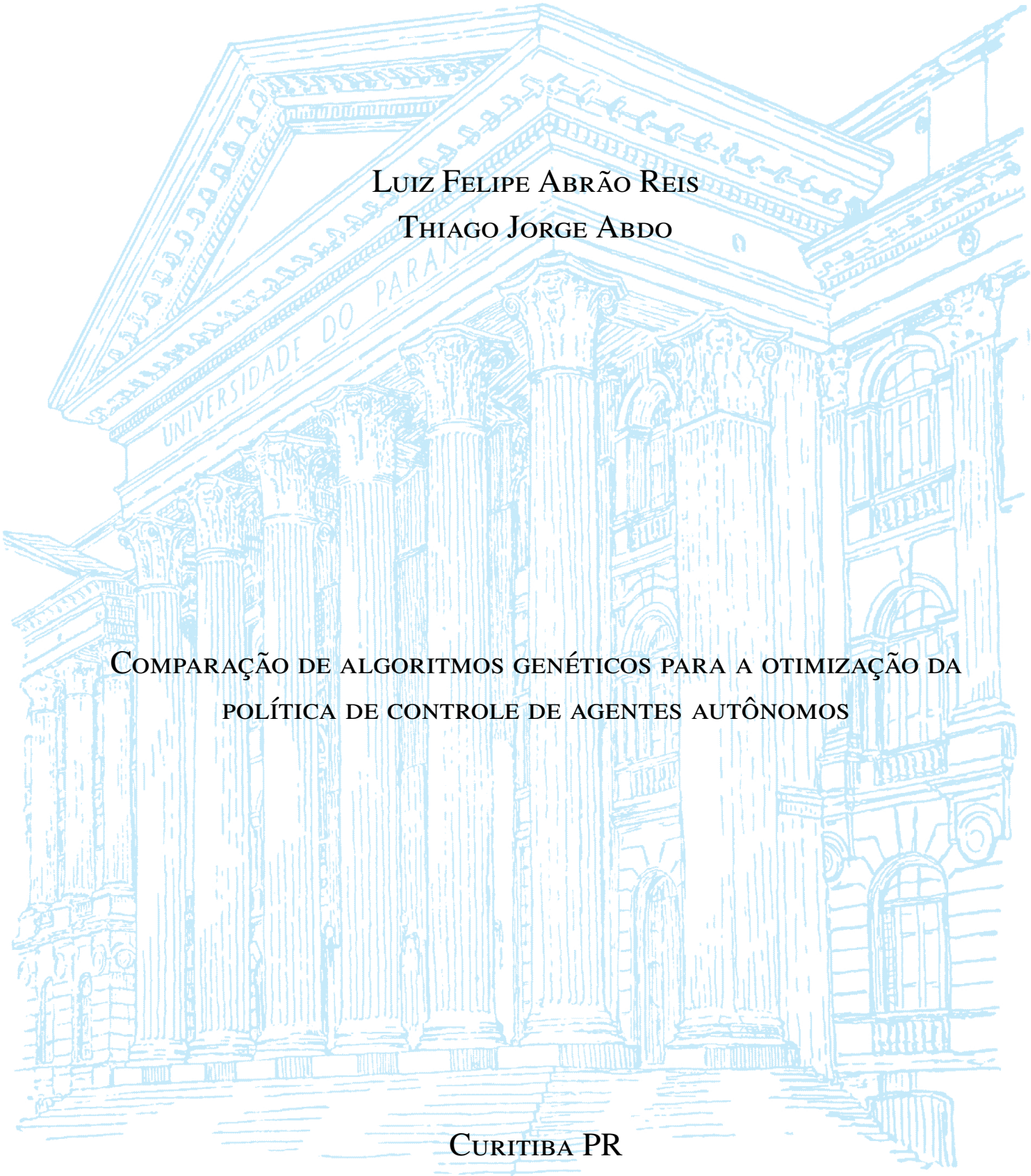
Universidade Federal do Paraná

Luiz Felipe Abrão Reis
Thiago Jorge Abdo



Comparação de algoritmos genéticos para a otimização da política de controle de agentes autônomos

Curitiba PR
2018

Luiz Felipe Abrão Reis
Thiago Jorge Abdo

Comparação de algoritmos genéticos para a otimização da política de controle de agentes autônomos

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Jaques Spinosa.

Curitiba PR
2018

Agradecimentos

Agradecemos de forma especial à nossa família, por não medirem esforços para que pudéssemos levar nossos estudos adiante.

Agradecemos, também, a nosso orientador, Eduardo Spinosa, pela paciência, dedicação e ensinamentos que possibilitaram que realizássemos este trabalho.

Agradecemos a esta instituição pelo excelente ambiente oferecido aos seus alunos e os professores qualificados que disponibiliza para nos ensinar.

Por fim, agradecemos aos nossos amigos, especialmente à Alice, ao Pedro e a nossos amigos do forró, por confiarem em nós e estarem do nosso lado em todos os momentos da vida.

Resumo

Por meio deste trabalho, é exposta uma comparação entre um algoritmo genético padrão e o *Neuro-evolution of augmenting topologies (NEAT)*, usando *fitness* baseada em objetivo e em novidades, na busca por uma política de controle satisfatória para carros autônomos em um espaço Cartesiano de duas dimensões. Os testes consistiram na avaliação da distância percorrida pelos carros em 200 *timesteps* de simulação, tempo suficiente para dar uma volta nos circuitos definidos. Através dos resultados, que mostraram uma leve vantagem da busca por novidades sobre a busca por objetivo, no caso do NEAT, foi percebido que a exploração por novos comportamentos pode ter influência direta na qualidade das redes geradas por este algoritmo, trazendo benefícios reais para problemas nos quais escolher a topologia da rede manualmente seja difícil.

Palavras-chave: algoritmo evolutivo, controle de agente, inteligência artificial.

Abstract

In this work, a comparison between a standard genetic algorithm and *Neuro-evolution of augmenting topologies (NEAT)* is shown, through the use of objective and novelty based fitness, in the pursuit of a satisfactory action selection policy for autonomous cars in an Cartesian bidimensional space. Tests consisted in the evaluation of the car's travelled distance after 200 timesteps of simulation, enough time for the cars to complete a lap in the defined circuits. The results obtained, which revealed a slight advantage of the Novelty Search over objective based fitness, when using NEAT, led to the conclusion that the search for new behaviors could have a direct influence on the quality of the neural networks generated by this algorithm, benefiting problems in which it is difficult to choose the network topology manually.

Keywords: evolutionary algorithm, agent control, artificial intelligence.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

AG	Algoritmo Genético
MLP	Multilayer perceptron
NEAT	Neuro-evolution of Augmenting Topologies (Neuroevolução de topologias aumentantes)
NS	Novelty Search (Busca por novidade)

Lista de Símbolos

$\sum$	Simbolo para representar somatório
σ	Sigma minusculo, décima oitava letra do alfabeto grego

Sumário

Capítulo 1

Introdução

Com o avanço de pesquisas na área de inteligência artificial e o aumento do poder computacional, agentes autônomos para problemas complexos se tornaram possíveis e passaram a ser muito discutidos por aprenderem um comportamento sem intervenção humana. Isso implica na criação de agentes sem um conhecimento prévio do domínio do problema. Diversas abordagens, como algoritmos genéticos e *reinforcement learning*, podem ser utilizados para solucionar o problema do aprendizado de uma política de seleção de ações de agentes autônomos, que dada uma leitura do ambiente decide qual a melhor ação a ser tomada para que o agente consiga realizar seus objetivos.

O objetivo deste trabalho é comparar diferentes abordagens de algoritmos genéticos para a otimização da política de controle de agentes autônomos e identificar a mais eficiente para o problema definido: agentes controladores de veículos. Para isto, foi desenvolvido um simulador em duas dimensões em que carros andam em uma pista por um período de tempo fixo e tem como objetivo percorrer o maior trajeto possível sem colidir com obstáculos ou sair da pista.

Os testes executados comparam o algoritmo NEAT com um algoritmo genético padrão, além do impacto da *Novelty Search* nos resultados desses algoritmos. É importante ressaltar que tais testes e seus resultados podem ser expandidos para problemas mais complexos e ambientes menos previsíveis e que através deles buscamos confirmar as seguintes hipóteses:

- O uso de um algoritmo genético padrão, com uma rede de controle bem definida, traz resultados superiores aos de uma rede evoluída pelo NEAT, em uma quantidade limitada de iterações
- A busca por novidades influencia na qualidade das redes de controle evoluídas pelo NEAT
- A busca por novidades não é superior a uma busca por objetivos quando o problema não possui caráter enganador ou ótimos locais

Este trabalho está dividido em 6 capítulos. O capítulo 2 define conceitos básicos de algoritmos bio-inspirados, como redes neurais e algoritmos genéticos. O capítulo 3 dá uma visão geral de como esses algoritmos são aplicados no problema do controle de agentes autônomos. No capítulo 4 apresentamos nosso ambiente de simulação, para no capítulo 5, descrevermos nossos experimentos e apresentarmos os resultados obtidos. No capítulo 6 concluímos com as considerações finais e trabalhos futuros.

Capítulo 2

Algoritmos bio-inspirados

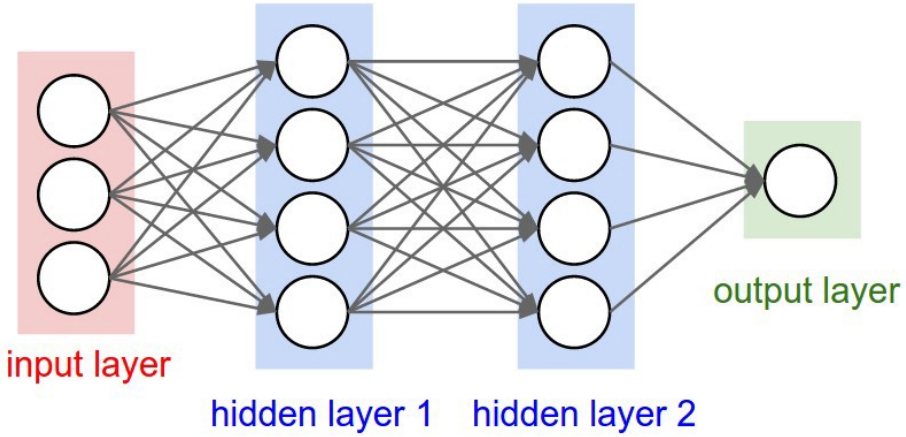
2.1 Redes Neurais

Redes neurais artificiais são sistemas inspirados em sistemas nervosos de animais capazes de aprendizado e do ponto de vista matemático, são aproximadores universais (Csáji, 2001). Uma rede neural artificial é composta por diversas unidades de processamento simples, chamadas de neurônios, que possuem sinais de saída e recebem um ou mais sinais de entrada.

2.1.1 Multi-layer perceptron

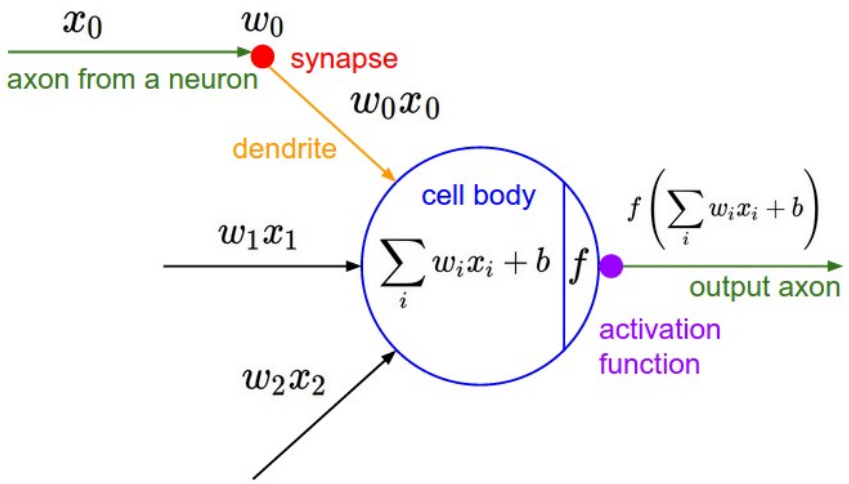
Um Multi-Layer perceptron (MLP) é um tipo de rede neural, composto por séries de neurônios em camadas. Cada neurônio recebe as saídas dos neurônios da camada anterior. A primeira e a última camada são as camadas de entrada e saída, respectivamente; as demais camadas são consideradas camadas intermediárias. Enquanto um MLP de uma camada intermediária é capaz de aproximar qualquer função contínua, um de duas camadas pode aproximar qualquer função (Cybenko, 1989). Adicionalmente, cada camada possui um neurônio adicional, cuja entrada é sempre 1. Seu nome é *bias* e tem a finalidade de deslocar a função de ativação.

Figura 2.1: Ilustração de uma rede neural. Retirada de Karpathy (2015)



A saída Y de um neurônio é determinada pela aplicação de uma função de ativação f sobre a combinação linear, ponderada por W_i, de todas as suas n entradas $(X_1, X_2, ..., X_n)$.

Figura 2.2: Neurônio de uma rede neural. Retirada de Karpathy (2015)



Várias funções podem ser utilizadas como funções de ativação. Entre as mais comuns se encontram a função sigmóide (2.1) tanh (2.2).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$tanh(x) = 2 \cdot \sigma(2x) - 1 \tag{2.2}$$

2.2 Computação evolutiva

Computação evolutiva é um ramo da inteligência computacional que consiste de algoritmos de otimização global. Estes algoritmos são inspirados em fenômenos naturais, tais como a evolução e, a partir de uma população inicial, melhoram a solução iterativamente.

O processo de otimização desses algoritmos é estocástico e baseado em meta-heurísticas e, geralmente, acontece sobre uma função chamada *fitness*, que quantifica o quão boa é uma dada solução para o problema. (Michalewicz, 1996)

2.2.1 Algoritmo Genético

Algoritmos genéticos (AG) são uma família de modelos computacionais evolutivos baseados na teoria da evolução. Estes algoritmos codificam potenciais soluções para um problema em estruturas de dados similares a cromossomos e aplicam operadores de recombinação de maneira a preservar informações críticas. É comum que algoritmos genéticos sejam vistos como otimizadores de funções, mas suas aplicações podem ir além disso (Whitley, 1994).

Uma implementação genérica de um algoritmo genético se inicia com uma população de indivíduos aleatórios, que em seguida são avaliados quanto à qualidade da solução que ele representa para o problema. Então, ocorre um processo de reprodução em que os indivíduos com as melhores avaliações possuem mais chance de propagarem seus genes. A partir deste processo de reprodução, obtém-se os descendentes, que constituirão a próxima população dentro do processo evolutivo.

A seguir, observa-se um algoritmo genético padrão:

Dentre os operadores de recombinação do código genético, se destacam:

Algoritmo 1 Pseudocódigo de um Algoritmo Genético

```
1: P ← população inicial aleatória
2: for g ← 1 to número de gerações do
3:     avalia(P)
4:     S ← seleção(P)
5:     P ← ∅
6:     for all par(i1, i2) in S do
7:         O ← cruzamento(i1, i2)
8:         P ← P ∪ mutação(O)
9:     end for
10: end for
```

- Seleção: escolhe indivíduos para reprodução baseado em suas avaliações.
- Cruzamento: gera novos indivíduos a partir do cruzamento de dois indivíduos selecionados. Geralmente é utilizada uma taxa que define a probabilidade de ocorrer a troca de genes entre os dois indivíduos geradores em seus descendentes.
- Mutação: troca aleatória de parte do código genético de um indivíduo, cuja probabilidade é definida pela taxa de mutação.

2.2.2 *Neuro-evolution of augmenting topologies*

Quando os algoritmos genéticos canônicos são utilizados para otimizar os pesos das conexões de uma rede neural, deve-se definir a topologia da rede previamente ao processo de treino. Isso faz com que um processo de tentativa e erro seja necessário para determinar uma topologia adequada.

O algoritmo *Neuro-evolution of augmenting topologies* (NEAT) complementa esses algoritmos genéticos otimizando o modelo da rede neural e seus pesos simultaneamente.

O NEAT começa com uma rede MLP com apenas neurônios de entrada e de saída. Conforme o processo de evolução progride, novos neurônios e conexões podem ser adicionados a essa rede, aumentando o conjunto de funções que a rede neural desse indivíduo é capaz de aproximar.

Essas mudanças na topologia da rede acontecem durante a fase de cruzamento e a de mutação. No caso de um cruzamento, os indivíduos geradores podem ter diferentes topologias, assim, seus sucessores devem refletir uma mistura das redes neurais de seus pais. No caso de uma mutação, novas conexões são criadas entre dois neurônios desconexos ou uma conexão existente é quebrada em duas, para dar lugar a um novo neurônio.

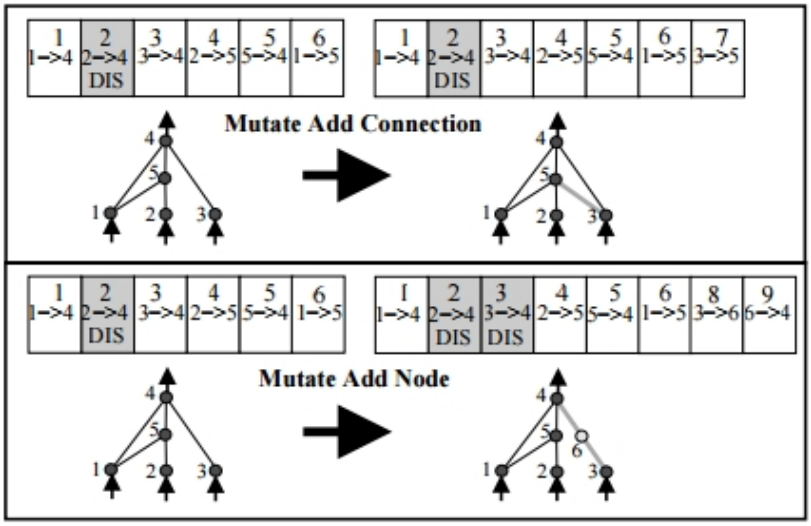


Figura 2.3: Exemplo de mutação da topologia de um indivíduo. Retirada de Stanley e Miikkulainen (2002)

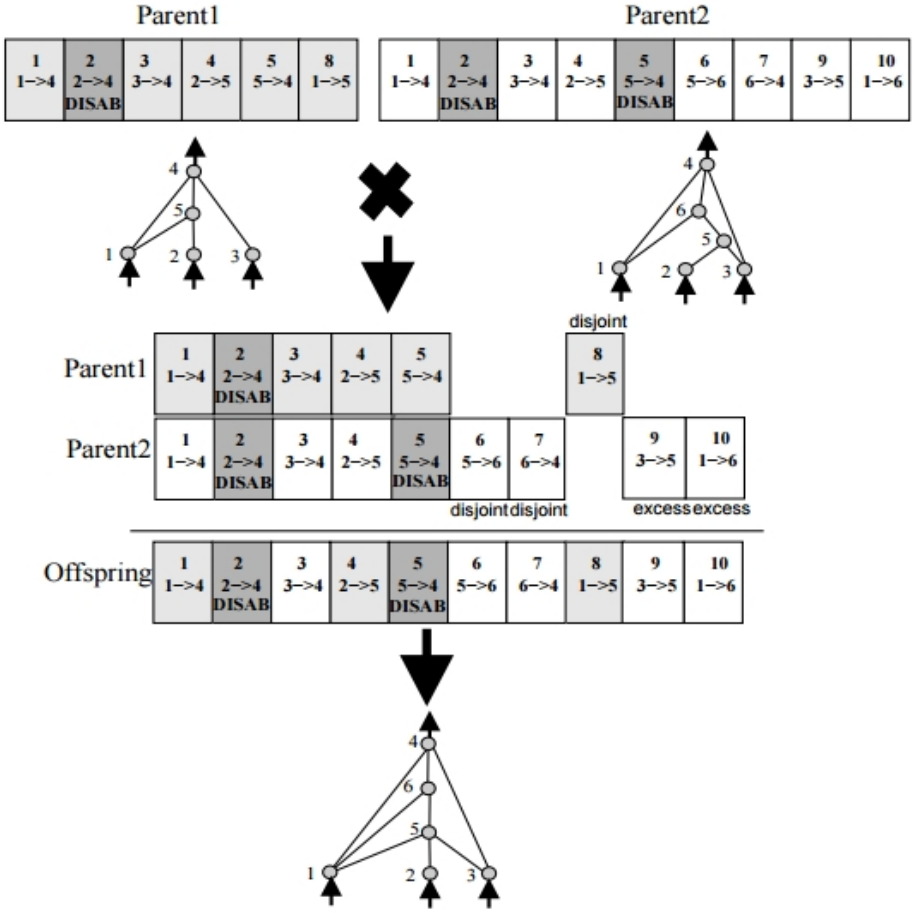


Figura 2.4: Exemplo de cruzamento entre dois indivíduos com topologias diferentes. Retirada de Stanley e Miikkulainen (2002)

2.2.3 *Novelty Search*

Funções objetivas - nas quais a recompensa é maior quanto mais perto do objetivo - são suscetíveis a enganos e ótimos locais visto que não necessariamente recompensam os passos intermediários no espaço de busca que ultimamente levarão ao objetivo. O algoritmo de *Novelty Search* (NS) foi desenvolvido para minimizar os impactos desses possíveis problemas na busca e recompensar os passos intermediários (Lehman e Stanley, 2008).

Segundo Lehman e Stanley (2008), poucas alterações são necessárias para incorporar a busca por novidades em um algoritmo evolutivo. Na verdade, a única alteração necessária para incentivar novos comportamentos é trocar a função de fitness por uma *métrica de novidade*. Esta métrica deve medir o quão diferente um indivíduo é dos outros e cria uma pressão para que a busca encontre novos comportamentos. Com isso, ao invés de recompensar o desempenho relativo a um objetivo, a busca por novidades recompensa a divergência de um novo comportamento em relação aos comportamentos já explorados.

Existem diversas maneiras de medir novidade através da análise e quantificação de comportamentos para caracterizar suas diferenças, e tais maneiras devem ser compatíveis com o domínio do problema em questão. É importante que o cálculo da novidade seja feito com base no comportamento de um indivíduo, e não em seu genótipo, visto que os dois não necessariamente estão correlacionados.

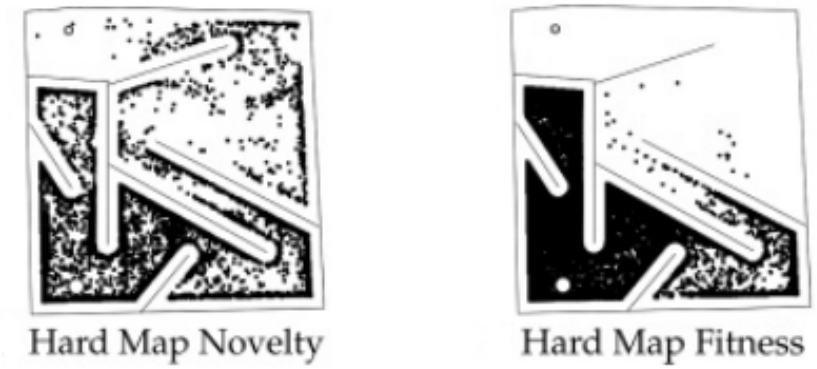


Figura 2.5: Concentração dos indivíduos em um labirinto - Novidade vs. Objetivo. Adaptada de Lehman e Stanley (2011)

Capítulo 3

Controle de agentes autônomos

3.1 Agentes autônomos

Segundo Maes (1995), agentes autônomos são sistemas computacionais que residem em um ambiente complexo e dinâmico, percebem este ambiente e agem autonomamente baseados em seus sentidos e, assim, executam um conjunto de objetivos e tarefas paras as quais foram projetados.

3.2 Políticas de seleção de ações

Agentes autônomos percebem o ambiente através de sensores e, baseados neles, devem escolher uma ação que os ajude a alcançarem seus objetivos. A política de seleção de ações (ou política de controle) de um agente é o que seleciona qual a melhor ação baseada no conhecimento do agente e em sua percepção do ambiente. Diferentemente do controle do agente, a política de controle não sabe como aplicar uma ação, seu único interesse é escolher uma ação apropriada, que contribua com os objetivos do agente, para que então, o controle do agente possa efetuá-la.

Teoricamente, em ambientes determinísticos ou probabilísticos, é possível calcular uma política de seleção de ações para um agente, que possua objetivos definidos, que seja ótima (Tyrrell, 1993). Porém, a maior parte dos agentes reais sofre de problemas que tornam isso impossível, como:

- limitações de recursos (tempo, computação e memória)
- informações e percepções incompletas ou incorretas do ambiente através dos sensores
- ambientes dinâmicos, não determinísticos ou não probabilísticos
- objetivos variáveis

Assim, muitas pesquisas na área de agentes autônomos não estão tão interessadas em provar que a política de seleção de ações de um agente é ótima quanto estão interessadas na robustez e adaptatividade da seleção de ações, bem como na capacidade do agente de atingir seus objetivos dentro das restrições impostas pelo ambiente e tarefa em questão (Maes, 1993).

3.2.1 Redes neurais como políticas de seleção de ações de um agente autônomo

É possível utilizar uma rede neural para servir como a política de seleção de ações de um agente autônomo. Esse mapeamento é feito da seguinte forma: os sensores do agente são a entrada da rede; as outras camadas computam qual ação tomar baseadas nas entradas da rede; a camada de saída indica qual a ação escolhida para determinada combinação dos sensores do agente.

Mapeamento

Diferentes espaços de ações implicam em diferentes mapeamentos. Por exemplo, para um agente que possui uma quantidade finita de ações a escolher, é possível montar a rede neural baseada numa rede de classificação, onde as ações possíveis fazem o papel de classes e os sensores são mapeados de modo a escolher uma probabilidade para cada classe. Neste caso, o número de saídas da rede é igual ao tamanho do conjunto de ações possíveis e a classe com a maior probabilidade representa a ação escolhida pela política de controle.

Em casos onde o espaço de ação é contínuo, é possível mapear as saídas para o intervalo desejado através das funções de ativação: funções como a sigmóide (2.1) e tanh (2.2) são contínuas dentro de um intervalo ($[0..1]$ e $[-1..1]$ respectivamente). Assim, basta apenas mapear a função de ativação de saída para o intervalo utilizado pelo agente.

Treinamento da rede

Geralmente, em problemas de otimização da seleção de ações de um agente autônomo, não existem exemplos de entradas com a saída esperada, o que torna impossível o treinamento da rede através de métodos padrões como a *backpropagation* (LeCun et al., 1988). Para contornar esse problema, é possível utilizar algoritmos de aprendizado por reforço ou algoritmos de otimização, aliados a alguma métrica de desempenho da política atual da rede. Essas abordagens são baseadas em tentativa e erro e exploram o espaço das possíveis soluções, tentando encontrar soluções cada vez melhores medidas pela função de *fitness*.

Capítulo 4

Ambiente de simulação

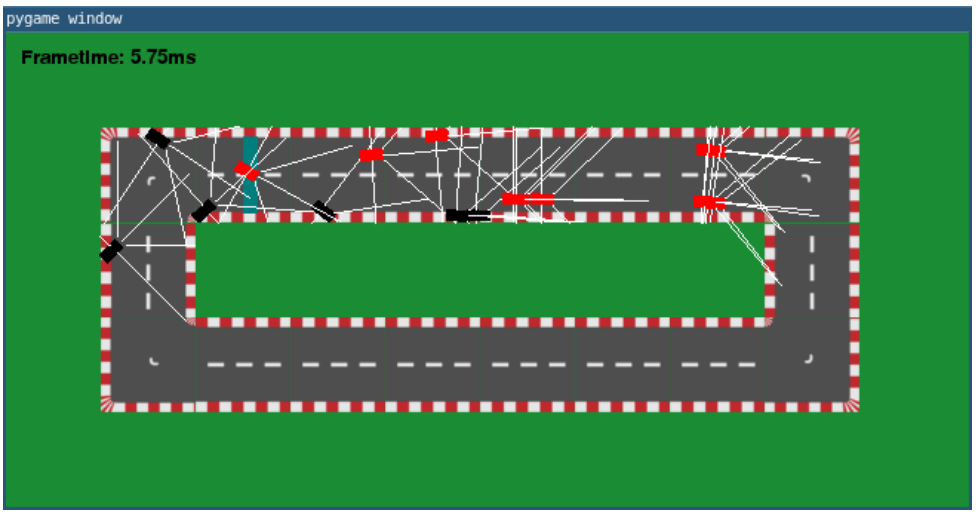
Para avaliar o comportamento das diferentes formas de otimização com o uso de algoritmos genéticos, foi necessário um ambiente de testes que simulasse o comportamento de um veículo em um circuito fixo. Tal ambiente se deu por meio de um simulador mais básico do que os encontrados na literatura, desenvolvido especificamente para o problema em questão, garantindo um maior controle do ambiente e um processamento mais rápido das simulações.

4.1 Simulador

O simulador utilizado nos testes foi desenvolvido em python e simula apenas duas dimensões. Os carros recebem de seus controladores (redes neurais para a seleção de ações) dois valores entre -1 e 1: o primeiro controla o quanto o volante do carro deve virar e o segundo controla o acelerador/freio.

Os cálculos do simulador não levam em conta fatores da física, como o atrito e a função de distância percorrida contabiliza apenas movimentos na direção da pista, ou seja, reflete a distância da linha de partida até a posição atual do carro.

Figura 4.1: Imagem exemplo do simulador



As simulações rodam 200 *timesteps* de 0,03 segundos, totalizando 6 segundos de simulação, tempo que - devido a velocidade dos carros e tamanho da pista - se provou suficiente para que os carros dessem uma volta completa nos circuitos de teste. O tamanho de um *timestep*

foi definido de modo que a alta velocidade dos carros não introduzisse problemas no cálculo da física e na detecção de obstáculos.

A cada intervalo os controladores dos agentes decidem qual a ação a ser tomada pelo carro. Além disso, embora os cálculos do simulador sejam feitos num espaço cartesiano (e portanto contínuo), os circuitos são definidos baseados em um *grid* (e portanto discretos).

Durante a simulação, os carros estão ativos até colidirem com obstáculos ou saírem da pista. No momento em que um carro é desativado - seja durante ou no fim da simulação -, sua *fitness* é definida como a maior distância que esse carro alcançou a partir da linha de partida.

Na parte gráfica do simulador, a área verde é considerada fora da pista. Carros pretos são carros que já colidiram ou saíram do circuito definido, enquanto carros vermelhos são carros ativos. As linhas brancas que saem de cada carro são projeções de seus sensores de distância.

4.2 Modelagem dos veículos

Todas as redes neurais utilizadas nos testes receberam a mesma entrada e devolveram a mesma saída:

- Como entrada, cada rede recebeu a velocidade atual do carro em relação a pista e 5 sensores de distância (limitados em 64 pontos no espaço cartesiano, ou 1 ponto no *grid* em que o circuito é definido)
- Como saída cada rede calculava o valor de dois atuadores entre $[-1, 1]$, um controlando a aceleração do carro e o outro controlando o volante

Figura 4.2: Imagem de um carro, incluindo a projeção de seus sensores

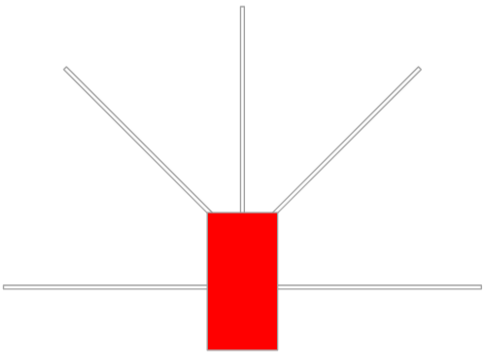
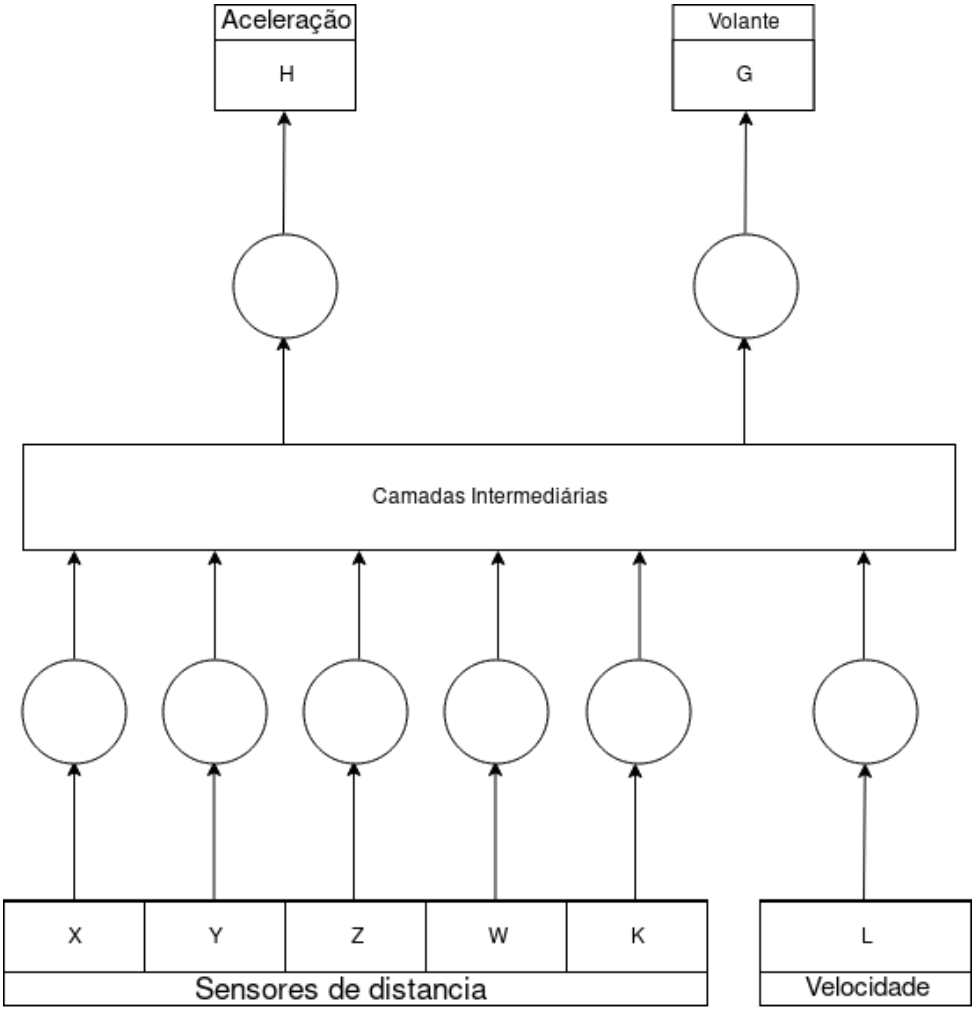


Figura 4.3: Imagem exemplificando o mapeamento dos sensores e atuadores de um carro para a rede neural de seleção de ações



4.3 Funções de fitness

As Funções de fitness variam de acordo com o modo em que a busca é conduzida na árvore de estados durante a otimização. Buscas baseadas em objetivos possuem uma função de fitness que está diretamente conectada à peformance do indivíduo. Por outro lado, buscas baseadas em novidade precisam de um meio de expressar comportamentos do indivíduo, para que as diferenças na complexidade do comportamento do indivíduo possam ser medidas e utilizadas para calcular sua fitness.

4.3.1 Fitness baseada em objetivos

Para buscas orientadas a objetivo, utilizamos como fitness a distância percorrida pelo veículo à partir do ponto de início do circuito. O tempo não foi incluído no cálculo dessa função, visto que com um tempo limitado de vida, indivíduos que viveram o mesmo tempo e possuem uma fitness maior serão mais rápidos.

4.3.2 Fitness baseada em novidades

Fitness baseadas em novidade precisam de uma forma de expressar a complexidade comportamental de um indivíduo. Neste estudo, o comportamento dos indivíduos foi quantificado da seguinte maneira:

O vetor de comportamento, que mapeia o comportamento de um indivíduo para um espaço cartesiano, é montado com as coordenadas X e Y da posição final do carro e com a distância percorrida pelo carro a partir da linha de largada. Assim, o vetor do comportamento de um indivíduo é da forma $[x\_final, y\_final, distância\_percorrida]$ e a diferença entre dois comportamentos pode ser medida através da distância Euclidiana.

Após o cálculo do vetor de comportamento, é aplicado a média da distância entre os k-vizinhos mais próximos para obter, então, o valor de fitness de um indivíduo, onde k é um parâmetro fixo e escolhido experimentalmente. É fácil perceber que quanto maior a distância entre os comportamentos mais próximos, maior é a diferença - e portanto a novidade - de um novo comportamento. O espalhamento p para um comportamento x é dado por

$$p(x) = \frac{1}{k} \sum_{i=0}^{k} dist(x, u_i)$$

onde u_i é o i-ésimo vizinho mais próximo de x levando em conta a medida $dist$, que mede a diferença comportamental entre dois indivíduos no espaço de comportamentos.

Capítulo 5

Experimentos

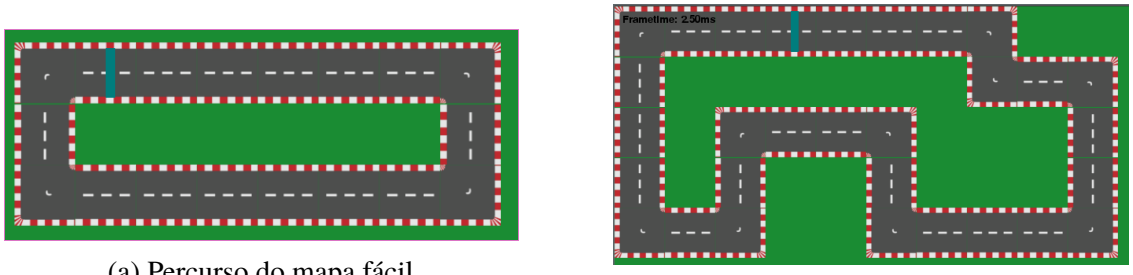
Cada algoritmo e método de busca foi testado diversas vezes em cada mapa e a diferença no número de testes se deu por causa do maior tempo de execução de alguns algoritmos. Para todos os testes foram executadas 25.000 avaliações da *fitness* e os parâmetros foram obtidos de artigos ou otimizados através de testes preliminares.

A velocidade máxima dos carros foi definida em 500 pontos por segundo, onde cada *tile* do mapa equivale a 64 pontos. Essa velocidade se provou suficiente para que os carros dessem uma volta completa na pista durante o tempo da simulação e introduziu algumas dificuldades para os agentes na hora de fazer curvas.

Como a simulação consiste de 200 *timesteps* de 0.03s cada, o tempo máximo de simulação é 6 segundos. Se considerarmos um cenário onde o carro apenas acelera, sem reduzir sua velocidade ou virar, podemos estabelecer um valor máximo teórico para a *fitness* de 3000 pontos.

5.1 Mapas

Para treinar a rede e verificar as diferenças em diferentes mapas, foram criados dois mapas: um mapa fácil contendo apenas retas e curvas para a direita e um mapa misto, com curvas próximas umas das outras e variando entre esquerda ou direita, resultando em um mapa difícil que de certa forma representa quase todas as curvas possíveis em nosso simulador.



(a) Percurso do mapa fácil

(b) Percurso do mapa difícil

Figura 5.1: Mapas utilizados nos testes

5.2 Parâmetros dos testes

5.2.1 Algoritmo genético com *fitness* baseada em objetivos

A biblioteca DEAP (Fortin et al. (2012)) foi utilizada para obter as funções de algoritmos genéticos já implementadas. Com base em testes preliminares, foram escolhidos torneios de tamanho 3 para seleção e chance de cruzamento de 80% e de mutação de 30%. Foram efetuados testes para populações de 250 indivíduos, com 25.000 avaliações da função de *fitness*, resultando em 100 gerações.

A rede neural definida para selecionar a ação do agente é um *multi-layer perceptron* com uma camada intermediária de 7 neurônios. Tal rede se provou suficiente para resolver o problema e foi obtida a partir de testes, onde o número de neurônios na camada intermediária foi aumentado gradualmente até que a rede fosse suficiente para aproximar uma política de controle satisfatória.

5.2.2 NEAT com *fitness* baseada em objetivos

Para os teste de NEAT com fitness objetiva, foi utilizada uma população de tamanho 250, resultando em 100 gerações. Adicionalmente, os seguintes parâmetros foram extraídos e adaptados dos testes com NEAT de Lehman e Stanley (2011).

Parâmetro	**Valor**	**Descrição do parâmetro**
Population	250	Tamanho da população utilizado, impacta no número de indivíduos avaliados simultaneamente e portanto no número de cruzamentos a cada geração
Clones	False	Permite que dois indivíduos sejam exatamente iguais
Loops	False	Caso True, permite que neurônios de uma camada se conectem com neurônios de camadas anteriores. Utilizado como False, para garantir que as redes neurais dos indivíduos tenham estrutura semelhante a um MLP
Mutate Weights	90%	Probabilidade de ocorrer uma mutação nos pesos das conexões da rede de um indivíduo
Add Neuron	7%	Define a probabilidade da adição de um neurônio
Add Link	15%	Define a probabilidade da adição de uma conexão
Remove Link	1%	Define a probabilidade da remoção de uma conexão

Tabela 5.1: Parâmetros usados nos testes do NEAT

Os parâmetros *Mutate Weights*, *Add Neuron*, *Add Link* e *Remove Link* são relativamente altos para incentivar a geração de indivíduos diferentes em todos os estágios da busca.

5.2.3 *Novelty search* utilizando NEAT

Os parâmetros do NEAT com *fitness* baseada em objetivo foram mantidos para os testes de *Novelty Search*. Adicionalmente, os seguintes parâmetros, que foram adaptados de Lehman e Stanley (2011), foram utilizados:

Parâmetro	Valor	Descrição do parâmetro
K-nearest Neighbors	10	Número de vizinhos utilizados para computar novidade. Impacta nos valores do espalhamento de um indivíduo: ao considerar menos indivíduos, os indivíduos mais distantes não participam do cálculo
Recompute sparseness	20	Recalcula o espalhamento para todos os indivíduos após N avaliações
Dynamic Pmin	True	Ajusta o limite para adicionar um indivíduo ao arquivo de indivíduos durante a execução, baseado na novidade dos indivíduos recentemente avaliados

Tabela 5.2: Parâmetros do Novelty Search utilizados nos testes

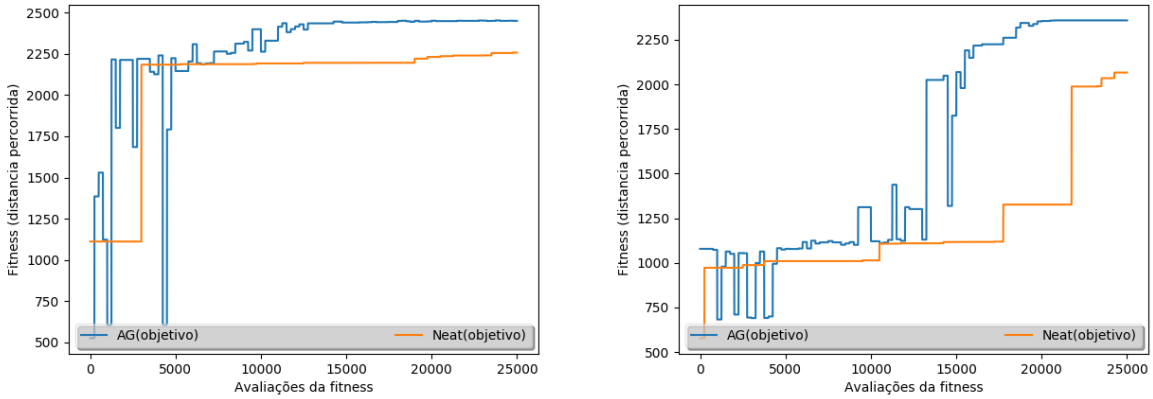
A implementação do NEAT com Novelty Search avalia um indivíduo por vez. Tal decisão foi tomada com base em Lehman e Stanley (2011), de modo que a população atual complemente o arquivo de soluções representando os comportamentos visitados mais recentemente.

5.3 Resultados

A seguir, estão demonstrados os resultados obtidos e sua relação com as hipóteses levantadas na Introdução.

5.3.1 AG com *fitness* objetiva e NEAT com *fitness* objetiva

Para responder se o uso de um algoritmo genético padrão, com uma rede de controle bem definida, traz resultados superiores aos de uma rede evoluída pelo NEAT em uma quantidade limitada de iterações, é necessário comparar o uso de uma função de *fitness* objetiva para Algoritmo Genético Padrão e no NEAT.



(a) Resultados obtidos no mapa fácil

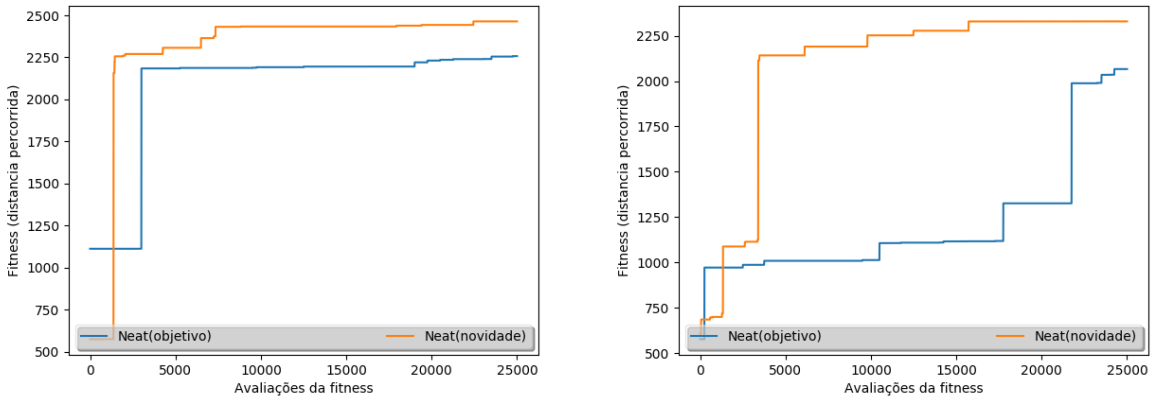
(b) Resultados obtidos no mapa difícil

Figura 5.2: Resultados para AG com *fitness* objetiva e NEAT com *fitness* objetiva

Através dos gráficos do mapa fácil e difícil, é possível ver que considerando o emprego de uma função de fitness objetiva, o uso de um algoritmo genético padrão com uma rede de controle pré-definida trouxe resultados superiores as redes evoluídas pelo NEAT.

5.3.2 NEAT com *fitness* objetiva e NEAT com *fitness* de novidade

A segunda hipótese a ser avaliada é se a busca por novidades influencia na qualidade das redes neurais evoluídas pelo NEAT. Para isso, foi observado os resultados do uso de *fitness* objetiva e da *Novelty Search* no NEAT.



(a) Resultados obtidos no mapa fácil

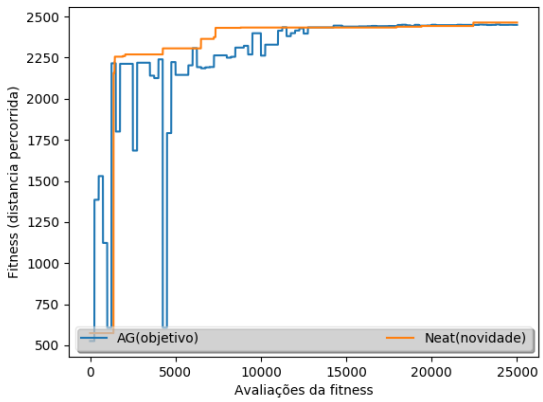
(b) Resultados obtidos no mapa difícil

Figura 5.3: Resultados para NEAT com *fitness* objetiva e NEAT com *fitness* de novidade

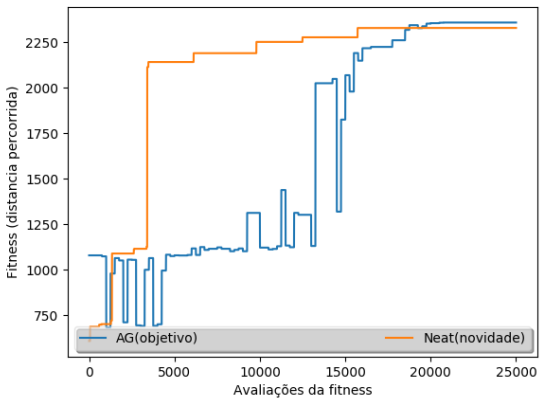
É possível perceber uma melhora significativa no desempenho do NEAT quando combinado com a *Novelty Search*. Este aumento no desempenho se deve a melhora na qualidade das redes neurais de controle, que apresentaram uma complexidade maior em indivíduos evoluídos pela busca por novidades. Portanto, a busca por novidades influenciou positivamente na geração de redes de controles mais evoluídas que as obtidas com o uso da *fitness* objetiva.

5.3.3 AG com *fitness* objetiva e NEAT com *fitness* de novidade

Outro ponto a ser avaliado é se a busca de novidades não é superior a uma busca por objetivos, considerando um problema que não possua caráter enganador ou ótimos locais. Para tal, foi decidido comparar o melhor resultado obtido com *fitness* objetiva e o melhor resultado obtido com a *Novelty Search*.



(a) Resultados obtidos no mapa fácil



(b) Resultados obtidos no mapa difícil

Figura 5.4: Resultados para AG com *fitness* objetiva e NEAT com *fitness* de novidade

Através dos gráficos acima é possível verificar que a busca por novidade não apresentou ganhos significativos de desempenho sobre o algoritmo genético padrão, se considerarmos o desempenho do melhor indivíduo ao final das 25.000 avaliações. Isso se deve a natureza do problema definido, que não sofre de ótimos locais.

Capítulo 6

Considerações finais

Este trabalho apresentou uma comparação de algoritmos genéticos para otimização da política de controle de agentes autônomos. Um simulador foi implementado para possibilitar um ambiente controlado e de rápida execução para os testes desejados.

A partir da análise dos resultados, verificou-se que para o problema proposto, buscas por novidade ou evolução da topologia da rede de controle não são superiores a uma abordagem de rede fixa e fitness baseada em objetivos. Em alguns casos, a abordagem usando NEAT não foi capaz de evoluir uma rede neural para aproximar a política de controle do carro de forma competitiva à rede definida no caso do algoritmo genético padrão, dentro do número máximo de avaliações utilizado.

Também é possível perceber que a abordagem utilizando Novelty Search foi superior ao teste de NEAT com fitness objetiva. Tal diferença se deu por que a exploração mais ampla proporcionada pela busca por novidades gerou redes de controle mais evoluídas que as obtidas com fitness objetiva.

Como o problema utilizado na avaliação não possui um caráter enganador, nem máximos locais, já era esperado que o algoritmo Novelty Search não apresentasse ganhos expressivos em relação ao algoritmo genético padrão. Porém foi possível perceber suas vantagens no caso dos testes com NEAT, visto que a procura por novos comportamentos refletiu em redes neurais mais complexas, que tiveram um melhor desempenho no controle dos carros.

Em trabalhos futuros, cabe uma análise mais detalhada do impacto da Novelty Search nas topologias das redes geradas pelo NEAT e como as diferenças no comportamento dos indivíduos é um reflexo das diferenças nas topologias.

Referências Bibliográficas

Csáji, B. C. (2001). Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M. e Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

Karpathy, A. (2015). Convolutional neural networks for visual recognition. `https://cs231n.github.io/neural-networks-1/`. Accessado: 2018-07-16.

LeCun, Y., Touresky, D., Hinton, G. e Sejnowski, T. (1988). A theoretical framework for back-propagation. Em *Proceedings of the 1988 connectionist models summer school*, volume 1, páginas 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann.

Lehman, J. e Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. Em *ALIFE*, páginas 329–336.

Lehman, J. e Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.

Maes, P. (1993). Modeling adaptive autonomous agents. *Artificial life*, 1(1_2):135–162.

Maes, P. (1995). Artificial life meets entertainment: lifelike autonomous agents. *Communications of the ACM*, 38(11):108–114.

Michalewicz, Z. (1996). Heuristic methods for evolutionary computation techniques. *Journal of Heuristics*, 1(2):177–206.

Stanley, K. O. e Miikkulainen, R. (2002). Efficient evolution of neural network topologies. Em *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, páginas 1757–1762. IEEE.

Tyrrell, T. (1993). *Computational mechanisms for action selection*. Tese de doutorado, University of Edinburgh Edinburgh, Scotland.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85.