ARMANDO HONORIO PEREIRA

RESOLUÇÃO DO PROBLEMA DE FORAGING UTILIZANDO REDES NEURAIS ARTIFICIAIS E ALGORITMOS GENÉTICOS

LAVRAS – MG

2014

ARMANDO HONORIO PEREIRA

RESOLUÇÃO DO PROBLEMA DE FORAGING UTILIZANDO REDES NEURAIS ARTIFICIAIS E ALGORITMOS GENÉTICOS

Artigo apresentado ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Raphael Winckler de Bettio

Co-Orientador
Ariel Felipe Ferreira Marques

LAVRAS – MG

2014

ARMANDO HONORIO PEREIRA

RESOLUÇÃO DO PROBLEMA DE BOX-PUSHING UTILIZANDO REDES NEURAIS E ALGORITMOS GENÉTICOS

Trabalho de Conclusão de Curso de Graduação apresentado ao Colegiado do Curso de Bacharelado em Ciência da Computação, para obtenção do título de Bacharel.

APROVADA em 26 de novembro de 2014.

Dr. Luiz Henrique Andrade Correia

Dr. Renato Ramos da Silva

Dr. Raphael Winckler de Bettio (Orientador)

Ariel F. F. Marques (Coorientador)

LAVRAS-MG
Novembro/2014

Resolução do problema de foraging utilizando redes neurais artificiais e algoritmos genéticos

Armando Honorio Pereira[1] **, Raphael Winckler de Bettio**[1]**, Ariel F. F. Marques**[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA) – Lavras, MG – Brazil

armandu@computacao.ufla.br, raphaelwb@dcc.ufla.br,
arielffmarques@gmail.com

Abstract. *Robots with the foraging behavior are capable of search objects and transport them to a collection point. They can be used to operate in the cleaning of safe and risk environments to man, like disaster areas caused by earthquakes or landslides. In this paper the foraging problem is broken in a search problem and a box-pushing problem, where in each one of them a genetic algorithm is trusted to evolve the neural network responsible to model the behavior of the mobile robot, determining its actions from the information retrieved of the environment through its sensors. The success of the evolved robot in the proposed problems confirm that the neural networks model the problems accordingly.*

Resumo. *Robôs com o comportamento de foraging são robôs capazes de procurar objetos e transportá-los para um ponto de coleta. Eles podem ser utilizados para atuar na limpeza de ambientes seguros e de risco para o homem, como em áreas de desastres causadas por terremotos ou desmoronamentos. Neste trabalho o problema de foraging é dividido em um problema de busca e um problema de box-pushing, onde em cada um deles um algoritmo genético é encarregado de evoluir uma rede neural responsável por modelar o comportamento do robô móvel, definindo suas ações à partir das informações obtidas do ambiente através dos seus sensores. O sucesso do robô evoluído nos problemas propostos confirma que as rede neurais modelam adequadamente os problemas.*

1. Introdução

Robôs móveis capazes navegar por um ambiente e, coletar objetos e carregá-los para uma determinada área, ou seja, o comportamento de *foraging*, tem inúmeras aplicações. Eles podem ser utilizados para atuar na limpeza de ambientes seguros ou de risco para o homem, como em áreas de desastres causadas por terremotos ou desmoronamentos. Também pode ser utilizados para áreas onde o acesso do homem não é possível, como na exploração espacial.

Robôs com o comportamento de *foraging* são capazes de procurar objetos e ao encontrá-los, transportá-los para um ponto de coleta [Winfield et al., 2013]. O objetivo deste trabalho é desenvolver um controle autônomo de robô, utilizando algoritmos genéticos e redes neurais artificiais, com o comportamento descrito anteriormente. Sob a perspectiva de [Winfield, 2009b] o problema de *foraging* neste trabalho é dividido em um problema de busca, onde o robô deve procurar por um caixa em um ambiente e coletá-la e em um problema de box-pushing, no qual o robô deve depositar a caixa em um localização de destino.

O problema de busca pode ser visto como um problema de planejamento de locomoção onde o objetivo é encontrar uma rota livre de colisões para atingir seu objetivo [Sadati and Taheri, 2002], semelhante a [Singh and Parhi, 2009] o controlador utilizado pelo robô é modelado com um rede neural artificial *feedforward*. No problema de *box-pushing* o robô deve empurrar uma caixa para uma localização específica [po Lee et al., 1997], assim como [Sprinkhuizen-Kuyper, 2001] este trabalho utiliza redes neurais artificiais *feedfoward* para controlar o robô, embora o robô possua menos sensores. Diferentemente dos trabalhos da literatura que utilizam o simulador *Khepera* esse trabalho utiliza uma plataforma de simulação própria com motor de física.

O robô evoluído para o problema de busca foi capaz de resolver dois dos três cenários propostos para o problema enquanto em um deles ele tinha dificuldades. A rede neural artificial evoluída pelo algoritmo genético foi capaz de resolver todos os cenários propostos para o problema de *box-pushing* nas instâncias simuladas para o problema. O sucesso do robô evoluído nos três cenários confirma que a rede neural modelada para controlá-lo representa adequadamente o problema.

O documento está organizado da seguinte forma, na seção 2 é apresentado o referencial teórico abordando os principais conceitos teóricos envolvidos e os trabalhos relacionados. Na seção 3 é descrito como o problema foi modelado. Na seção 4 são descritos os problemas abordados e na seção 5 discutidos os resultados para eles. Por fim, na seção 6 são expostas as considerações finais e abordado os trabalhos futuros.

2. Referencial teórico

Nesta seção são apresentados e explicados os conceitos fundamentais sobre algoritmos genéticos, redes neurais artificiais e os problemas de *foraging* e *box-pushing* tratados neste trabalho. Além disso, são apresentados trabalhos da literatura que envolvem a aplicação de algoritmos genéticos e redes neurais artificiais ao problema de *box-pushing*.

2.1. Algoritmo genético

Um algoritmo genético começa com um conjunto de indivíduos, cada um representado como uma string sobre um alfabeto, gerados aleatoriamente chamado de população [Russell and Norvig, 2010]. Uma função objetivo chamada função de *fitness* é usada para avaliar cada indivíduo da população. A reprodução é feita utilizando o operador de *crossover* com os melhores indivíduos selecionados da população. Os indivíduos também podem ser suscetíveis a uma mutação aleatória onde uma parte de sua representação é alterada [Poli et al., 2008]. As gerações futuras da população são criadas através dos processos de seleção e utilização dos operadores genéticos mencionados anteriormente. O algoritmo é geralmente executado até que um número fixado de gerações seja atingido ou até que alguma solução que satisfaça um critério seja encontrada [Russell and Norvig, 2010].

2.2. Redes neurais artificiais

Redes neurais artificiais são uma forma de computação que utiliza sistemas que se assemelham ao cérebro humano, estes sistemas são constituídos de unidades simples de processamento, também chamados de neurônios, organizados em camadas e ligados por

conexões com pesos associados, que calculam funções a partir dos dados de entrada [Anderson, 1995]. O conhecimento adquirido pela rede neural é obtido através de um algoritmo de aprendizagem, responsável por ajustar os pesos das ligações entre os neurônios. O desempenho obtido por uma rede neural depende de sua topologia assim como dos parâmetros de seu algoritmo de aprendizagem [Haykin, 2001].

Para modelar o comportamento dos robôs nos problemas propostos, este trabalho utilizada redes neurais artificiais com topologia *feedforward*, também chamadas de rede alimentadas adiante de múltiplas camadas, que são estruturas simples capazes de resolver problemas de múltiplas variáveis onde a saída depende somente das suas entradas e camadas ocultas [Haykin, 2001] [Bishop, 2006] .

2.3. Ambiente de simulação

O presente trabalho utiliza uma plataforma desenvolvida pelo Grubi (Grupo de redes ubíquas) no Departamento de Ciência da Computação da Universidade Federal de Lavras para simular os robôs móveis autônomos. A plataforma permite a modelagem e implementação de robôs móveis de várias morfologias, é possível criar robôs com vários sensores, rodas e formas específicas. A plataforma também permite que sejam desenvolvidos ambientes para que o comportamento do robô seja simulado e avaliado. A plataforma foi desenvolvida utilizando o motor de física ODE4J [1], ele permite a detecção de colisões e a simulação da dinâmica de corpos rígidos. O motor de física ODE permite simular robôs, objetos e um ambiente tridimensional de realidade virtual dentro da plataforma desenvolvida.

2.4. Foraging

No problema de *foraging*, um único robô ou um grupo de robôs deve recolher objetos espalhados por um ambiente [Cao et al., 1997]. Robôs com o comportamento de *foraging* são robôs capazes de procurar objetos e ao encontrá-los, transportá-los para um ponto de coleta [Winfield et al., 2013]. Esse comportamento é importante pois abstrai uma classe de problemas de exploração e transporte, em sistemas multi-robôs é um problema estabelecido para avaliação de cooperação de robôs e além disso muitas aplicações reais de robôs são instâncias do problema de *foraging*, tais como limpeza, *search and rescue* e exploração espacial [Winfield, 2009a].

De acordo com [Stephens and Krebs, 1986] o problema de *foraging* pode ser definido como a seguinte sequência de ações: busca, contato e decisão. O termo *foraging* significa procura de alimento, a busca feita por um robô e a busca feita por um animal possuem similaridades, o robô deve ser capaz de reconhecer o alvo de sua busca, ele deve conduzir a busca em um ambiente desconhecido sem a certeza de que atingirá o objetivo, o agente pode ser limitado por recurso e tempo e saber mais de uma estratégia para uma dada situação [Suarez and Murphy, 2011].

Segundo [Winfield, 2009b] o comportamento de *foraging* pode ser representado com quatro estados, buscar, agarrar, regressar e depositar. No estado de busca o robô está procurando, utilizando seus sensores, objetos dentro do espaço de busca, ao encontrá-lo o robô passa para o estado de agarrar, onde o robô captura o objeto para transporte, assim que o robô agarra o objeto ele entra no estado de regresso onde deve transportar o

[1] http://www.ode4j.org/

objeto coletado para um local de destino, ao alcançar o destino o robô deve depositá-lo. Após a conclusão desses estados o robô pode novamente resumir o estado de busca. Sob essa perspectiva o problema de *foraging* pode ser dividido em dois problemas, sendo o primeiro um problema de busca e o segundo, englobando a sequência de estados agarrar, regressar e depositar, pode ser abstraído para um problema de *box-pushing*.

O problema de *box-pushing* pode ser visto como um comportamento de baixo nível onde o objetivo é empurrar um objeto dentro de um determinado ambiente [Spronck et al., 2001]. O problema pode ser encontrado na literatura utilizando um único agente ou múltiplos. O comportamento básico de empurrar uma caixa é relevante para futebol de robôs, navegação e manipulação de objetos [Sprinkhuizen-kuyper et al., 2001].

Ao realizar o comportamento de *box-pushing* o robô deve permanecer sempre em contato com a caixa e não há restrição nas posições iniciais do robô e da caixa, sendo assim necessário que o robô mova deliberadamente para a posição onde a caixa está. A alteração do movimento de locomoção do robô devido a física de empurrar a caixa também contribui para a complexidade do problema [po Lee et al., 1997].

O presente trabalho tem como foco desenvolver um controle autônomo de robô, utilizando algoritmos genéticos e redes neurais artificiais, com um comportamento capaz de navegar por um ambiente procurando uma caixa e ao encontrá-la, movê-la para uma localização específica do ambiente, além de cumprir objetivos menores como evitar colisão da caixa e do robô.

2.5. Trabalhos relacionados

No problema de planejamento de locomoção o objetivo é encontrar uma rota livre de colisões na qual o robô pode segui-la para atingir seu objetivo a partir da posição inicial [Sadati and Taheri, 2002], dentro dessa abordagem o problema de busca é um problema de planejamento de locomoção onde o robô deve a partir de sua posição inicial, encontrar a caixa em uma dada posição do ambiente sem realizar colisões. Um controlador baseado em redes neurais artificiais é utilizado em [Singh and Parhi, 2009] para resolver o problema de planejamento do caminho do robô, o trabalho utilizada uma rede *feedforward* de quatro camadas para modelar o controlador do robô, a camada de entrada possui quatro neurônios para as informações dos sensores do robô, três neurônios recebe mos valores das distâncias dos objetos à frente, à esquerda e à direita do robô, a quarta entrada recebe o ângulo do destino. A saída da rede neural é o ângulo de direção para controlar a orientação do movimento do robô.

Em [po Lee et al., 1997] é introduzido um comportamento de *box-pushing* utilizando programação genética para cumprir a tarefa de empurrar a caixa para uma localização específica indicada por uma fonte de luz. Para completar a tarefa o autor a divide em duas subtarefas, *box-pushing* e *box-side-following*, o objetivo de *box-pushing* é manter o robô empurrando a caixa adiante enquanto o objetivo de *box-side-following* é garantir que o robô se mova junto ao lado da caixa. O comportamento é desenvolvido usando simulação e programação genética e depois é transferido para um robô real, o robô móvel *Khepera* [Mondada et al., 1994].

No trabalho [Sprinkhuizen-Kuyper et al., 2000] são avaliadas funções de *fitness* para o problema de *box-pushing*. Neste artigo são avaliadas quatro funções de *fitness*, externa global, externa local, interna global e interna local, aplicadas ao problema em

que um robô deve empurrar uma caixa entre duas paredes, com o objetivo de determinar qual é a melhor perspectiva para avaliação da função de *fitness* de robôs móveis no comportamento de *box-pushing*. O artigo conclui que a função de *fitness* de avaliação externa global é a mais apropriada para evoluir o comportamento de *box-pushing* entre duas paredes.

Em [Sprinkhuizen-Kuyper, 2001] é descrito o desenvolvimento e os resultados dos experimentos para desenvolver um controlador utilizando redes neurais para a tarefa de *box-pushing* entre duas paredes. É aplicado algoritmos evolucionários para o desenvolvimento dos controladores dos robôs. O artigo analisa quais funções de *fitness* obtém os melhores resultados, que tipo de topologia de rede neural é preferível e quais parâmetros do algoritmo evolutivo fornecem os melhores resultados. Os pesos da rede neural são aprendidos usando um algoritmo evolucionário. O artigo conclui que a função de *fitness* externa global, que utiliza informações do ambiente externas ao robô, retorna os melhores resultados e por isso ela é usada em todos os experimentos. Para determinar qual tipo de controlador fornece os melhores resultados para o comportamento de *box-pushing* são utilizados dois tipos de redes neurais, *feedforward* e recorrentes. A partir dos experimentos e resultados os autores concluem que a topologia de rede neural recorrente gera resultados melhores que as redes *feedforward*.

No artigo [Spronck et al., 2001] é avaliado qual tipo de controlador produz o melhor desempenho para o problema de *box-pushing*, ele constata que redes recorrentes possuem um melhor desempenho que redes *feedforward* mesmo se for utilizada uma representação mais geral de rede *feedforward*. Neste trabalho é abordado o problema de empurrar uma caixa entre duas paredes usando a mesma função de *fitness* de [Sprinkhuizen-kuyper et al., 2001]. Segundo os autores isso acontece porque as redes recorrentes são capazes de controlar as rodas com maior sutileza do que redes *feedforward*.

Semelhante ao trabalho [Sprinkhuizen-Kuyper, 2001] o presente trabalho utiliza redes neurais artificiais *feedfoward* para controlar o robô, embora não sejam utilizadas redes recorrentes, além disso o robô deste trabalho possui quatro sensores enquanto o robô no trabalho citado possui oito. No trabalho [Sprinkhuizen-Kuyper et al., 2000] as simulações são realizadas no simulador *Khepera*, que é um simulador bidimensional, enquanto neste trabalho as simulações são feitas em um ambiente tridimensional com um motor de física. As funções de *fitness* neste trabalho possuem informações externas ao robô assim como em [Sprinkhuizen-Kuyper et al., 2000].

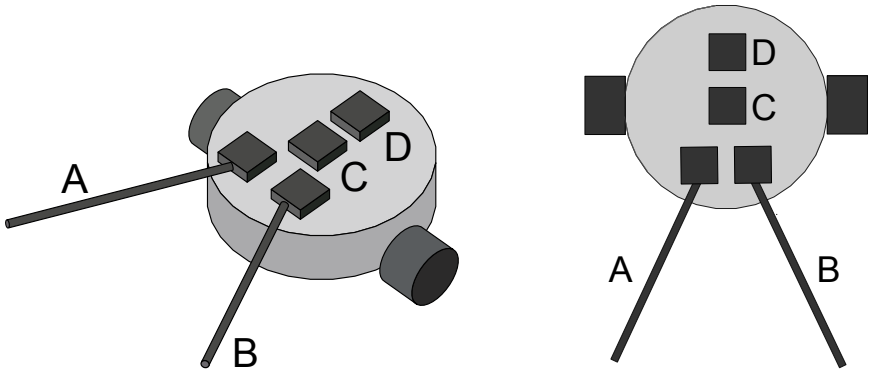
3. Modelagem do problema

O objetivo deste trabalho é desenvolver um robô móvel capaz de navegar de maneira autônoma por um ambiente procurando por uma caixa, e ao encontrá-la, carregá-la para uma localização pré-determinada do ambiente. O problema pode ser abstraído para o problema de *foraging*. Segundo Winfield [2009b] o problema de *foraging* pode ser representado com quatro estados, buscar, agarrar, regressar e depositar. Sob essa perspectiva o problema pode ser dividido em duas partes: no problema de encontrar a caixa, entrar em contato com ela e pegá-la e no problema de arrastá-la para uma determinada área do ambiente. Neste trabalho foi utilizada essa divisão, sendo o primeiro estado abstraído para um problema de busca e os outros três estados para um problema de *box-pushing*. Sendo assim o robô pode usar uma máquina de estados para decidir suas ações, caso ele esteja à

procura da caixa ele utiliza uma rede neural e se ele estiver arrastando a caixa outra rede neural.

3.1. Hardware do robô

Na Figura 1 é exibido a parte física do robô. O robô é formado por um chassi cilíndrico, duas rodas laterais que funcionam como seus motores e quatro sensores que recebem informações sobre o ambiente. Os sensores indicados por A e B na Figura 1 tem como função evitar que o robô colida dentro do ambiente, no caso eles fornecem a distância que um objeto detectado se encontra do robô. Os sensores indicados por C e D são usados para auxiliar o robô na busca pela caixa dentro do ambiente, eles captam a luz emitida pela caixa, sendo a intensidade do sensor inversamente proporcional a sua distância da caixa, são usados dois sensores para que a direção da caixa possa ser discernida.



(a) Visão tridimensional do robô. (b) Visão superior do robô.

Figura 1. Robô móvel utilizado no ambiente de simulação.

3.2. Software do robô

O comportamento do robô no problema de busca e no de *box-pushing* é modelado por uma rede neural artificial que a partir das informações obtidas do ambiente de simulação pelos sensores do robô, mapeia suas ações de acordo com o problema sendo solucionado.

Na Figura 2 é exibida a topologia da rede neural para o problema de busca da caixa no ambiente, ela possui três neurônios de entrada, 10 neurônios na camada oculta e um neurônio de saída. Duas das entradas recebem as informações vindas dos sensores A e B disposto na Figura 1, a terceira entrada recebe a direção da luz processada a partir dos sensores C e D exibidos na mesma figura.

Para descobrir a direção da luz a intensidade recebida em cada sensor é comparada, se a diferença entre elas for menor do que 0.1 a direção da luz é para frente do robô, se for maior que 0.15 a direção é para à esquerda, caso contrário a direção é para a direita. O raio do campo de alcance da luz emitida pela caixa, definido como 70 unidades no ambiente de simulação, pode ser visto na Figura 4(a), quanto menor a distância do robô para a caixa maior será a intensidade de luz recebida.

A saída da rede neural é o comportamento que o robô deve realizar, que são três no total, andar para frente, virar para esquerda e virar para a direita. A ré não é incluída no comportamento do robô porque ele não possui sensores em sua traseira, além disso, o efeito do comportamento de ré pode ser emulado com as ações disponíveis.

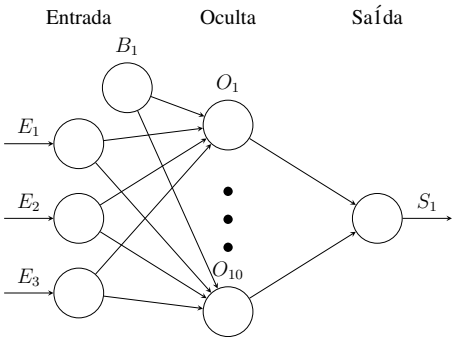


Figura 2. Topologia *feedforward* da rede neural que modela o comportamento de busca do robô.

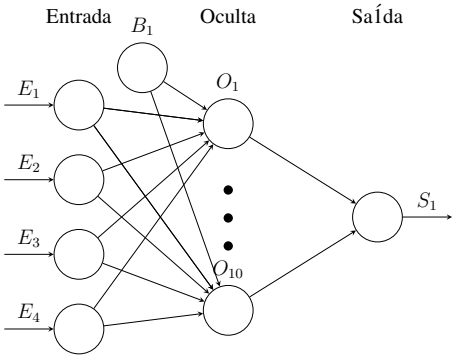
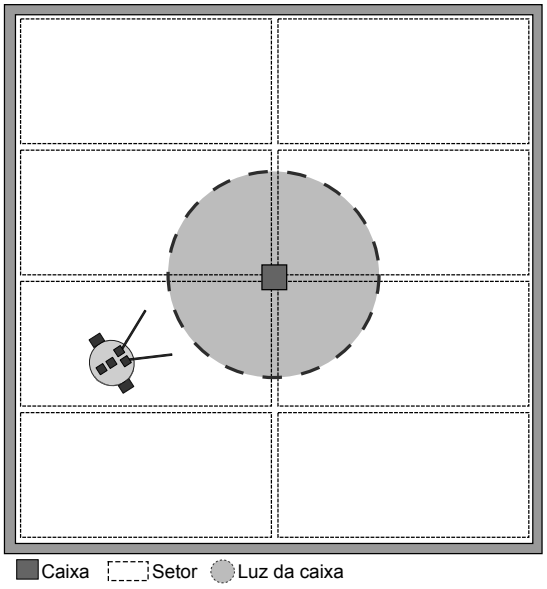
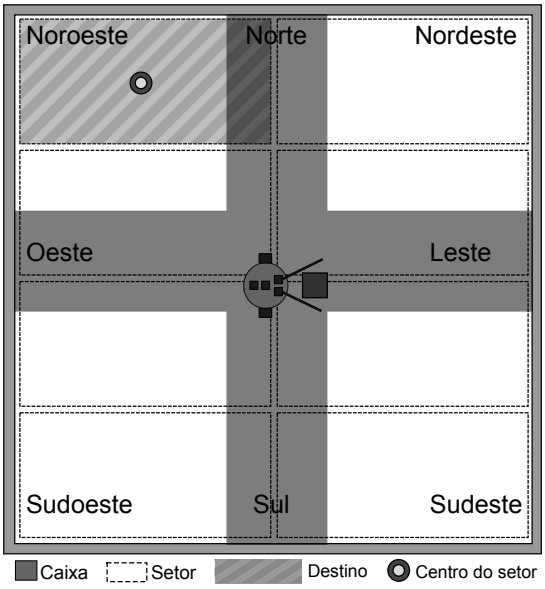


Figura 3. Topologia *feedforward* da rede neural que modela o comportamento de *box-pushing* do robô.

A topologia da rede neural para o problema de *box-pushing* é apresentada na Figura 3. Para evitar colisões no ambiente a rede neural recebe como entradas, assim como no problema de busca, as informações dos sensores A e B dispostos na Figura 1. A terceira entrada é a direção do setor que o robô deve atingir, que pode ser norte, nordeste, noroeste, sul, sudeste, sudoeste, leste e oeste.



(a) O robô deve determinar a direção da caixa a partir da luz emitida por ela.



(b) O robô deve determinar a direção que deve empurrar a caixa a partir da sua posição e o centro do setor de destino.

Figura 4. Definição da direção que o robô deve seguir em cada um dos problemas propostos.

Na Figura 4(b) é mostrado as direções que o setor pode estar em relação ao robô. A direção do setor é calculada a partir das diferenças das coordenadas X e Y do centro do setor e da posição atual do robô, no caso são checadas as posições em que as diferenças estão em relação a largura da cruz, definida como 40 unidades dentro do ambiente de simulação, que acompanha o robô, e.g. se a diferença das coordenadas X for negativa e das coordenadas Y for positiva e maiores em módulo que a largura da cruz temos que o

setor está a noroeste do robô. De acordo com a Figura 4(b), seja o ponto $A = (50, 175)$ igual ao centro do setor e ponto $B = (100, 100)$ a posição atual do robô na Figura 4(b). Temos que a diferença das coordenadas X dos pontos A e B é -50 e das coordenadas Y é 75 e consequentemente o setor está a noroeste do robô.

A quarta entrada é a direção em que o robô está se locomovendo dentro do ambiente, que pode ser para cima, baixo, esquerda ou direita, ela é calculada a partir do ângulo do robô dentro do ambiente. Se o ângulo do robô for maior que $315°$ e menor que ou igual a $45°$ a direção é para direita, se o ângulo do robô for maior que $45°$ e menor que ou igual a $135°$ a direção é para cima, Se o ângulo do robô for maior que $135°$ e menor que ou igual a $225°$ a direção é para esquerda, caso nenhuma das condições anteriores sejam satisfeitas a direção é para baixo, ela funciona em conjunto com a direção do setor para guiar a locomoção do robô até o local onde deve largar a caixa.

Todas as entradas constantes para a rede neural são discretizadas para o intervalo $[-1, 1]$. Por exemplo, a direção do setor pode assumir 8 valores constantes dependendo da posição do robô, dessa maneira, o intervalo $[-1, 1]$ é dividido em oito subintervalos e cada um deles é atribuído a uma das direções. Foi utilizada como função de ativação para as camadas da rede neural artificial a função $ActivationLOG$, disponível no *framework* utilizado para simular as redes neurais [2], exibida na equação (1). Foram realizados testes utilizando-se aprendizado supervisionado com uma rede neural pré-definida para determinar a função de ativação, a função $ActivationLOG$ mostrou-se a melhor opção, apresentando um baixo erro inicial e uma rápida convergência, isso pode ser visto no gráfico da Figura 5.

$$ActivationLOG(x) = \begin{cases} \log(1+x) & \text{se } x \geq 0 \\ -\log(1-x) & \text{se } x < 0 \end{cases} \tag{1}$$

3.3. Algoritmo genético

O algoritmo genético é responsável por evoluir a rede neural artificial que modela o comportamento do robô. A rede neural é convertida para uma representação em forma de *array* contendo os pesos das ligações entre os neurônios dela, uma representação do indivíduo pode ser vista na Figura 6. O cromossomo contendo os pesos da rede neural é manipulado pelo algoritmo genético para que a rede neural atinja o comportamento desejado, dessa forma, o algoritmo genético é o responsável pela calibragem de pesos da rede neural.

O operador de *crossover* trata o cromossomo da rede neural como um *array* de pontos flutuantes que representam os pesos da rede neural artificial. O operador recebe a representação genética dos dois pais e seleciona dois pontos de corte para os cromossomos que irão gerar um novo indivíduo, os dois pontos crião três regiões que terão seu material copiado para o filho, todos os cortes são aproximadamente do mesmo tamanho, isto é, o tamanho do *array* dividido por três. O operador pode ser visualizado na Figura 6, que mostra os pesos dos pais sendo copiados para o filho.

O operador de mutação realiza uma perturbação aleatória nos pesos da rede neural. Ele recebe como entrada a representação genética contendo os pesos da rede neural e

[2]http://www.heatonresearch.com/encog

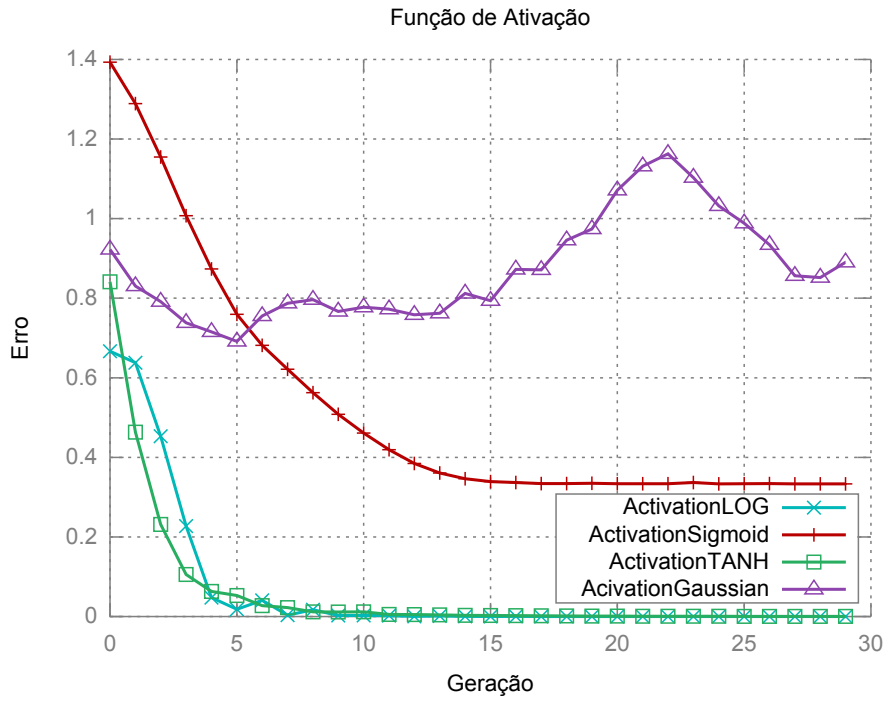


Figura 5. Gráfico de convergência para as funções de ativações testadas.

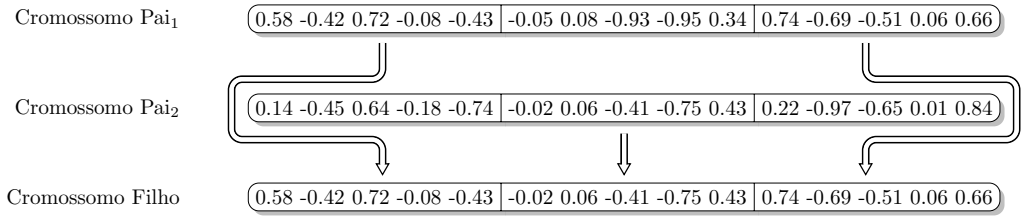


Figura 6. Operador de *crossover* com corte em dois pontos.

retorna como saída os pesos após a pertubação. O distúrbio que um peso da rede neural sofre é determinando por: $peso[i] = peso[i] + (4 - (random() * 8))$, onde $random()$ é um número aleatório maior que ou igual a 0.0 e menor que 1.0 e $peso[i]$ é um elemento na posição i do *array* de pesos da representação genética do indivíduo. Os valores para pertubação foram determinados empiricamente, além disso, é importante ressaltar que uma taxa baixa de mutação não irá inserir variedade genética na população e que também uma taxa muito alta de mutação pode acabar tornando o problema em uma busca aleatória.

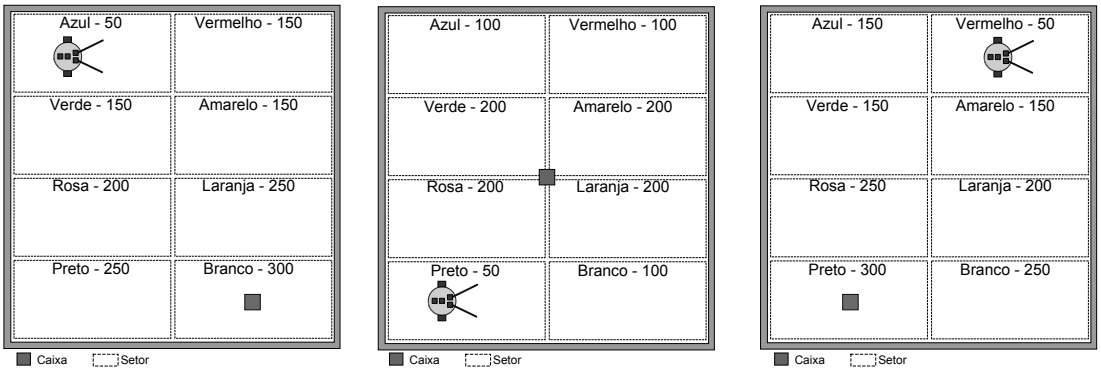
4. Problemas abordados

Nesta seção é apresentado o problema de busca e o problema de *box-pushing* que são abordados neste trabalho. O problema é inicialmente descrito e ilustrado, em seguida sua função de *fitness* é apresentada junto com a explicação dos seus termos, logo após, os parâmetros de configuração usados para a função de fitness e a simulação são relatados.

4.1. Problema de busca

O objetivo deste problema é desenvolver uma rede neural artificial capaz de navegar pelo ambiente e encontrar uma caixa contida no mesmo. O robô deve, a partir de uma posição

inicial P_1, navegar por um ambiente quadricular com lados de tamanho igual a 200 unidades do ambiente de simulação e, encontrar e agarrar uma caixa em uma posição P_2, a caixa emite uma luz que é capturada pelo robô, essa luz tem a função de apontar para o robô a direção que ele deve seguir, pois caso contrário o processo de busca se torna completamente aleatório. Para a resolução do problema de busca foram criados os cenários 1, 2 e 3 apresentados na Figura 7.



(a) Cenário 1: O robô deve sair do setor azul e pegar a caixa no setor branco.

(b) Cenário 2: O robô deve sair do setor preto e pegar a caixa no centro do ambiente.

(c) Cenário 3: O robô deve sair do setor vermelho e pegar a caixa no setor preto.

Figura 7. Cenários definidos para o problema de busca.

O cenário 1 possui a característica de incentivar uma rede neural que caminhe por todo o ambiente, principalmente pelos cantos, para poder encontrar a caixa. No cenário 2 o robô deve ser capaz de navegar pela área central do ambiente para poder pegar a caixa. O cenário 3 possui as mesmas característica que o cenário 1 para incentivar uma rede neural que caminhe por todo o ambiente, entretanto neste caso o comportamento incitado para o desenvolvimento da rede neural pode ser diferente do cenário 1, em vista da posição inicial diferente dos robôs e também da caixa.

4.1.1. Função de fitness

A função de *fitness* é responsável por indicar o comportamento desejado do robô, avaliando a qualidade do indivíduo gerado pelo algoritmo genético para resolver o problema. Dessa forma, a função de *fitness* é aplicada no processo evolutivo para qualificar os indivíduos. O objetivo da função de *fitness* é de maximização, portanto quanto maior o *fitness* melhor será o indivíduo. Os parâmetros utilizados para a função de *fitness* são:

Parâmetros:

C_s Custo de um setor.
C_{col} Custo de uma colisão.
C_{grab} Custo de pegar a caixa.
C_{inten} Coeficiente da intensidade da luz emitida pela caixa.
C_{dist} Custo da distância percorrida pelo robô.

Dados da simulação:

$Inten$ Maior intensidade de luz capturada pelo robô, indica o quão perto o robô esteve da caixa.

$Grab$ Informa se a caixa foi agarrada.
$Cols$ Número de colisões do robô.
$Dist$ Distância total percorrida pelo robô.
S Conjunto de setores visitados pelo robô.

A função de *fitness* de um cenário é dada por:

$$fitness_i = (\sum_{s \in S} C_s) - (Cols \times C_{col}) + (Inten \times C_{inten}) + (Grab \times C_{grab}) + (Dist \times C_{dist}) \tag{2}$$

onde $i \in \{1, 2, 3\}$. Depois de ter calculado a função de *fitness* de cada cenário a função de *fitness* final é dada por:

$$\mu = \frac{1}{3} \sum_{i=1}^{3} fitness_i \tag{3}$$

$$\sigma = \sqrt{\frac{1}{3} \sum_{i=1}^{3} (fitness_i - \mu)^2} \tag{4}$$

$$fitness = \sigma - \mu \tag{5}$$

Como o robô deve procurar a caixa pelo cenário, quanto mais ele vagar pelo cenário maior será suas chances de encontrá-la, portanto a função de *fitness* da equação (5) recompensa o robô de acordo com distância percorrida e com a quantidade de setores visitados, que também é usada para evitar que o robô ande em círculos. A função de *fitness* gratifica os robôs que agarraram a caixa, pois eles resolveram o problema e também favorece os robôs que passam nas proximidades da caixa já que eles estão mais perto de resolver o problema. O robô deve evitar colidir-se com o ambiente, portanto ele é punido por suas colisões. A mesma função de *fitness* é utilizada para os três cenários, a diferença está no custo de setor para cada um deles, o que pode ser visto na Figura 7.

O objetivo do robô é resolver todos os cenários apresentados na Figura 7, e não um cenário específico, para que dessa forma ele obtenha um comportamento geral para o problema. Como o robô deve ter um comportamento integral nos cenários, sua função de *fitness* deve depender simultaneamente do seu comportamento em cada um deles, a função de *fitness* do robô em um cenário individual é calculada usando a equação (2), a média e o desvio padrão de todos eles é calculada usando as equações (3) e (4) respectivamente. Por último, a função de *fitness* completo é dado pela equação (5), que privilegia concomitantemente os robôs que possuem uma função de *fitness* com média alta nos três cenários com um desvio padrão baixo.

Por exemplo, considere um indivíduo bom nos cenários 1 e 2, mas péssimo no cenário 3, nesse caso, apesar dele ter um valor alto para a média na equação (3), o desvio padrão na equação (4) também será alto, o objetivo da equação (5) é diminuir o desvio padrão entre a função de *fitness* de cada cenário mantendo a média entre eles o mais alto possível.

4.1.2. Parâmetros de configuração

Os valores dos parâmetros da função de *fitness* C_{col}, C_{grab}, C_{dist} e C_{inten} apresentados na Seção 4.1.1 são exibidos na Tabela 1, também são exibidos os outros parâmetros utilizados para a simulação. Os valores para o parâmetro C_s de cada setor são exibidos na Figura 7. Para cada unidade da intensidade de luz detectada pelo robô o coeficiente C_{inten} é incrementado em 10 unidades, por exemplo, uma intensidade de 1.2 tem coeficiente igual a 10 e uma intensidade de 2.6 tem coeficiente igual a 20.

Parâmetro	***Valores***
Dimensões da área de simulação	$(200, 200)$
Dimensões de um setor	$(50, 100)$
Intensidade da luz emitida pela caixa	$(0, 7)$
Steps da simulação	3000
C_{col} da equação (2)	100
C_{grab} da equação (2)	1600
C_{dist} da equação (2)	$\frac{1}{10}$
C_{inten} da equação (2)	$(10, 70)$

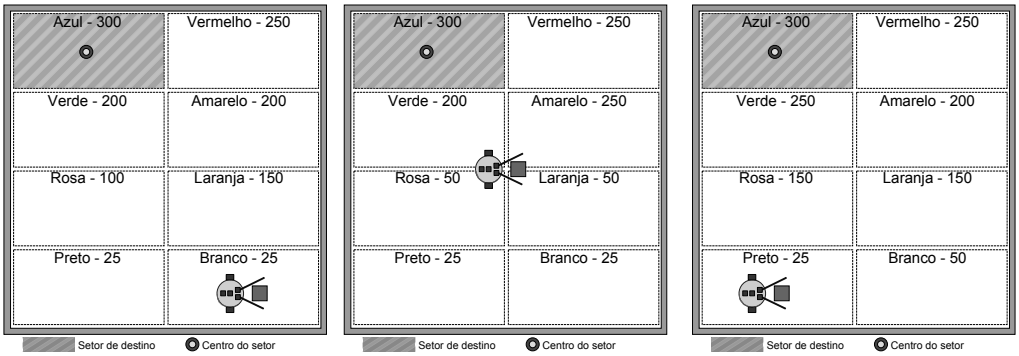
Tabela 1. Valores dos parâmetros da simulação para o problema de busca.

A seleção dos indivíduos para a realização do *crossover* foi feita utilizando a seleção de torneio com dois indivíduos, a seleção de torneio seleciona dois indivíduos aleatoriamente da população e retorna aquele com a melhor função de *fitness*, o processo é repetido duas vezes para selecionar os dois indivíduos para a reprodução. Também foi utilizado o elitismo, nesse caso o melhor indivíduo era automaticamente passado para a próxima geração. A taxa de mutação para as simulações foi de 20% e a taxa de *crossover* foi de 100%, ambos os valores foram determinados empiricamente através de testes.

4.2. Problema box-pushing

Neste problema o robô deve, à partir de uma posição P_1, empurrar uma caixa em uma posição P_2 para um setor, com lados de 100 unidades de comprimento e 50 unidades de altura no ambiente de simulação, limitado pelos pontos P_3 e P_4 dentro de um ambiente quadricular com lados de tamanho igual a 200. Como explicado na seção 2 o problema pode ser abstraído como o problema de *box-pushing*. O objetivo deste problema é evoluir uma rede neural, com a topologia exibida na Figura 3, para que ela seja capaz de arrastar a caixa para o setor destino dentro do ambiente, o problema é dificultado pelo fato de que a física de empurrar a caixa afeta a locomoção do robô já que ao andar em linha reta ele desvia incrementalmente para uma de suas laterais.

Para a resolução do problema foram criados três cenários que são apresentados na Figura 8. No cenário 1 o robô e a caixa estão no lado direito da parte inferior do ambiente, no cenário 3 eles também estão na parte inferior mas, dessa vez no lado esquerdo e no cenário 2 o robô e a caixa estão no centro do ambiente. Os cenários 1 e 3 foram definidos com o objetivo de manter o robô o mais distante possível do setor de destino, o cenário 2 foi escolhido devido a quantidade de direções iniciais que ele oferece para atingir o setor de destino. Para que o robô fosse capaz de resolver o problema de forma geral, os três



(a) Cenário 1 para o problema de *box-pushing*. **(b) Cenário 2 para o problema de *box-pushing*.** **(c) Cenário 3 para o problema de *box-pushing*.**

Figura 8. Cenários definidos para o problema de *box-pushing*.

cenários eram simulados com a mesma rede neural e a função de *fitness* avaliava o robô à partir do seu desempenho nos três cenários.

4.2.1. Função de fitness

Neste problema a função de *fitness* é responsável por caracterizar o comportamento de *box-pushing* do robô, classificando a qualidade do indivíduo gerado pelo algoritmo genético para resolver o problema, dessa forma a função de *fitness* é aplicada no processo evolutivo para avaliar os indivíduos examinando o quão próximos eles estão de resolver o problema. Assim como a função de *fitness* do problema de busca a função desse problema também é de maximização. Os parâmetros utilizados para a função de *fitness* são:

Parâmetros:

C_s Custo de um setor.
C_{col} Custo de uma colisão.
C_{drop} Custo de soltar a caixa dentro do setor de destino.
C_{DtS} Custo da distância até o centro do setor.
C_{DtI} Custo da distância do robô com relação a sua posição inicial.

Dados da simulação:

$Drop$ Informa se a caixa foi depositada dentro do setor.
$Cols$ Número de colisões do robô, incluí também o número de colisões da caixa.
DtI Distância do robô à sua posição inicial.
DtS Distância do robô ao centro do setor.
S Conjunto de setores visitados pelo robô.
P_i Posição inicial do robô.
P_f Posição final do robô.

A função de *fitness* de um cenário é dada por:

$$fitness_i = (\sum_{s \in Setores} C_s) - (Cols \times C_{col}) + (Drop \times C_{drop}) - (DtS \times C_{DtS}) + (DtI \times C_{DtI}) \quad (6)$$

onde $i \in \{1, 2, 3\}$. Depois de ter calculado o *fitness* de cada cenário o *fitness* final é dado por:

$$\mu = \frac{1}{3} \sum_{i=1}^{3} fitness_i \tag{7}$$

$$\sigma = \sqrt{\frac{1}{3} \sum_{i=1}^{3} (fitness_i - \mu)^2} \tag{8}$$

$$fitness = \sigma - \mu \tag{9}$$

Durante sua locomoção para deixar a caixa no setor de destino, o robô deverá passar pelos setores adjacentes ao objetivo, dessa forma, a função de *fitness* na equação (6) recompensará o robô que andar pelos setores adjacentes ao setor de destino. A função de *fitness* favorece os robôs que depositaram a caixa, pois estes resolveram o problema, e também favorece os robôs de acordo com sua proximidade do centro do setor de destino. Para evitar que o robô ande em círculos e incentivar seu deslocamento ele é beneficiado de acordo com sua distância em relação a sua posição inicial. O robô é punido pelas colisões dele e da caixa com o ambiente.

O objetivo do robô não é resolver um cenário específico, mas sim todos os cenários dispostos na Figura 8, dessa forma a função de *fitness* deve depender de seu desempenho dentro dos três cenários simulados, para isso, a função de *fitness* do robô em cada cenário é calculada usando a equação (6) e a média e o desvio padrão delas são calculadas usando as equações (7) e (8) respectivamente. Por fim a função de *fitness* geral é calculada usando a equação (9), onde nesse caso temos o robô com a melhor função de *fitness* em todos os cenários devido ao baixo desvio padrão no desempenho em cada um deles.

4.2.2. Parâmetros de configuração

Os valores para os parâmetros C_{col}, C_{drop}, C_{DtS} e C_{DtI} da função de *fitness* apresentados na seção 4.2.1 são exibidos na Tabela 2, assim como os outros parâmetros usados na simulação. Os valores para o parâmetro C_s de cada setor são exibidos na Figura 8.

Parâmetro	***Valores***
Dimensões da área de simulação	$(200, 200)$
Dimensões de um setor	$(50, 100)$
Steps da simulação	3000
C_{col} da equação (6)	100
C_{DtS} da equação (6)	1
C_{drop} da equação (6)	2500
C_{DtI} da equação (6)	1

Tabela 2. Valores dos parâmetros da simulação para o problema de *box-pushing*.

Para a seleção dos indivíduos para a realização do *crossover* foi utilizado a seleção de torneio com dois indivíduos assim como no problema de busca. Também foi utilizado o elitismo, onde o melhor indivíduo era automaticamente passado para a próxima geração. A taxa de mutação para as simulações foi de 20% e a taxa de *crossover* foi de 100%, ambos os valores foram determinados empiricamente com testes.

5. Resultados

Nesta seção são apresentados e discutidos os resultados das simulações para o problema de busca e *box-pushing*. Os resultados dos problemas abordados são dispostos em forma de gráfico com a média de *fitness* para cada geração nas simulações. Também é exibido um mapa de comportamento da execução do robô com a melhor função de *fitness* dentro do ambiente simulado, esse mapa exibe a trajetória feita pelo robô para atingir seu objetivo.

5.1. Problema de busca

Antes de resolver os três cenários em simultâneo, eles foram simulados individualmente, o robô nesse caso foi capaz de aprender a encontrar a caixa dentro do ambiente em cada um dos cenários definidos na Figura 7. Como o objetivo é desenvolver um robô capaz de resolver o problema de forma geral às execuções individuais serviram como uma forma de julgar a qualidade da modelagem da rede neural para o problema.

Para avaliar a capacidade do robô em aprender a resolver o problema de busca foram executadas 10 simulações usando os cenários dispostos na Figura 7, cada simulação evoluiu por 30 gerações usando uma população de 100 indivíduos, os resultados com a média da função de *fitness* para cada geração das simulações estão dispostos no gráfico da Figura 9 junto com a função de *fitness* do melhor indivíduo encontrado durante todas as execuções.

O robô evoluído foi capaz de resolver os cenários 1 e 3, entretanto o cenário 2 não foi resolvido pelo robô, isto pode ser visto na Figuras 10 que exibe o mapa de comportamento do robô com a melhor função de *fitness*. Isso acontece pois o robô desenvolvido para os cenários 1 e 3 tem um comportamento mais constante que o robô desenvolvido para o cenário 2, pois em ambos o robô deve percorrer os cantos, e este comportamento pode impedi-lo de visitar a região central do ambiente. Nos cenários 1 e 3 o robô se locomove pelas regiões dos setores próximas da fronteira do cenário, como estes cenários demandam que o robô visite as extremidades do ambiente, esse robô irá resolvê-los, entretanto o cenário 2 demanda um robô que seja capaz de caminhar pelas áreas centrais do ambiente, esse é um comportamento mais errático visto que o robô deve girar em ângulos pequenos para que sua locomoção permaneça na direção do centro. Outro fator que dificulta a captura da caixa no centro é que o robô possui somente dois sensores para a luz da caixa, o que impede de calcular com precisão a direção da caixa. Vale ressaltar que um dos indivíduos das 10 instâncias simuladas foi capaz de resolver simultaneamente os cenários 1 e 2.

Através do gráfico na Figura 9 é possível observar que o algormito genético prioriza os indíviduos com a melhor função de *fitness* total para os três cenários, ao invés de uma função de *fitness* que se sai bem em um cenário mas não é tão boa em outro, ou seja, indivíduos com uma função de *fitness* alta para um dado cenário mas baixa para outro,

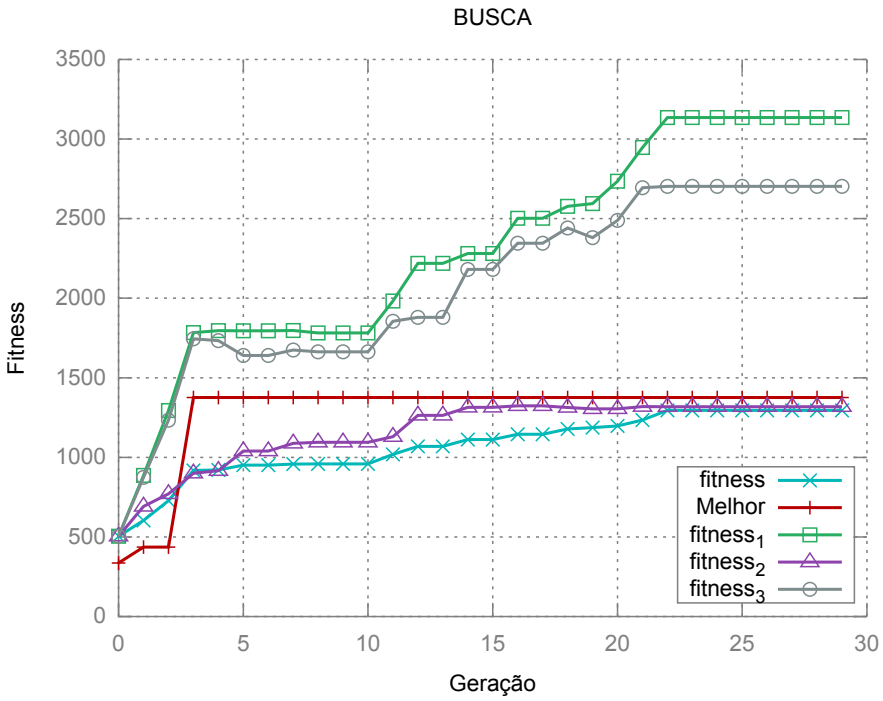
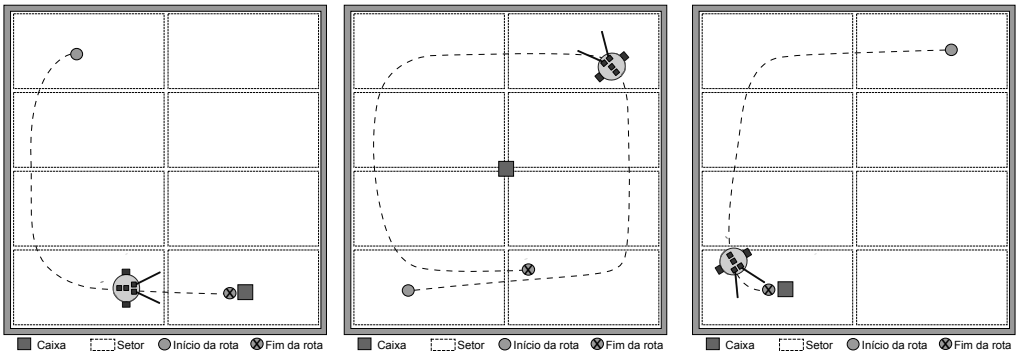


Figura 9. Fitness médio por geração, as curvas $fitness_i$ onde $i \in 1, 2, 3$ representam as médias por geração das funções de fitness da equação (2), a curva $fitness$ exibe a média da função de fitness da equação (5). A curva Melhor representa a função de fitness do melhor indivíduo evoluído nas simulações.



(a) Robô com o melhor *fitness* para o cenário 1.

(b) Robô com o melhor *fitness* para o cenário 2.

(c) Robô com o melhor *fitness* para o cenário 3.

Figura 10. Mapa de comportamento para o problema de busca baseado no vídeo https://www.youtube.com/watch?v=WuWI73TkaB4.

eram substituídos por indivíduos com uma função de *fitness* mais balanceada e com um menor desvio padrão para todos os cenários, mesmo que neste caso a função de *fitness* tivesse um valor menor para o cenário que antes ela possuía um valor maior. Isso pode ser visto no gráfico da Figura 9, observe que próximo da geração 5 o $fitness_3$ diminui enquanto o $fitness_2$ e o $fitness$ total aumentam.

5.2. Problema box-pushing

Para avaliar a modelagem do problema de *box-pushing*, foram executadas 10 simulações usando os cenários 1, 2 e 3 dispostos na Figura 8, cada simulação evoluiu por 30 gerações usando uma população de 100 indivíduos, para obter informação estatística sobre os resultados a média foi calculada para as funções de *fitness* de cada geração das simulações e estão dispostas no gráfico da Figura 11 junto com a função de *fitness* do melhor indivíduo encontrado. No gráfico da Figura 11 é possível perceber o progresso da função de *fitness* até que um indivíduo que resolvia o problema fosse atingido, a partir desse ponto a taxa de crescimento da função de *fitness* acontecia a uma taxa bem menor que antes. A importância de manter a diversidade na população é enfatizada pela diferença entre a função de *fitness* do melhor indivíduo e a média da população.

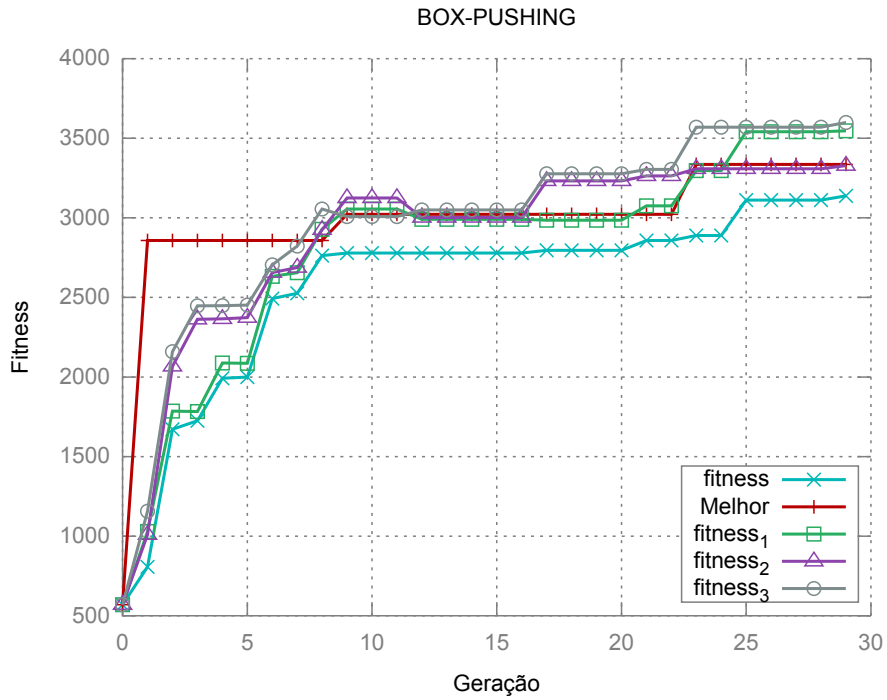
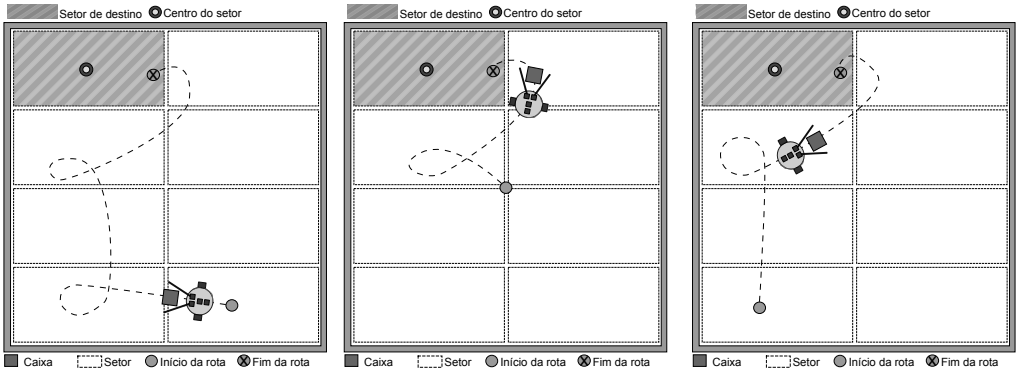


Figura 11. Fitness médio por geração, as curvas $fitness_i$ onde $i \in 1, 2, 3$ representam as médias por geração das funções de fitness da equação (6), a curva $fitness$ exibe a média da função de fitness da equação (9). A curva Melhor representa a função de fitness do melhor indivíduo evoluído nas simulações.

No gráfico da Figura 11 é possível perceber através do íngreme crescimento da melhor função *fitness*, o efeito do parâmetro C_{drop} da função de *fitness* que indica o custo de depositar a caixa dentro do setor de destino, o parâmetro funciona como um discriminante dos indivíduos que são capazes de resolver o problema dos que não são dentro da população. A Figura 12 exibe a solução da melhor função de *fitness* para os três cenários definidos, no caso é possível perceber a partir da figura que em todos os cenários o robô demonstrava um comportamento similar que servia como solução geral, o robô fazia uma curva no setor verde antes de se dirigir ao setor vermelho, de onde ele atingia o setor de destino.



(a) Robô com o melhor *fitness* para o cenário 1. **(b) Robô com o melhor *fitness* para o cenário 2.** **(c) Robô com o melhor *fitness* para o cenário 3.**

Figura 12. Mapa de comportamento para o problema de *box-pushing* baseado no vídeo https://www.youtube.com/watch?v=Vtd2IiXJJ3o.

A rede neural artificial evoluída pelo algoritmo genético foi capaz de resolver todos os cenários propostos para o problema de *box-pushing* nas 10 instâncias simuladas para o problema, usando em média 1392 *steps* do tempo total de 3000 *steps* destinados a execução. O sucesso do robô evoluído nos três cenários confirma que a rede neural modelada para controlá-lo representa adequadamente o problema à partir das informações obtidas do ambiente pelos sensores do robô.

6. Conclusão

Criar um controle de robô para realizar um determinado comportamento é uma tarefa complexa visto que as informações do ambiente que ele tem acesso e consegue interpretar são limitadas. Esse trabalho apresentou uma abordagem utilizando redes neurais artificiais e algoritmos genéticos para resolver o problema *foraging* em robótica móvel. O trabalho mostrou como a rede neural foi modelada para controlar o comportamento do robô a partir das informações que ele recebe do ambiente através de seus sensores de distância e de luz. Também foram apresentadas as funções de *fitness* para avaliar o desempenho do robô dentro dos cenários de simulação, as funções de *fitness* enfatizavam o comportamento geral do robô evitando que ele tivesse sucesso em somente um cenário ao invés de todos.

Os resultados mostram que o problema de busca e de *box-pushing* propostos neste trabalho foram resolvidos. No problema de busca o robô era capaz de evoluir e resolver individualmente cada um dos cenários propostos, na simulação geral ele resolvia dois dos cenários enquanto tinha dificuldade no outro. No problema de *box-pushing* o algoritmo genético foi capaz de evoluir um robô capaz de resolver todos os cenários propostos.

Como estudos futuros tem-se a integração do problema de busca e de *box-pushing* em único, onde nesse caso o algoritmo genético deve evoluir simultaneamente as duas redes neurais que modelam o comportamento do robô. Outro ponto de estudo é a adição de uma entrada aleatória na rede neural do problema de busca e a adição de mais sensores no robô para que ele seja capaz de calcular com precisão a origem da fonte de luz, nesse caso a entrada aleatória auxiliaria o robô a navegar pelo ambiente e a cobrir mais áreas, entretanto quando o robô detectasse a luz da caixa a entrada aleatória da rede deveria ser desativada. Ainda como estudo futuro tem-se a adição de mais dois fatores na função de

fitness, a quantidade de *steps* que ele usa para resolver o problema e também um fator para penalizar se o robô entrar em loop, o que acontecia em alguns indivíduos incapazes de resolver o problema.

Uma das principais contribuições desse trabalho é o fornecimento da modelagem da rede neural artificial responsável pelo comportamento do robô nos problemas de busca e *box-pushing*, o que pode ser usado como ponto de partida e referência para outros pesquisadores que desejam estender o problema ou abordá-lo sob outra perspectiva. Além disso, a divisão do problema em cenários com uma função de *fitness* para cada um deles e uma função de *fitness* geral que leva em conta a média e o desvio padrão pode ser utilizada como abordagem para outros problemas complexos que precisam de uma solução geral ao invés de somente uma específica.

Referências

J.A. Anderson. *An Introduction to Neural Networks*. A Bradford book. MIT Press, 1995. ISBN 9780262510813. URL `http://books.google.com.br/books?id=\_ib4vPdB76gC`.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Y. Uny Cao, Alex S. Fukunaga, and Andrew Kahng. Cooperative mobile robotics: Antecedents and directions. *Auton. Robots*, 4(1):7–27, March 1997. ISSN 0929-5593.

Simon Haykin. *Redes neurais : princípios e prática*. Bookman, Porto Alegre, RS, 2. ed. edition, 2001. ISBN 978-85-7307-718-6.

Francesco Mondada, Edoardo Franzi, and Paolo Ienne. Mobile robot miniaturization: A tool for investigation in control algorithms., 1994.

Wei po Lee, John Hallam, Henrik Hautop Lund, and Edinburgh U. K. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *In Proceedings of IEEE 4th International Conference on Evolutionary Computation*, pages 495–499. IEEE Press, 1997.

Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008. ISBN 1409200736, 9781409200734.

S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010. ISBN 9780136042594. URL `http://books.google.com.br/books?id=8jZBksh-bUMC`.

N. Sadati and J. Taheri. Solving robot motion planning problem using hopfield neural network in a fuzzified environment. In *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, volume 2, pages 1144–1149, 2002. doi: 10.1109/FUZZ.2002.1006665.

Mukesh Kumar Singh and Dayal R. Parhi. Intelligent neuro-controller for navigation of mobile robot. In *Proceedings of the International Conference on Advances in Computing, Communication and Control*, ICAC3 '09, pages 123–128, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-351-8. doi: 10.1145/1523103.1523129. URL `http://doi.acm.org/10.1145/1523103.1523129`.

Ida G. Sprinkhuizen-Kuyper. Artificial evolution of box-pushing behaviour, 2001.

Ida G. Sprinkhuizen-Kuyper, Rens Kortmann, and Eric O. Postma. Fitness functions for evolving box-pushing behaviour. In *PROCEEDINGS OF THE TWELFTH BELGIUM-NETHERLANDS ARTIFICIAL INTELLIGENCE CONFERENCE*, pages 275–282, 2000.

Ida G. Sprinkhuizen-kuyper, Eric O. Postma, and Rens Kortmann. Evolutionary learning of a robot controller: Effect of neural network topology. In *Proceedings of the Tenth Belgium-Dutch Conference on Machine Learning (BENELEARN'01*, pages 55–60, 2001.

Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Evolutionary learning of a neural robot controller. In *In Computational Intelligence for Modelling & Control, Proceedings of the CIMCA 2001, Idea Group Publishing*, pages 9–11. IOS Press, 2001.

David W. Stephens and John R. Krebs. *Foraging theory*. Princeton University Press, Princeton, 1986 1986.

J. Suarez and R. Murphy. A survey of animal foraging for directed, persistent search by rescue robotics. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 314–320, Nov 2011. doi: 10.1109/SSRR.2011.6106744.

Alan FT Winfield. Foraging robots. In *Encyclopedia of Complexity and System Science*, pages 3682–3700. Springer, 2009a.

Alan FT Winfield. Towards an engineering science of robot foraging. In *Distributed Autonomous Robotic Systems 8*, pages 185–192. Springer Berlin Heidelberg, 2009b.

Alan FT Winfield, Serge Kernbach, and Thomas Schmickl. Collective foraging: cleaning, energy harvesting and trophallaxis. 2013.