COMMUNICATIONS AND INNOVATIONS GAZETTE

Investigação da Eficácia de Técnicas de Nicho e Diferenciação Ambiental aplicadas a Algoritmos da Robótica Evolutiva

Brenda S. Machado[1], **Arthur H. Bianchini**[1], **Jônata T. Carvalho**[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)

{brendamachado29016,jonatatyska}@gmail.com

arthur.h.bianchini@grad.ufsc.br

Abstract. *The Evolutionary Strategies (ES) algorithm has proven to be an efficient optimization technique over the decades. Recently, an adaptation of the method, proposed by researchers from the company OpenAI, demonstrated the advantages of using ES techniques in parallel as an important alternative to the also relevant Reinforcement Learning method. Techniques that use population optimization approaches, such as evolutionary strategies, benefit from the diversity of candidate solutions in the evolutionary process. For this reason, mechanisms that preserve diversity, such as the creation of islands and niches during the evolutionary process, have been proposed and investigated in other evolutionary algorithms. This work aims to analyze how the addition of niche techniques, which include environmental differentiation between subpopulations, can be relevant to Evolutionary Robotics algorithms using the version of the ES algorithm recently proposed by OpenAI. Using the well-known double-pole balancing problem as a test task, we compared the effectiveness of solutions generated with and without the niche mechanism in the OpenAI-ES and Stochastic Steady State (SSS) algorithms. The results obtained demonstrated performance increases of approximately 8.6% and 53.5% for OpenAI-ES and SSS, respectively, when the niche mechanism is used.*

Resumo. *O algoritmo Estratégias Evolutivas (ES) têm se mostrado uma técnica eficiente de otimização ao longo das décadas. Recentemente, uma adaptação do método, proposta por pesquisadores da empresa OpenAI, demonstrou as vantagens de se usar técnicas de ES de forma paralelizada como uma importante alternativa ao também relevante método de Aprendizado por Reforço. Técnicas que utilizam abordagens populacionais de otimização como é o caso das estratégias evolutivas, se beneficiam da diversidade das soluções candidatas no processo evolutivo. Por este motivo, mecanismos que preservam a diversidade como, por exemplo, a criação de ilhas e nichos durante o processo evolutivo foram propostos e investigados em outros algoritmos evolutivos. Este trabalho tem o objetivo de analisar como a adição de técnicas de nicho, que incluem diferenciação ambiental entre as subpopulações, podem ser relevantes a algoritmos da Robótica Evolutiva utilizando a versão do algoritmo ES recentemente proposto pela OpenAI. Utilizando como tarefa de teste o bem conhecido problema do baleancemento de mastros duplos (double-pole balancing), comparamos a efetividade das soluções geradas com e sem o mecanismo de nichos nos*

algoritmos OpenAI-ES e Stochastic Steady State (SSS). Os resultados obtidos demonstraram aumentos de performance de aproximadamente 8,6% e 53.5% para OpenAI-ES e SSS, respectivamente, quando o mecanismo de nichos é utilizado.

1. Introdução

O campo da Robótica Evolutiva tem como inspiração a evolução biológica, na qual os indivíduos mais bem adaptados ao ambiente são aqueles com melhores chances de sobrevivência. Dessa forma, os agentes robóticos mais promissores são aqueles que se adaptam melhor no ambiente em que estão inseridos. Trazendo para o contexto da Robótica Adaptativa, a adaptação desses agentes deve fazer com que eles desenvolvam comportamentos que os tornem capazes de realizar uma tarefa de forma autônoma, ou seja, com o mínimo de intervenção humana possível [Nolfi 2021].

Para fazer a evolução dos agentes robóticos, isto é, para que o aprendizado deles em interação com o ambiente seja feito, técnicas como as Estratégias de Evolução [Rechenberg 1973] são utilizadas. Elas consistem em uma família de algoritmos de otimização utilizados na solução de problemas complexos. Um desses algoritmos, proposto pela OpenAI e que nessa pesquisa chamaremos de OpenAI-ES, foi apresentado em [Salimans et al. 2017], no qual analisou-se de que forma Estratégias de Evolução podem ser uma boa alternartiva em relação ao Aprendizado por Reforço. Algumas das vantagens do OpenAI-ES em relação à métodos tradicionais de Aprendizado por Reforço é que ele é altamente escalável por ser paralelizável, mais simples de implementar e mais robusto.

A pesquisa conduzida em [Whitley et al. 1998] buscou gerar comportamentos mais adaptativos, assim como melhorar a diversidade ambiental na qual os agentes poderiam atuar. Por isso, tal abordagem é uma das referências para os métodos testados neste trabalho. Em [Whitley et al. 1998], investigou-se a separação da população em ilhas, as quais possuiam mecanismos de migração periódica. Demonstrou-se que a separação em subpopulações melhorou a diversidade de soluções, uma vez que os mesmos tinham certo grau de independência e, dessa forma puderam explorar melhor o espaço de busca.

Em [Carvalho 2017], foi estudada a utilização da proposta dos nichos, uma adaptação da proposta de [Whitley et al. 1998], no campo da Robótica Adaptativa. Foi aplicada a separação da população em nichos, os quais consistem em ilhas com técnicas de diferenciação ambiental. Tal abordagem foi testada no algoritmo *Stochastic Steady State* (SSS) [Milano et al. 2017]. Os resultados apresentados pelos autores demonstraram que a separação de nichos em relação a uma população única, poderia fazer com que os agentes gerassem soluções mais adaptativas. Os algoritmos foram testados no conhecido problema chamado *double-pole*, que consiste em um carro com dois mastros presos a ele, no qual os mastros precisam ser equilibrados pelo agente. Os resultados apresentados pelos autores demonstraram a eficácia do método em relação ao SSS clássico, sendo que os agentes em ambientes com nicho foram os que melhor performaram em relação aos outros.

Um trabalho recente estendeu a proposta de [Carvalho 2017] ao OpenAI-ES, investigando se a divisão da população em nichos pode ser favorável também ao OpenAI-ES. A pesquisa [Bianchini et al. 2023a] reproduziu os experimentos com o SSS e criou um algoritmo que chamou de OpenAI-ES-NE *(Niche Evolution)*, a contraparte com ni-

chos do OpenAI-ES. Os experimentos também foram conduzidos no *double-pole* e tiveram resultados promissores, com um aumento de cerca de 9% da performance do OpenAI-ES-NE em relação ao OpenAI-ES. Isso demonstra que esse método pode ser favorável ao desenvolvimento de agentes sob Estratégias de Evolução.

Dessa forma, e inspirada pelos trabalhos realizados até então, essa pesquisa buscou reproduzir e analisar os resultados de [Bianchini et al. 2023a] e compará-los com os de [Carvalho 2017]. Os resultados parciais indicaram que esse método foi favorável tanto ao OpenAI-ES quanto ao SSS, com as suas contrapartes com nichos apresentando melhores performances.

2. Revisão Bibliográfica

Os algoritmos evolutivos são baseados na ideia de seleção natural, ou seja, dada uma população de soluções candidatas, pressões ambientais e seletivas levam à sobrevivência dos indivíduos mais aptos, aumentando assim a aptidão da população. Tecnicamente, o esquema de um algoritmo evolutivo é inicializar, de forma aleatória, uma população de possíveis soluções candidatas para o problema em questão, e avaliar o desempenho de cada uma delas usando uma função de aptidão normalmente definida pelo projetista. Após a avaliação, o algoritmo seleciona um grupo de indivíduos com melhor desempenho, e os permite gerar descendentes mutantes que serão avaliados no próximo ciclo. Estes ciclos se repetem até que um critério de parada seja alcançado.

Conforme apresentado em [Fogel 1997], algumas vantagens dos algoritmos evolutivos são as seguintes: (1) simplicidade conceitual; (2) ampla aplicabilidade, pois podem ser aplicados a praticamente qualquer tarefa de otimização; (3) melhor desempenho em problemas reais em comparação com métodos clássicos; (4) potencial para hibridização com outros métodos; (5) paralelismo; (6) capacidade de se adaptar a circunstâncias em mudança; (7) capacidade de se auto-otimizar, ou seja, encontrar seus melhores parâmetros por si próprio; e finalmente (8) ser capaz de resolver problemas com soluções desconhecidas.

No entanto, no espaço de busca de um problema, podem haver vários ótimos locais para os quais a função de aptidão pode convergir prematuramente. Isso pode ocorrer devido à falta de diversidade na população. De acordo com [Ekárt and Németh 2002], uma população de um algoritmo evolutiva é diversa se as amostras estiverem espalhadas em tantas regiões do espaço de busca quanto possível. Uma possível maneira de manter a diversidade da população é o uso de técnicas de ilhas, as quais tentam manter uma distância razoável entre as amostras, separando as amostras em subpopulações centradas em pontos diferentes do espaço de busca, aumentando assim a exploração do espaço e favorecendo a diversidade da população.

Algumas técnicas de nicho são descritas em [Sareni and Krahenbuhl 1998], como o compartilhamento de *fitness*, que reduz o retorno de aptidão em áreas densamente populadas em um nicho; os métodos de aglomeração, que substituem elementos semelhantes por novos; e o *clearing*, que mantém o retorno de aptidão para um certo número dos melhores indivíduos no nicho enquanto redefine o retorno de aptidão para os demais.

Além da diversidade, outro fator que pode ser importante para a aptidão da população são as condições ambientais [imp 2019, Linder and Nye 2010]. Ao usar algoritmos evolutivos na robótica, a importância das condições ambientais deve-se ao fato

de que o comportamento de um robô é altamente dependente de fatores externos, como os estímulos que os sensores recebem. Como exemplo na vida real, temos o efeito de ataques de spoofing em carros autônomos, que ocorrem emitindo sinais para um sensor *Light Detection and Ranging* (LIDAR) de um carro robô, fazendo-o perceber um obstáculo que não existe, possivelmente gerando um comportamento inesperado e potencialmente perigoso do agente [Chowdhury et al. 2020].

Esse fator torna as técnicas de nicho promissoras no contexto da Robótica Adaptativa quando os nichos têm diferentes condições ambientais, como apresentado em [Carvalho 2017], já que a população será exposta a mais condições ambientais de maneira sistemática. Isso permite que os agentes desenvolvam estratégias mais gerais do que aquelas evoluídas em um único ambiente estático [Whitley et al. 1998].

3. Métodologia Desenvolvida

Neste trabalho, reproduzimos os resultados da investigação a aplicação da estratégia de nichos no algoritmo SSS, proposta por [Carvalho 2017], e no algoritmo OpenAI-ES, proposta por [Bianchini et al. 2023a]. A implementação dos algoritmos baseados em nichos foi feita com base no método proposto por [Bianchini et al. 2023a] e adaptado nesse trabalho. A comparação do desempenho dos algoritmos foi conduzida por meio de 10 experimentos com *seeds*, números iniciais para geradores de números aleatórios, aleatórias e 10 bilhões de *steps*, ciclos de avaliação dentro do algoritmo e de acordo com o problema escolhido para a evolução, executados no simulador de comportamento robótico evorobotpy2 [Nolfi 2020] e no conhecido problema do *double-pole*. Para se certificar que todos os algoritmos estariam em suas melhores performances, dentro do simulador, utilizamos uma biblioteca de otimização de hiperparâmetros chamada irace [López-Ibáñez et al. 2016], a qual forneceu o conjunto de valores para os hiperparâmetros de cada um dos algoritmos. Além disso, a comparação foi feita com os resultados obtidos na pós-avaliação dos 10 melhores indivíduos evoluídos com 1000 condições iniciais aleatórias, para cada um dos algoritmos.

3.1. Evorobotpy2

O evorobotpy2 [Nolfi 2020] é um simulador de comportamento robótico que apresenta diversos ambientes *gym* [Brockman et al. 2016], que oferecem uma simulação eficiente de locomoção robótica, com aprendizado por algoritmos previamente implementados como, por exemplo, o SSS e o OpenAI-ES, assim como possibilita a implementação de algoritmos por parte do projetista. Para os testes com os algoritmos bases, utilizou-se o OpenAI-ES e o SSS disponibilizados pelo simulador, e as versões com nichos foram implementadas a partir destas versões base.

3.2. Double-Pole

Como o problema do *double-pole* foi usado em [Carvalho 2017] e [Bianchini et al. 2023a], utilizamos neste trabalho com o objetivo de, primeiramente, reproduzir e comparar os resultados obtidos nesses trabalhos.

O ambiente em duas dimensões consiste em um carro com um mastro longo e um curto presos à ele. O objetivo é manter os mastros equilibrados o máximo de tempo possível, enquanto o carrinho se move para a esquerda e para a direita, sem exceder os

limites de posição do carro e ângulo dos mastros impostos pelo problema. A versão utilizada é a não-Markoviana, isto é, existem apenas três estados observáveis: a posição angular de cada um dos mastros e a posição do carro. O carro, ao longo da evolução, pode ser gerado em diversas posições iniciais, sendo tarefa do algoritmo e do robô lidar com isso da melhor forma possível, sem deixar que os mastros caiam.

O *double-pole* contém uma rede neural composta por três neurônios sensores, dez neurônios internos e um neurônio motor, os quais controlarão o agente ao longo da evolução. Os neurônios sensores são responsáveis por manipular a posição do carro e a posição angular dos mastros, já os motores controlam a força aplicada ao carro. Os pesos dessa rede neural são otimizados pelos algoritmos, atuando como os parâmetros.

O ambiente está disponível no evorobotpy2 [Nolfi 2020] e está descrito em mais detalhes em [Carvalho 2017].

3.3. SSS

O *Steady-State Stochastic* (SSS) é um algoritmo evolucionário (μ+1) bem conhecido na literatura pelo seu desempenho superior em relação a outros métodos, na área de neuroevolução [Pagliuca et al. 2018]. Seu funcionamento consiste que, a cada geração, cada parente gerará um descendente, ambos serão avaliados e, os melhores indivíduos da avaliação serão os novos parentes, e assim sucessivamente [Pagliuca et al. 2018]. Além disso, sua principal diferença com relação ao OpenAI-ES é que o SSS utiliza um único conjunto de ambientes para evoluir toda a população de agentes.

A implementação do SSS utilizada neste trabalho e disponibilizada no evorobotpy2 [Nolfi 2020], funciona gerando um genótipo aleatório para a população inicial e em seguida a avalia no problema escolhido sob uma *seed* gerada aleatoriamente. A *fitness* é armazenada a cada geração e os descendentes são avaliados, sendo que os melhores indivíduos se tornam os parentes da próxima geração. Esse ciclo termina quando o limite de *steps*, chamado no simulador de *maxsteps*, é atingido.

3.4. OpenAI-ES

O algoritmo baseado em ES desenvolvido e apresentado em [Salimans et al. 2017], o qual chamamos de OpenAI-ES, opera como um método de otimização de *black-box*, o que significa que algoritmo não tem acesso direto ou conhecimento sobre a estrutura interna da função de otimização [Salimans et al. 2017]. A principal diferença do algoritmo OpenAI-ES de um ES clássico é que o OpenAI-ES utiliza a técnica de otimização estocástica de gradiente Adam [Kingma and Ba 2014] para perturbar os parâmetros e mover seus *centroides*, que são conjuntos de soluções candidatas, numa direção que favoreça a evolução.

A implementação do OpenAI-ES que utilizamos e está disponível no evorobotpy2 [Nolfi 2020], consiste num algoritmo que cria uma matriz de números aleatórios para constituir a população. Os descendentes são duas cópias dessa população, uma que foi perturbada com a adição e outra com a subtração de um vetor de números aleatórios [Nolfi 2021]. Os indivíduos são avaliados e sua *fitness* é organizada em ordem ascendente. Os melhores descendentes são então pós-avaliados, sendo que o melhor indivíduo da avaliação e o da pós-avaliação são armazenados. O gradiente é calculado e então Adam é utilizado para otimizar e atualizar os centroides. Esse método termina ao atingir o limite do estabelecido.

3.5. Algoritmos baseados em NE

Algorithm 1 Niche Evolution

Aleatoriamente gere N nichos
Gere N centroides
enquanto steps <= MAX_STEPS **faça**
  para nGens iterações **faça**
    para n=1 **até** n=N **faça**
      $Avalie\ n^{th}\ centroide\ em\ nicho\ n$
      $Otimize\ n^{th}\ centroide$
    fim para
  fim para
  $fitMatrix \leftarrow zeros(N, N)$
  para n=1 **até** n=N **faça**
    $cn \leftarrow n^{th}\ centroide$
    para m=1 **até** m=N **faça**
      se nicho n <> nicho m **então**
        $Avalie\ cn\ em\ nicho\ m$
        $fitMatrix_{n,m} \leftarrow fitness\ cn\ em\ nicho\ m$
      se não
        $fitMatrix_{n,m} \leftarrow 0$
      fim se
    fim para
  fim para
  para m=1 **até** m=N **faça**
    $maxFit \leftarrow max(fitMatrix_{all,m})$
    $j \leftarrow index\ de\ max(fitMatrix_{all,m})$
    $i \leftarrow m^{th}\ centroide$
    se maxFit >= fitness i **então**
      $o \leftarrow j^{th}\ centroide$
      $Troque\ i\ com\ o\ em\ nicho\ m\ numa\ matriz\ temporária$
      $fitMatrix_{all,j} \leftarrow 0$
      $fitMatrix_{m,all} \leftarrow 0$
    fim se
  fim para
  $Centroides\ são\ efetivamente\ trocados$
fim enquanto

Como o mecanismo do algoritmo OpenAI-ES é essencialmente diferente do algoritmo SSS, projetamos uma série de experimentos para a reprodução da técnica de nicho apresentada em [Carvalho 2017] e da apresentada em [Bianchini et al. 2023a]. Primeiro, desenvolvemos a versão de nicho do SSS, que chamamos de *Steady-State Stochastic - Niche Evolution* (SSS-NE), e do OpenAI-ES e sua contraparte OpenAI-ES-NE, com implementação disponível em [Bianchini et al. 2023b], a qual adaptamos.

A implementação dos nichos consiste em uma matriz com número de colunas igual ao número de nichos e o número de linhas igual ao número de tentativas, definidas

na *policy*. Um nicho, então, é um elemento dessa matriz, o qual conterá um conjunto de número aleatórios, que representam uma *seed* e que consistem no conjunto de condições iniciais para seis variáveis de estado do double-pole. A cada iteração dos algoritmos, *seed* será resetada para todos os nichos.

Para fazer a implementação do mecanismo baseado em *Niche Evolution* nos algoritmos acima, foi utilizado o algoritmo genérico apresentado em [Bianchini et al. 2023a] com algumas modificações.

O mecanismo ficou da forma como é descrita no Algoritmo 1, em que nas duas primeiras linhas são gerados N nichos e seus respectivos centroides. Em seguida, até que o limite de passos seja atingido, o processo de evolução irá continuar, avaliando cada centroide de um nicho em si mesmo e o otimizando, linhas 4 a 9, para em seguida serem avaliados em outros nichos, linhas 11 a 21. Nas linhas seguintes, 22 a 32 é feita a comparação, caso o centroide tenha tido uma melhor performance num nicho diferente do seu, sendo melhor mesmo que o centroide original, o melhor centroide substituirá o original no nicho, utilizando uma matriz temporária, uma adição dessa pesquisa, e ao fim da avaliação efetivamente trocando os centroides. As matrizes de *fitness* são resetadas e o processo continua até que o limite de *steps* seja atingido.

Os experimentos rodaram em 10 bilhões de *steps* que, como explicado anteriormente, consistem em ciclos de avaliação dentro do algoritmo. Além disso, os experimentos foram replicados em 10 *seeds* aleatórias para cada um dos algoritmos, ou seja, uma condição inicial aleatória para a geração dos números aleatórios dentro da evolução. Além disso, o código do simulador e os experimentos dos resultados podem ser acessados em [Machado et al. 2023].

3.6. Irace

Da mesma forma que [Bianchini et al. 2023a], os hiperparâmetros de cada método testado foram otimizados utilizando a biblioteca [López-Ibáñez et al. 2016]. Os métodos desta biblioteca permitem testar iterativamente valores para os hiperparâmetros e avaliar a performance do algoritmo com cada um, a fim de encontrar o conjunto de valores que gera os melhores resultados. A otimização utilizou 1 bilhão de *steps*, 10 % do custo computacional dos experimentos dos resultados deste artigo, em razão de limites computacionais e de tempo. Os hiperparâmetros avaliados foram o número de episódios de avaliação e tamanho da população, no caso do OpenAI-ES e do SSS, e além desses o número de nichos e o número de gerações antes da colonização de outros nichos, no caso dos algoritmos baseados em NE, todos foram avaliados com um conjunto de valores variando de 1 a 100, à exceção da população, variando de 1 a 300, e do número de nichos, de 2 a 10.

Todos os algoritmos dessa pesquisa tiveram seus hiperparâmtros otimizados. Os resultados estão dispostos na Tabela 1. É importante explicitar que apesar da diferença dos parâmetros entre as versões com e sem nicho, o custo computacional é o mesmo para todos os algoritmos, expresso na forma do número de *steps*.

3.7. Teste de pós-avaliação

Para fazer a comparação dos desempenhos dos algoritmos deste trabalho, um teste de pós-avaliação foi feito. Esse teste consiste em pegar os 10 melhores indivíduos de cada

Tabela 1. O melhor conjunto de hiperparâmetros encontrados pela biblioteca de otimização irace no ambiente *double-pole* com custo computacional de 1 bilhão de *steps*.

Hyperparameter	OpenAI-ES	SSS	OpenAI-ES-NE	SSS-NE
Tamanho da população	101	38	30	280
Número de episódios	2	9	6	4
Número de nichos	N/A	N/A	10	7
Gerações até migração	N/A	N/A	98	71

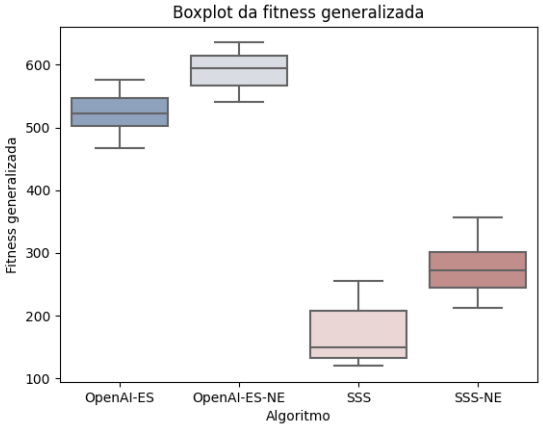

Figura 1. *Boxplot* da *fitness* generalizada do desempenho dos algoritmos OpenAI-ES, OpenAI-ES-NE, SSS e SSS-NE para 1000 condições iniciais na pós-avaliação dos 10 melhores indivíduos evoluídos.

algoritmo ao fim da evolução em 10 bilhões de *steps* para o problema do *double-pole*, para 10 *seeds* aleatórias. Em seguida, esses indivíduos serão pós-avaliados em 1000 condições ambientais aleatórias, também no problema do *double-pole*, armazenando-se a *fitness* obtida para cada uma delas.

Após o armazenamento dos resultados, o *boxplot* é gerado fazendo-se a média dos 1000 valores obtidos para cada um dos 10 indivíduos e comparando-os com o dos outros algoritmos.

4. Resultados Parciais

Os resultados apresentados no *boxplot* de *fitness* generalizada da Figura 1 dizem respeito à reprodução da proposta em [Carvalho 2017], com os algoritmos SSS e SSS-NE, e à reprodução de [Bianchini et al. 2023a], com os algoritmos OpenAI-ES e OpenAI-ES-NE, dentro do teste da pós-avaliação com 1000 condições iniciais aleatórias para os 10 melhores indivíduos de cada algoritmo.

O desempenho superior dos algoritmos NE em relação à suas contrapartes pode ser observado com a média de valores de *fitness* obtidas pelos algoritmos baseados em NE sendo superiores às suas contrapartes sem nicho. No caso do OpenAI-ES-NE, a *fitness* se concentra nos valores entre 570 e 620, com dispersão baixa em torno da média, como pode ser observada pelo tamanho diminuto do comprimento das hastes e da caixa na Figura 1, com cerca de 40 unidades de intervalo entre o quartil superior e o inferior. Além

disso, pode-se inferir pelo *boxplot* do OpenAI-ES, Figura 1, que seus valores de *fitness* se encontram, na maioria, entre aproximadamente 500 e 550, com dispersão e intervalo interquartílico, cerca de 50 unidades, ligeiramente maior do que sua contraparte, o que indica um aumento de performance de aproximadamente 8,6% e dados ligeiramente mais consistentes para sua contraparte em NE.

O desempenho superior do SSS-NE em relação à sua contraparte SSS, pode ser atestado observando o *boxplot* de ambos na Figura 1. Os valores de *fitness* do SSS se encontram principalmente no intervalo 130 a 210, com cerca de 80 unidades de intervalo interquartílico, enquanto no SSS-NE a *fitness* se encontra na maioria entre 250 e 300, com intervalo de cerca de 50 unidades, o que indica um aumento de performance de aproximadamente 53,5% e uma dispersão significativamente menor que sua contraparte. Importante ressaltar que esses resultados são mais baixos do que os obtidos por [Carvalho 2017], em que os algoritmos tiveram performances superiores, entretanto, deve-se considerar que os algoritmos de [Carvalho 2017] não performaram no simulador evorobotpy2 [Nolfi 2020]. Além de que a diferença entre as contrapartes SSS e SSS-NE se mantém consistentes com o obtido por [Carvalho 2017].

Esses resultados indicam que a adição técnicas de nicho efetivamente aumentaram a performance dos algoritmos em que foram aplicadas, especialmente para o SSS-NE, que obteve um aumento de mais de 50% em comparação com seu original. Além disso, demonstrou-se a relevância do método para o algoritmo OpenAI-ES, o que pode indicar que o método pode ser favorável a outros algoritmos baseados em ES.

5. Considerações Finais e Trabalhos Futuros

Nesse trabalho buscou-se trazer um breve panorama da relevância do uso dos mecanismos de nichos no campo da Robótica Adaptativa, assim como apresentar pesquisas que obtiveram bons resultados por meio da aplicação dos mecanismos de nicho. Dessa forma, fizemos uma comparação dos resultados obtidos em relação à essa investigação, apresentados em [Carvalho 2017] e [Bianchini et al. 2023a], realizando uma análise da performance dos algoritmos SSS, SSS-NE, OpenAI-ES e OpenAI-ES-NE no ambiente *double-pole*.

Os resultados indicaram a melhora de performance dos algoritmos SSS e OpenAI-ES sob técnicas de nicho, o que demonstra a relevância do emprego desse mecanismo na evolução do comportamento robótico, gerando soluções mais gerais e mais eficientes do que aquelas encontradas em ambientes estáticos. Assim, tais resultados indicam que pode ser promissor aplicar esses mecanismos em outros algoritmos baseados em Estratégias de Evolução como, por exemplo, o CMA-ES e o xNES, [Pagliuca et al. 2020].

Dessa forma, para trabalhos futuros, o objetivo é investigar o mecanismo de *Niche Evolution* em outros algoritmos baseados em ES, avaliando suas performances de forma mais aprofundada, por exemplo, medindo a diversidade das soluções obtidas e também verificando a movimentação dos indivíduos entre nichos por meio do mecanismo de colonização. Além disso, pretendemos investigar o desempenho dos algoritmos usando nichos em outros ambientes e tarefas disponíveis no simulador evorobotpy2 [Nolfi 2020].

Referências

(2019). *The impact of environmental history on evolved robot properties*, volume ALIFE 2019: The 2019 Conference on Artificial Life of *ALIFE 2021: The 2021 Conference*

Bianchini, A. H., Machado, B. S., and Carvalho, J. T. (2023a). The effectiveness of niching on openai-evolution strategies in the evolution of robotic behavior. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 2354–2357, New York, NY, USA. Association for Computing Machinery.

Bianchini, A. H., Nolfi, S., and Machado, B. S. (2023b). A stripped-down version of evorobotpy2 with openai-es-ne and some experiments results. `https://github.com/Brenda-Machado/stripped-down-version-of-evorobotpy2/`.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Carvalho, Jonata Tyska e Nolfi, S. (2017). Favoring the evolution of adaptive robots through environmental differentiation. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7.

Chowdhury, A., Karmakar, G., Kamruzzaman, J., Jolfaei, A., and Das, R. (2020). Attacks on self-driving cars and their countermeasures: A survey. *IEEE Access*, 8:207308–207342.

Ekárt, A. and Németh, S. Z. (2002). Maintaining the diversity of genetic programs. In *European Conference on Genetic Programming*, pages 162–171. Springer.

Fogel, D. B. (1997). The advantages of evolutionary computation. In *Bcec*, pages 1–11.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Linder, M. H. and Nye, B. (2010). Fitness, environment and input: Evolved robotic shepherding. *Dept. Comput. Sci., Swarthmore College, Swarthmore, PA, USA, Tech. Rep*.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Machado, B. S., Nolfi, S., and Bianchini, A. (2023). Evorobotpy2 with algorithms based in niche evolution. `https://github.com/Brenda-Machado/evorobotpy2-with-ne/`.

Milano, N., Carvalho, J. T., and Nolfi, S. (2017). Environmental variations promotes adaptation in artificial evolution. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7.

Nolfi, S. (2020). A tool for training robots through evolutionary and reinforcement learning methods. https://github.com/snolfi/evorobotpy2.

Nolfi, S. (2021). Behavioral and cognitive robotics: An adaptive perspective. Roma, Italy: Institute of Cognitive Sciences and Technologies, National Research Council (CNR-ISTC).

Pagliuca, P., Milano, N., and Nolfi, S. (2018). Maximizing adaptive power in neuroevolution. *PloS one*, 13(7):e0198788.

Pagliuca, P., Milano, N., and Nolfi, S. (2020). Efficacy of modern neuro-evolutionary strategies for continuous control optimization. *Frontiers in Robotics and AI*, 7.

Rechenberg, I. (1973). Evolutionsstrategie. *Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution*.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. arXiv.

Sareni, B. and Krahenbuhl, L. (1998). Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106.

Whitley, D., Rana, S., and Heckendorn, R. (1998). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7.