de Evolução (ES) e a Programação Evolucionária utilizando distúrbio Gaussiano (EP) e de Cauchy (FEP) na mutação. Os parâmetros utilizados pelos Algoritmos Evolucionários nas execuções são listados na tabela II.

Para o algoritmo de Estratégias de Evolução, foi utilizado a quantidade de filhos de 7 (sete) vezes a quantidade de pais.

Um detalhe importante é de que o cálculo das aptidões foi realizado através da média das recompensas resultantes de três episódios distintos gerados aleatoriamente.

Tabela II: Parâmetros para os Algoritmos Evolucionários

Parâmetros	**Valores**
Gerações	1.000
Tamanho da População	100 150
Estratégia Mínima ε_0	0.0001
Perturbações	Gaussiana (ES, EP) Cauchy (FEP)

IV. Resultados

Toda análise foi realizada utilizando a linguagem de programação Python [2], dentro do ambiente Gym (ver seção III).

Os algoritmos ES, EP e FEP foram executados 2 (duas) vezes, utilizando os parâmetros da tabela II, e os resultados são os valores médios mostrados na tabela III. As medidas comparativas de qualidades dos algoritmos utilizadas foram o MBF (*Mean Best Fitness*) e o AES (*Average number of Evaluations to a Solution*). Especificamente para o AES, foi definido que o processo foi sucedido quando atingiu 5 (cinco) gerações sucessivas com valor médio de aptidão para a população acima de 50 pontos.

Tabela III: Resultados dos algoritmos

Algoritmo	**População**	**MBF**	**AES**
ES	100	95.14	460.000
ES	150	102.09	550.100
EP	100	73.81	-
EP	150	75.12	83.100
FEP	100	56.02	36.000
FEP	150	69.70	30.400

Da tabela III, nota-se que um parâmetro importante é o tamanho da população. Para todos os algoritmos executados com uma população de 150, foram obtidos melhores resultados do que os executados com uma população de 100. Também pode-se verificar que o algoritmo que obteve melhores indivíduos foi o de Estratégias de Evolução. Um outro detalhe obtidos pela tabela de resultados é que, de acordo com os resultados de AES, o algoritmo de Estratégias de Evolução precisou avaliar mais a função aptidão para atingir o patamar de 50 pontos (aproximadamente uma ordem de grandeza em relação aos outros algoritmos). Uma observação para o EP com população de 100 não atingiu a aptidão média sucessiva necessária para contabilizar o AES em nenhuma das duas execuções.

Nas Figuras 4, 5, 6, foram plotados os gráficos de melhor aptidão (em azul) e aptidão média (em laranja) por geração

[2]Implementação encontra-se em https://github.com/antonionicampos/lander-control-using-ga

para cada um dos algoritmos analisados, para as execuções com população de 150. Nota-se que, apesar do algoritmo de Estratégia de Evolução gerar indivíduos com melhor aptidão com o passar das gerações, a aptidão média converge para um valor em torno de 50 para todos os algoritmos.

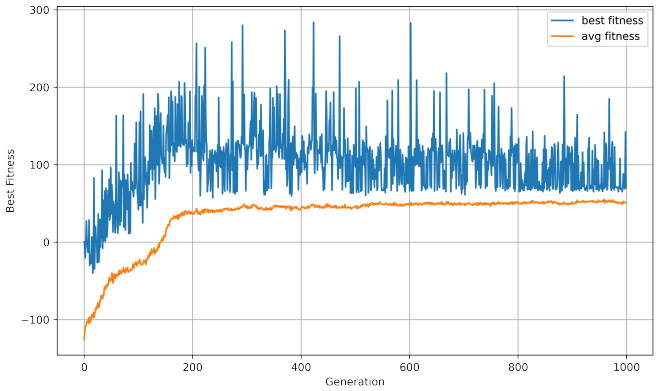

Fig. 4: Aptidão por gerações para a Estratégia de Evolução (ES), população de 150

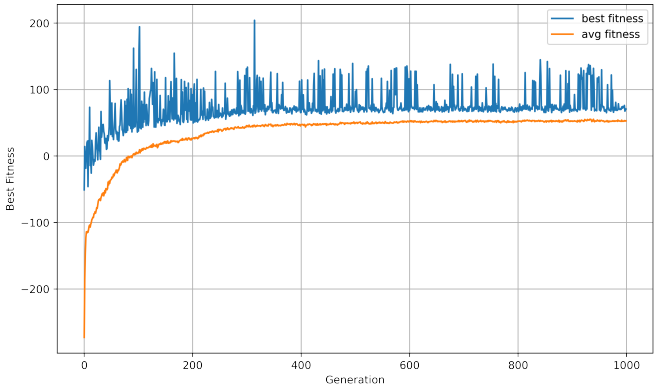

Fig. 5: Aptidão por gerações para a Programação Evolucionária (EP), população de 150

V. Conclusão

Analisando os resultados comparativos entre os algoritmos, a etapa de recombinação implementada na Estratégia de Evolução (ES) com a diversidade obtida pela grande quantidade de filhos geradas a partir dos pais em cada geração ($\lambda = 7\mu$) demonstrou ter importância no problema gerando indivíduos mais aptos combinado . Por outro lado, a Programação Evolucionária (EP/FEP) demonstrou alcançar indivíduos com aptidão relativamente alta com uma menor quantidade de chamadas à função de aptidão.

Por outro lado, devido a essa alta pressão seletiva do ES ($\lambda = 7\mu$), os algoritmos ES tiveram que chamar a função aptidão uma quantidade maior de vezes (uma ordem de grandeza

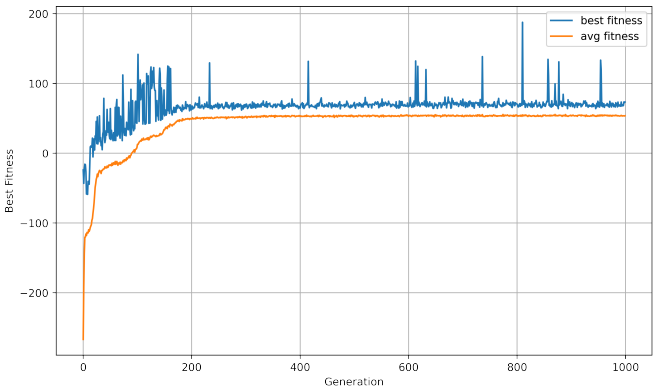

Fig. 6: Aptidão por gerações para a Programação Evolucionária utilizando Cauchy (FEP), com população de 150

em relação aos demais algoritmos) levando a um esforço computacional maior para atingir o patamar de valores médios de aptidão de 50 pontos. Mas, essa característica proveu ao ES a capacidade de explorar o espaço dos parâmetros da Rede Neural nas gerações mais avançadas gerando indivíduos mais aptos.

Analisando os valores das estratégias para todos os algoritmos, os valores para o ES, no final das 1.000 gerações estavam em valores muito altos em comparação aos algoritmos EP e FEP (FEP específicamente atingindo o mínimo de 0.0001 na sua maioria), também podendo ser constatado pelas melhores aptidões por geração nas Figuras 4, 5, 6, sendo um indicativo de que o ES não atingiu uma convergência. Uma questão que vale a pena ser avaliada é se o ES conseguiria atingir maiores patamares de aptidão para a população ou se a Rede Neural com uma única camada mostrou-se não ser suficiente para obter resultados no patamar de solução do problema (com valores médios de aptidão para a população acima de 100).

O objetivo secundário foi analisado de maneira mais subjetiva através de vídeos gerados (no formato de gif) e armazenados no repositório de códigos do trabalho [3]. O objetivo foi atingindo e pode ser verificado pelos arquivos `movie_gen_100.gif` e `movie_gen_1000.gif`. Eles mostram o comportamento do melhor indivíduo para uma execução do algoritmo ES para uma população de 150, nas gerações 100 e 1.000 respectivamente. Na geração 100, a nave não tem o mínimo de controlabilidade e, a ação predominante é de não fazer nada levando a nave chocar com o solo no final do episódio. Já na geração 1.000, a nave já possui uma controlabilidade suficiente para conseguir utilizar seus motores e leva-la para dentro da região de pouso.

Referências

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] https://github.com/antonionicampos/lander-control-using-ga

[2] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 11, pp. 241–276, 1999.
[3] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.