

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DEISE DA SILVA CÔRTEZ

**Modelo Neuro-Evolutivo de Coordenação
Adaptativa em Ambientes Dinâmicos**

Dissertação apresentada como requisito parcial
para a obtenção do grau de Mestre em Ciência
da Computação

Dr. Luis Otávio Campos Alvares
Orientador

Porto Alegre, outubro de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Côrtes, Deise da Silva

Modelo Neuro-Evolutivo de Coordenação Adaptativa em Ambientes Dinâmicos / Deise da Silva Côrtes – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

72 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Luis Otávio Campos Alvares.

1. Sistemas multiagentes. 2. Coordenação. 3. Neuro-evolução. 4. Redes Neurais. 5. Algoritmos Genéticos.

I. Alvares, Luis Otávio Campos. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. José Carlos Ferraz Hennemann

Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca

Pró-Reitora de Pós-Graduação: Profa. Valquíria Linck Bassani

Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux

Coordenador do PPGC: Prof. Flávio Rech Wagner

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Eu sempre soube que fazer uma dissertação de mestrado não seria fácil. Muito embora, as maiores dificuldades que encontrei tenham sido as menos esperadas.

Antes de tudo e de todos, agradeço a Deus, por ter me dado forças para continuar toda vez em que desistir parecia ser inevitável.

Agradeço a todas as pessoas que torceram por mim e me apoiaram na realização deste trabalho.

Em especial, agradeço:

À minha família, todos tão distantes durante a realização deste trabalho e ao mesmo tempo tão presentes em todos os momentos da minha vida. Em particular, agradeço à minha avó Lidinha, meus pais e irmãos, meus dindos e Dinha, que nem por um momento, deixaram de pedir por mim em suas orações. Às minhas priminhas Nine e Bele, que não se cansam de perguntar quando eu volto.

Ao meu querido orientador, professor Luis Otávio Alvares, pelo apoio, pelas discussões científicas e pela atenção.

Aos meus amigos de ontem, hoje e sempre, Rick e Nivinha, e ao MSN e ao Skype que tornam a saudade suportável e possível a presença constante de vocês na minha vida.

À Mari e Nelma, pelo carinho, atenção e cumplicidade nos meses em que convivemos aqui em POA.

À Vaninha, pela amizade, pelo carinho e pelos puxões de orelha!

À Didi e às suas aulas magníficas de dança contemporânea, que além de me propiciarem mágicos momentos de auto-conhecimento, tornaram a minha vida aqui muito mais prazerosa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS	6
LISTA DE FIGURAS	7
LISTA DE TABELAS	9
RESUMO	10
ABSTRACT	11
1 INTRODUÇÃO	12
2 CONCEITOS BÁSICOS	14
2.1 Agentes e Sistemas Multiagentes.....	14
2.2 Coordenação.....	15
2.3 Tarefas de Decisão Sequencial.....	15
2.4 Aprendizado por Reforço.....	16
2.4.1 <i>Exploration e exploitation</i>	18
2.4.2 Estratégias de Aprendizado por Reforço	18
2.5 Algoritmos Genéticos.....	22
2.5.1 <i>Exploitation x Exploration</i> em AG	24
2.6 Redes Neurais	25
3 NEURO-EVOLUÇÃO.....	27
3.1 A abordagem evolucionária	27
3.1.1 Treinamento esparsos	29
3.1.2 Tempo de treinamento	29
3.2 Codificação da Rede	29
3.3 Topologia da Rede	30
3.4 Variáveis da Evolução	30
3.5 Métodos de Neuro-Evolução	30
3.5.1 <i>Symbiotic, Adaptive Neuro-Evolution</i>	31
3.5.2 <i>Enforced Sub-Populations</i>	33
3.5.3 <i>NeuroEvolution of Augmenting Topologies</i>	35
4 MODELO DE COORDENAÇÃO MULTIAGENTE NEURO-EVOLUTIVO...40	
4.1 Algoritmo evolucionário.....	40

4.2	Características do Modelo Proposto	42
4.2.1	Cromossomos	42
4.2.2	Operadores Genéticos	44
4.2.3	Entrada e Saída das Redes Neurais	44
4.3	Domínio de Aplicação	45
4.4	Diagrama de Classes.....	46
4.5	Ambiente de Simulação	50
4.5.1	Manipulação dos Comportamentos	50
4.5.2	Configurações	52
4.5.3	Visualização das estratégias	54
5	EXPERIMENTOS	55
5.1	Experimentos Comparativos	55
5.2	Experimentos Variando Diversos Parâmetros.....	57
5.2.1	Entrada do Agente	58
5.2.2	Quantidade de Neurônios na Camada Oculta	60
5.2.3	Quantidade de ciclos evolucionários	61
5.2.4	Função de Aptidão	62
5.2.5	Mundo toroidal x limitado	62
5.2.6	Grau de dificuldade da tarefa-alvo	62
6	CONCLUSÕES.....	64
	REFERÊNCIAS.....	66

LISTA DE ABREVIATURAS E SIGLAS

AR	Aprendizado por Reforço
AE	Algoritmos Evolucionários
AG	Algoritmo Genético
DP	<i>Dynamic Programming</i>
ESP	<i>Enforced Subpopulations</i>
MDP	<i>Markov Decision Process</i>
NEAT	<i>NeuroEvolution of Augmenting Topologies</i>
SANE	<i>Symbiotic, Adaptative Neuro-Evolution</i>
TD	<i>Temporal Difference</i>

LISTA DE FIGURAS

Figura 2.1:	Modelo padrão do aprendizado.....	17
Figura 2.2:	Visão Geral da AHC.....	20
Figura 2.3:	Algoritmo Genético Simples.....	23
Figura 2.4:	Algoritmo de Seleção da Roleta.....	24
Figura 3.1:	Processos da abordagem evolucionária.....	28
Figura 3.2:	Passos básicos em uma geração do SANE.....	31
Figura 3.3:	Neuro-evolução no SANE.....	32
Figura 3.4:	Rede neural formada a partir dos cromossomos que definem a camada oculta da rede.....	33
Figura 3.5:	Neuro-evolução em ESP. A rede é constituída por um neurônio de cada população de neurônios.....	34
Figura 3.6:	Mapeamento do genótipo para o fenótipo.....	36
Figura 3.7:	Os dois tipos de mutação estrutural no NEAT.....	37
Figura 3.8:	Recombinação de duas redes usando o NEAT.....	38
Figura 4.1:	Algoritmo básico de treinamento.....	41
Figura 4.2:	Algoritmo de Delta-Coding.....	42
Figura 4.3:	Codificação do neurônio no modelo proposto.....	43
Figura 4.4:	Codificação da rede neural partindo de cromossomos binários.....	43
Figura 4.5:	Ciclo evolucionário para a captura da presa.....	46
Figura 4.6:	Principais classes do modelo de coordenação evolucionário.....	47
Figura 4.7:	Todas as classes do modelo evolucionário.....	48
Figura 4.8:	Classes do ambiente de simulação.....	49
Figura 4.9:	Manipulação de comportamentos no ambiente.....	50
Figura 4.10:	Carrega o comportamento dos predadores.....	51
Figura 4.11:	Salva o comportamento atual dos agentes.....	51
Figura 4.12:	Configuração dos predadores.....	52
Figura 4.13:	Configuração da presa.....	52

Figura 4.14: Configuração do treinamento.....	53
Figura 4.15: Configurações do mundo.....	53
Figura 4.16: Menu Execução.....	54
Figura 4.17: Visualização das estratégias aprendidas.....	54
Figura 5.1: Entradas das redes neurais dos agentes.....	58
Figura 5.2: Média das melhores aptidões dos agentes por ciclo evolucionário. Time constituído por 3 agentes.....	61
Figura 5.3: Média das melhores aptidões dos agentes por ciclo evolucionário. Time constituído por 4 agentes.....	62

LISTA DE TABELAS

Tabela 5.1:	Comparação das configurações adotadas para treinamento dos agentes.....	55
Tabela 5.2:	Comparação de resultados com o ESP.....	56
Tabela 5.3:	Passos para captura da presa.....	59
Tabela 5.4:	Taxa de captura por tamanho da camada oculta.....	60
Tabela 5.5:	Tempo de treinamento por tamanho da camada oculta.....	61

RESUMO

Em ambientes dinâmicos e complexos, a política ótima de coordenação não pode ser derivada analiticamente, mas, deve ser aprendida através da interação direta com o ambiente. Geralmente, utiliza-se aprendizado por reforço para prover coordenação em tais ambientes.

Atualmente, neuro-evolução é um dos métodos de aprendizado por reforço mais proeminentes. Em vista disto, neste trabalho, é proposto um modelo de coordenação baseado em neuro-evolução. Mais detalhadamente, desenvolveu-se uma extensão do método neuro-evolutivo conhecido como *Enforced Subpopulations* (ESP).

Na extensão desenvolvida, a rede neural que define o comportamento de cada agente é totalmente conectada. Adicionalmente, é permitido que o algoritmo encontre, em tempo de treinamento, a quantidade de neurônios que deve estar presente na camada oculta da rede neural de cada agente. Esta alteração, além de oferecer flexibilidade na definição da topologia da rede de cada agente e diminuir o tempo necessário para treinamento, permite também a constituição de grupos de agentes heterogêneos.

Um ambiente de simulação foi desenvolvido e uma série de experimentos realizados com o objetivo de avaliar o modelo proposto e identificar quais os melhores valores para os diversos parâmetros do modelo. O modelo proposto foi aplicado no domínio das tarefas de perseguição-evasão.

Palavras-Chave: Sistemas Multiagentes, Coordenação, Neuro-evolução, Redes Neurais, Algoritmos Genéticos.

A Neuro-Evolutive Model for Adaptative Coordination in Dynamic Systems

ABSTRACT

In dynamic and complex environments, the optimal policy for coordination cannot be analytically derived; it must be learned through direct interactions with the environment. Generally, reinforcement learning is used to provide coordination in those environments.

Nowadays, neuro-evolution is one of the most prominent methods for reinforcement learning. Therefore, in the present work, we propose a model for coordination based in neuro-evolution mechanisms. More specifically, the proposed model is an extension of a neuro-evolution method known as Enforced Subpopulations (ESP).

In our extension, an agent's neural network is fully connected. Additionally, we allow the algorithm to come up with the number of neurons that must be present in the hidden layer of an agent's neural network. This variation provides flexibility in the definition of the neural network topologies and decreases the amount of time necessary to training the agents.

Furthermore, we developed a simulation environment, which facilitates the execution of a series of experiments. Some of these experiments help the evaluation of the optimal values for the model parameters. The proposed model was applied in the pursuit-evasion tasks domain.

Keywords: Multi-Agent System, Coordination, Neuro-evolution, Neural Networks, genetic Algorithms.

1 INTRODUÇÃO

Embora não exista uma definição universalmente aceita para agentes, geralmente, podemos visualizar um agente como uma entidade capaz de desempenhar algumas atividades autonomamente para alcançar seus objetivos. Costuma-se também considerar que agentes são capazes de se comunicar com outros agentes e com o ambiente¹.

Além da autonomia e da capacidade de comunicação, um agente pode possuir diversas outras características, tais como reatividade (capacidade de reagir apropriadamente às influências ou informações de seu ambiente), proatividade (capacidade de um agente de tomar iniciativas sob circunstâncias específicas) e mobilidade (habilidade do agente navegar pela rede).

Sistemas multiagentes, por sua vez, podem ser definidos como sistemas computacionais em que vários agentes interagem ou trabalham em conjunto para desempenhar algum conjunto de tarefas ou satisfazer algum conjunto de objetivos (LESSER, 1995).

Devido a sua natureza distribuída, sistemas multiagentes podem ser mais eficientes, mais robustos, e mais flexíveis que abordagens centralizadas. Contudo, para serem efetivos, normalmente os agentes precisam agir de forma coordenada.

Coordenação é o processo pelo qual um agente raciocina sobre suas ações locais e as ações dos outros para tentar garantir que a comunidade se comporte de uma maneira coerente. Em ambientes dinâmicos e complexos, a política ótima de coordenação não pode ser derivada analiticamente, mas, deve ser aprendida através da interação direta com o ambiente. Aprendizado por reforço é a técnica geralmente utilizada para prover a coordenação de agentes em tais ambientes (BERENJI, 2000; HAYNES, 1995; HAYNES, 1996; MORIARTY, 1996; YONG, 2001). Na aprendizagem por reforço, os agentes aprendem através de sinais que fornecem alguma medida de desempenho após a realização de uma seqüência de ações.

Existem duas estratégias básicas para se resolver problemas de aprendizagem por reforço. A primeira é a busca no espaço de comportamentos, procurando por um comportamento que desempenhe bem no ambiente. Esta abordagem vem sendo geralmente implementada com algoritmos genéticos e programação genética. A segunda estratégia, mais tradicional, é o uso de técnicas estatísticas e métodos de programação

¹ Algumas entidades podem agir completamente por conta própria. Contudo, geralmente não se refere a este tipo de sistema como sistema baseado em agentes, desde que um dos aspectos que despertam interesse, riqueza e complexidade aos sistemas de agentes são as iterações entre eles.

dinâmica que estimam a utilidade de determinadas ações no espaço de estados do ambiente (mundo).

Segundo Kaelbling (1996) não é claro qual conjunto de abordagens é melhor em que circunstâncias. Contudo, os métodos tradicionais de aprendizado por reforço, que utilizam programação dinâmica, se apóiam num mapeamento dos estados e ações do mundo, tornando-os menos extensíveis. Isto é, quanto mais o espaço de estados e ações do mundo crescer, mais lento será o processo de aprendizagem. Alguns pesquisadores (YONG, 2001; MORIARTY, 1996) constataram soluções mais rápidas e eficientes ao utilizarem neuro-evolução, um dos métodos disponíveis para aprendizado por reforço em ambientes dinâmicos.

Moriarty (1997) mostra que métodos evolucionários tomam vantagem sobre os métodos tradicionais de aprendizado por reforço por terem um mecanismo de atribuição de crédito mais robusto. Outros investigadores (BELEW, 1993; NOLFI, 1994) descobriram que a neuro-evolução, ou evolução simulada de redes neurais, é uma estratégia efetiva para resolver problemas de aprendizado por reforço, mesmo quando o sinal de reforço é esparso. Isto é, mesmo quando o agente só é avaliado após a execução de uma seqüência de ações e não para cada ação tomada individualmente.

Neste trabalho, propõe-se um modelo de coordenação baseado em neuro-evolução. Estendemos o *Enforced Subpopulations*, método de neuro-evolução proposto por Gomez (1997), permitindo maior flexibilidade na definição da topologia da rede neural de cada agente.

No capítulo 2, é fornecida uma visão geral sobre as principais disciplinas abordadas neste trabalho, fala-se brevemente sobre tarefas de decisão seqüencial, aprendizado por reforço, algoritmos genéticos e redes neurais artificiais. No capítulo 3 é apresentada a neuro-evolução, suas principais características e alguns dos métodos de neuro-evolução existentes na literatura. No capítulo 4, apresenta-se o modelo de coordenação proposto. No capítulo 5, descreve-se um subconjunto dos experimentos realizados a fim de se avaliar o modelo proposto e definir valores adequados para os parâmetros do modelo. Por fim, no capítulo 6, são apresentadas as conclusões deste trabalho.

2 CONCEITOS BÁSICOS

Este capítulo provê uma visão geral dos diversos conceitos relacionados ao presente trabalho. O objetivo deste trabalho foi o desenvolvimento de um modelo para coordenação de sistemas multiagentes em ambientes dinâmicos e complexos. Aprendizado por reforço é uma das técnicas mais utilizadas para aprendizagem de máquina, é também uma das técnicas mais adequadas para aprendizagem em ambientes dinâmicos e complexos, onde, na maioria das vezes, o comportamento ideal não é previamente conhecido. No aprendizado por reforço, não se estabelece, *a priori*, como cada agente deve se comportar. Ao invés disso, o agente aprende seu comportamento a partir dos sinais de reforço que recebe do ambiente. Estes sinais de reforço indicam quão bem ou quão mal o agente se comporta.

A neuro-evolução, ou evolução de redes neurais por algoritmos genéticos, é uma das formas existentes de aprendizado por reforço. Uma das vantagens da neuro-evolução é que ela possui um bom desempenho, mesmo para problemas de aprendizado por reforço em que o sinal de reforço é esparso (MORIARTY, 1997). Isto é, ao invés do ambiente enviar um sinal de reforço para cada ação do agente, o sinal de reforço é enviado após a realização de um conjunto de ações. Isto é o que geralmente ocorre para uma classe de problemas conhecida como tarefas de decisão seqüencial. Uma tarefa é dita de decisão seqüencial quando uma seqüência de ações precisa ser realizada para que o efeito delas possa ser medido.

2.1 Agentes e Sistemas Multiagentes

Neste trabalho, estamos considerando um agente como uma entidade capaz de realizar algumas tarefas, de forma autônoma, para atingir seus objetivos. Geralmente, agentes são capazes de se comunicar com outros agentes e com o ambiente².

Além da autonomia e da capacidade de comunicação, um agente pode possuir diversas outras características, tais como reatividade (capacidade de reagir apropriadamente às influências ou informações de seu ambiente) e pro-atividade (capacidade de um agente de tomar iniciativas sob circunstâncias específicas).

² Algumas entidades podem agir completamente por conta própria. Contudo, geralmente não se refere a este tipo de sistema como sistema baseado em agentes, desde que os aspectos que despertam interesse, riqueza e complexidade aos sistemas de agentes são as iterações entre agentes.

Pode-se visualizar sistemas multiagentes, como sistemas computacionais em que vários agentes interagem ou trabalham em conjunto para desempenhar algum conjunto de tarefas ou satisfazer algum conjunto de objetivos (LESSER, 1995).

Sistemas multiagentes tendem a ser mais eficientes, robustos e flexíveis que abordagens centralizadas. Porém, para serem efetivos, geralmente os agentes precisam agir de forma coordenada.

2.2 Coordenação

Segundo Kahn (2000), o desafio das aplicações como um todo, do ponto de vista de processamento e comunicação, é como implementar comportamentos complexos juntando o comportamento de um conjunto de indivíduos.

Todos nós temos um senso intuitivo do significado da palavra “coordenação”. Ao assistirmos um jogo de vôlei ou de futebol, nos damos conta de quão bem coordenadas são as ações desempenhadas por um grupo de pessoas. Frequentemente, contudo, nós notamos mais facilmente a falta de coordenação: quando o hotel em que fizemos reserva está lotado, ou quando esperamos por horas no aeroporto porque a linha aérea não sabe informar qual o portão de embarque.

De maneira geral, esse significado intuitivo do que é coordenação é suficiente. Contudo, ao tentar caracterizar uma nova área de estudo interdisciplinar, é importante se ter uma idéia mais precisa do que é coordenação.

Algumas definições de coordenação incluem:

“Coordenação consiste dos protocolos, tarefas e mecanismos de tomada de decisão projetados para alcançar objetivos comuns entre unidades inter-dependentes” (THOMPSON, 1967)

“Os esforços conjuntos de atores comunicantes independentes através de objetivos mutuamente definidos” (NSF, 1989)

“A integração e o ajuste harmonioso dos esforços individuais para cumprir um objetivo maior” (SINGH, 1992)

“Coordenação é gerenciar dependências entre atividades” (MALONE, 1994)

Para a proposta deste trabalho, contudo, consideraremos coordenação como o processo pelo qual um agente raciocina sobre suas ações locais e as ações dos outros para tentar garantir que a comunidade se comporte de uma maneira coerente.

2.3 Tarefas de Decisão Seqüencial

Tarefas de decisão seqüencial (MORIARTY, 1997) estão entre os problemas mais gerais e difíceis da aprendizagem de máquina. Uma tarefa é dita de decisão seqüencial se seu resultado só pode ser conhecido após a tomada de uma seqüência inteira de decisões.

Tarefas de decisão seqüencial (BARTO, 1990; GREFENSTETTE, 1990) podem ser caracterizadas pelo seguinte cenário: um agente observa o estado de um sistema dinâmico e escolhe uma ação **a** de um conjunto finito de ações. O sistema então fornece um novo estado, a partir do qual o agente deve selecionar uma outra ação. O sistema pode retornar uma recompensa ou após cada decisão do agente ou após um conjunto de

decisões. O objetivo é selecionar a seqüência de ações que retorne a maior recompensa acumulada. Geralmente, a melhor estratégia não é maximizar a recompensa por cada ação individual. Pois, algumas ações podem produzir altas recompensas, mas, levar a estados a partir dos quais não é possível receber recompensas altas posteriormente.

O fato de que as decisões freqüentemente levam a conseqüências tanto imediatas quanto futuras dificulta a solução de tarefas de decisão seqüencial. Em jogos como o xadrez, por exemplo, muitas vezes é difícil saber se um movimento isolado deve ser avaliado como bom, como ruim ou neutro. Considere, por exemplo, a captura de uma peça. Esta ação pode conseguir uma recompensa imediata alta, pois, diminui a quantidade de peças do adversário. Contudo, esta mesma captura pode levar a uma recompensa negativa no futuro se a peça responsável pela captura, sair de uma posição defensiva chave, desprotegendo o rei. O resultado de uma partida de xadrez só é conhecido após muitas decisões.

A estratégia de decisão deve levar em conta tanto as recompensas imediatas quanto as futuras de forma a otimizar a recompensa total. Logo, é difícil determinar como ajustar políticas em uma tarefa de decisão seqüencial. Minsky (1963) chamou este problema de atribuição de crédito.

O aprendizado por reforço é uma abordagem simbólica robusta para tarefas de decisão seqüencial. Moriarty (1997) compara métodos evolucionários e métodos tradicionais de aprendizado por reforço e mostra que métodos evolucionários têm vantagem devido ao mecanismo de atribuição de crédito mais robusto. Muitas outras pesquisas (BELEW, 1993; NOLFI, 1994) demonstraram que a neuro-evolução, ou a evolução simulada de redes neurais, é uma estratégia efetiva para resolver problemas de decisão seqüencial.

Algoritmos evolucionários (AE) são adequados para tarefas de decisão seqüencial porque eles trabalham naturalmente com reforço esparsos e buscam por uma solução de forma global, realizando poucas suposições sobre o domínio de solução.

2.4 Aprendizado por Reforço

O aprendizado por reforço (AR) é o problema de um agente aprender seu comportamento através de interações de tentativa e erro em um ambiente dinâmico. No AR, o agente tomador de decisão recebe respostas do domínio na forma de sinais de reforço. Tais sinais fornecem somente uma medida geral da proficiência na tarefa e não direcionam explicitamente o agente para qualquer curso de ação. Ao contrário da maioria das formas de aprendizagem de máquina, não há nada dizendo ao agente quais devem ser suas ações.

Neste trabalho, foi adotada a abordagem de alguns pesquisadores (KAELBLING, 1996; SUTTON, 1998) que definem aprendizado por reforço como uma classe de problemas e não como um conjunto de técnicas. Por outro lado, qualquer método que seja adequado para resolver tais problemas, é considerado um método de aprendizagem por reforço.

A principal diferença entre o aprendizado por reforço e o problema mais largamente estudado de aprendizado supervisionado é que, no aprendizado por reforço, não há nenhuma apresentação de pares de entrada/saída, ou seja, não há exemplos de comportamento fornecidos por um supervisor externo. Em aprendizado supervisionado, o agente acessa exemplos do comportamento correto e aprende através dos erros entre

suas decisões e as decisões corretas conhecidas. Na aprendizagem por reforço, o curso correto da ação não é conhecido. O agente precisa aprender o comportamento bom através de tentativa e erro, interagindo diretamente com o ambiente. O agente escolhe uma ação, recebe uma recompensa por esta ação e conhece o estado subsequente, mas, não sabe qual é a ação que teria atendido melhor seus interesses a longo prazo. É necessário que o agente reúna experiência útil sobre os estados possíveis do sistema, ações, transições e recompensas para agir de forma ótima.

Embora o aprendizado supervisionado seja um tipo de aprendizagem importante, em determinados ambientes, é frequentemente impraticável obter exemplos corretos e representativos do comportamento desejado do agente para todas as situações possíveis. Em ambientes dinâmicos e complexos, um agente precisa ser capaz de aprender a partir de sua própria experiência.

No modelo padrão de aprendizado por reforço, um agente é ligado a seu ambiente via percepção e ação. Conforme retratado na Figura 2.1, em cada passo de interação, o agente recebe como entrada alguma indicação do estado atual, s , do ambiente; o agente então escolhe uma ação, a , para gerar uma saída. A ação muda o estado do ambiente, e o valor desta transição de estado é informado ao agente através de um sinal de reforço r . O comportamento B do agente deve levar a ações que tendem a aumentar a soma de seus sinais de reforço ao longo da execução. O agente pode aprender a fazer isso através de sistemática tentativa e erro.

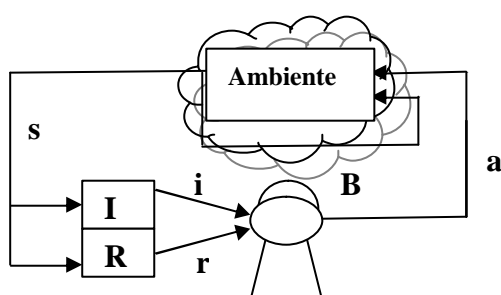


Figura 2.1: Modelo padrão do aprendizado.

Formalmente, o modelo consiste de:

- Um conjunto discreto de estados do ambiente, S ;
- Um conjunto discreto de ações do agente, A ;
- Um conjunto escalar de sinais de reforço; tipicamente $\{0,1\}$, ou números reais.

A figura também inclui uma função de entrada I , que determina como o agente vê o estado do ambiente; o agente pode perceber o estado exato ou parcial do ambiente.

O trabalho do agente é achar uma política p , que mapeie estados para ações, maximizando a medida de reforço ao longo do tempo. Espera-se, em geral, que o ambiente seja não-determinístico; isto é, tomando-se a mesma ação no mesmo estado em duas ocasiões diferentes podem resultar diferentes próximos estados e/ou em valores de reforço diferentes. Entretanto, supõe-se um ambiente estacionário; isto é, as probabilidades de fazer transições de estado recebendo sinais específicos de reforço não mudam com o tempo.

Quando o reforço está disponível somente ocasionalmente, o reforço é dito esparso. Reforço esparso significa que uma solução proposta terá de ser avaliada várias vezes antes de qualquer informação de retorno se tornar disponível. Jogos, como o xadrez, são os principais exemplos de domínios esparsos: muitos movimentos precisam ser realizados antes do sinal de perda ou ganho estar disponível.

2.4.1 *Exploration e exploitation*

De acordo com Sutton (1998), um dos maiores desafios que surgem para métodos de aprendizado por reforço e que não existem em outros tipos de aprendizado é o de balancear *exploration e exploitation*.

A única forma de obter nova informação sobre o espaço de soluções é gerar uma nova solução candidata e avaliá-la. A idéia central do algoritmo da subida da encosta é que valores de aptidão mais altos estão próximos dos pontos já avaliados como bons. Isto é, escolhe-se um ponto inicial arbitrário e investiga-se os elementos adjacentes a ele no espaço de busca. Se algum desses pontos possuir um valor maior, então esse ponto passa a ser o novo referencial (BARRETO, 2003). Se o espaço de valores de aptidão é monotônico (possui somente um pico) então a busca próxima da melhor solução atual irá sempre levar ao pico. A busca próxima de regiões conhecidas como *bias* é chamada *exploitation*, pois, explora o conhecimento corrente. A menos que o espaço de soluções seja completamente caótico, pontos próximos de um ponto conhecido como bom têm mais chance de produzir uma melhora na solução que uma tentativa aleatória.

Contudo, problemas interessantes são multimodais. Eles possuem muitos picos, chamados ótimos locais, alguns dos quais estão bastante distantes do verdadeiro ponto ótimo global. Técnicas estritas de subida na encosta, mesmo fazendo uso sofisticado da informação de gradiente local, estão pré-dispostas a convergir para um ponto ótimo local, simplesmente pelo fato deles serem maioria. Esta é a marca registrada de uma técnica *super-exploitative*. Uma forma de resolver este problema é re-iniciar em uma posição aleatória quando uma convergência é detectada. Um re-início aleatório é um movimento *exploratory*. Subidas da encosta são puramente *exploitative*, logo, são indicadas somente para descobrir ótimos locais.

Existe uma tensão inerente entre *exploitation e exploration* em qualquer método de aprendizagem de máquina gerado e testado. Para cada teste de uma solução é necessário decidir se explorar um novo território ou buscar por uma solução perto das regiões conhecidas como boas. Explorar uma nova região pode revelar soluções promissoras, mas tentativas aleatórias contínuas são tão ruins quanto enumeração exaustiva. Técnicas ideais devem continuamente balancear *exploitation e exploration*. Na seção 2.5, é citado como algoritmos genéticos realizam o balanceamento entre a busca em regiões conhecidas como boas e regiões ainda desconhecidas.

2.4.2 Estratégias de Aprendizado por Reforço

Há duas estratégias principais para resolver problemas de aprendizado por reforço. A primeiro é a busca no espaço de comportamentos para achar um que execute bem no ambiente. Esta estratégia é geralmente implementada com algoritmos genéticos e programação genética. A segunda abordagem consiste em três classes fundamentais de métodos: programação dinâmica, métodos de Monte Carlo e diferença temporal (SUTTON, 1998).

Métodos de programação dinâmica são bem desenvolvidos matematicamente, mas, requerem um modelo completo e preciso do ambiente. Métodos de Monte Carlo não precisam de um modelo e são conceitualmente simples, mas, a depender do tamanho dos espaços de ações e de estados, a convergência pode ser bastante lenta. Finalmente, métodos de diferença temporal não precisam de modelo e são totalmente incrementais, mas são mais complexos de analisar. Estes métodos também diferem em eficiência e velocidade de convergência. Para maiores detalhes, consulte Sutton (1998).

O termo programação dinâmica (DP) se refere a uma coleção de algoritmos que podem ser usados para computar políticas ótimas, desde que seja fornecido um modelo perfeito do ambiente como um processo de decisão de Markov (MDP). Algoritmos clássicos de DP possuem utilidade limitada no aprendizado por reforço principalmente por dois motivos: supõem um modelo perfeito do mundo e possuem alto custo computacional.

Métodos de Monte Carlo não assumem o conhecimento completo do ambiente; ao invés disso, eles precisam de experiência, ou seja, exemplos de estados, ações e recompensas de interações do agente com o ambiente. A maior desvantagem do método de Monte Carlo é a convergência lenta. Embora, a convergência pareça inevitável, ela não possui uma prova formal. Segundo Sutton (1998), esta é uma das questões mais importantes no aprendizado por reforço que ainda permanece em aberto.

Aprendizado por diferenças temporais é uma combinação das idéias de Monte Carlo e programação dinâmica. Da mesma forma que métodos de Monte Carlo, métodos de TD podem aprender diretamente da experiência sem um modelo das dinâmicas do ambiente. Como DP, métodos de TD podem atualizar estimativas baseados, em parte, nas estimativas aprendidas, sem esperar por uma saída final.

Segundo Moriarty (1997), TD é o método de aprendizado por reforço mais popular. Em TD, uma função valor prediz o retorno esperado do ambiente dado o estado corrente do mundo e a política de decisão corrente. Se a função valor for precisa, o agente pode basear todas as suas decisões nos valores previstos para os estados subsequentes do mundo. Em outras palavras, ao selecionar a próxima decisão, o agente considera o efeito daquela decisão através do exame do valor esperado da transição de estado causada por aquela decisão.

A função valor ótima é alcançada usando uma versão do algoritmo de aprendizagem TD(λ). TD(λ) usa observações de diferenças de previsão de estados consecutivos para aprender previsões de valores corretos. Suponha que dois estados consecutivos i e j retornem valores de previsão de recompensa 5 e 2, respectivamente. A diferença sugere que a recompensa do estado i pode ter sido superestimada e deve ser reduzida para concordar com a previsão do estado j . A atualização do valor da função V é realizada usando a seguinte regra:

$$V(i) = V(i) + \alpha((V(j) - V(i)) + R(i)) \quad (2.1)$$

onde α representa a taxa de aprendizagem e R a recompensa imediata. Logo, a diferença na previsão ($V(j) - V(i)$) de estados consecutivos é usada como uma medida de previsão de erro. Pode-se imaginar um longo caminho de valores de previsão $V(0) \dots V(n)$ de transições de estados consecutivos com o último estado $V(n)$ contendo a verdadeira

recompensa do ambiente. Os valores de cada estado são ajustados de forma que eles concorram com seus sucessores e eventualmente com a recompensa verdadeira em $V(n)$. Em outras palavras, a recompensa verdadeira é propagada de volta através do caminho das previsões de valores. O resultado da rede é uma função valor precisa que pode ser usada para acessar a utilidade das decisões comparando valores de transições de estados subsequentes.

As duas implementações mais proeminentes de TD são *Adaptive Heuristic Critic* (AHC)(BARTO, 1983) e *Q-learning* (WATKINS, 1989).

2.4.2.1 Adaptive Heuristic Critic

Um dos primeiros métodos de aprendizagem por reforço que usou a aprendizagem TD é o AHC. No AHC, uma função valor TD, chamada ‘crítica’, é treinada para prever o desempenho de um segundo agente que é responsável por gerar as decisões. A função crítica usa a equação 2.1 para aprender as previsões dos valores, dado as transições de estado causadas pelo agente de decisão. O agente de decisão simultaneamente atualiza sua política de decisão para maximizar o valor recebido da função crítica. Por exemplo, se o agente de decisão recebe um valor baixo da função crítica após tomar uma decisão, ele deveria reduzir a probabilidade de tomar aquela decisão na mesma situação. Desde que o agente de decisão recebe um retorno constante da função crítica, modificações da política podem ser feitas através de diferentes métodos de subida da encosta. A abordagem mais comum é usar uma variante do método *backpropagation*.

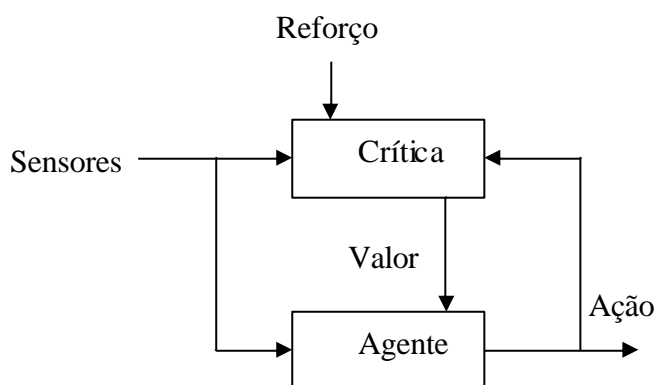


Figura 2.2: Visão Geral da AHC

2.4.2.2 Q-learning

Q-learning é a abordagem de aprendizado por reforço mais utilizada. Q-learning combina os agentes de crítica e decisão em uma única função chamada Q-função. A Q-função mapeia decisões e estados do mundo em estimativas de recompensa esperadas. Em outras palavras, a Q-função $Q(d,i)$ representa a utilidade de tomar uma decisão específica d no estado i . Dados valores Q-função precisos, chamados Q-valores, uma política ótima seleciona para cada estado a decisão com o mais alto valor Q associado (recompensa esperada).

A função-Q é aprendida através da seguinte equação de atualização TD:

$$Q(d,i) = Q(d,i) + \alpha(R(i) + \max_{d'} Q(d',i') - Q(d,i)) \quad (2.2)$$

onde d' é a próxima decisão e i' é o próximo estado. Essencialmente, esta equação atualiza $Q(d,i)$ baseada na recompensa corrente e a recompensa prevista se todas as decisões futuras são selecionadas otimamente. Watkins e Dayan (1992) provaram que se as atualizações são desempenhadas desta maneira, a Q -função irá convergir para os valores Q ótimos. O sistema de aprendizagem por reforço pode logo usar os Q valores para avaliar cada decisão que é possível a partir de um dado estado. A decisão que retorna o maior Q -valor é a escolha ótima.

2.4.2.3 Abordagem Evolucionária

Algoritmos evolucionários fornecem uma ferramenta de treinamento geral em que poucas suposições sobre o domínio são necessárias. Desde que algoritmos evolucionários somente precisam de uma única função de avaliação (aptidão) sobre toda a tarefa (possivelmente composta de vários passos), eles são capazes de aprender em domínios com reforços esparsos, o que os torna bem adequados para avaliar o desempenho de tarefas de decisão. Não é preciso exemplos do comportamento correto. O algoritmo evolucionário busca pelas estratégias de decisão mais produtivas usando somente recompensas raras retornadas por um sistema subjacente. Juntos, algoritmos evolucionários e redes neurais oferecem uma abordagem promissora para o aprendizado e aplicação de estratégias efetivas de decisão em várias situações diferentes.

Algoritmos evolucionários são técnicas de busca globais, inspiradas na teoria de Darwin da evolução natural. Soluções potenciais são codificadas em estruturas chamadas cromossomos. Durante cada iteração, o AE avalia soluções e gera filhos baseado na aptidão de cada solução na tarefa. Subestruturas, ou genes, das soluções são então modificados através de operações genéticas tais como mutação e recombinação. A idéia é que estruturas que levam a boas soluções em avaliações anteriores podem sofrer mutação ou serem combinadas para formar soluções melhores em avaliações posteriores.

No aprendizado por reforço evolucionário, as soluções tomam a forma de agentes tomadores de decisão que operam em ambientes dinâmicos. Agentes são localizados no mundo onde tomam decisões em resposta às condições ambientais. O AE seleciona o agente baseado no seu desempenho na tarefa, e aplica operadores genéticos para gerar novos tipos de agentes. Desde que algoritmos evolucionários requisitam somente uma única avaliação de aptidão sobre a tarefa inteira do agente (normalmente envolvendo vários passos), eles encontram soluções efetivas em domínios que retornam somente reforços ocasionais sobre uma seqüência de ações. O único retorno requerido do ambiente é uma medida geral da proficiência de cada agente.

Como métodos TD, o aprendizado por reforço evolucionário é uma abordagem livre de modelos, porque não requisita um modelo de simulação ou conhecimento das regras de transição de estado para formar suas políticas.

Segundo Kaelbling (1996), não é bem claro qual das abordagens, métodos tradicionais ou métodos evolucionários de aprendizado por reforço evolucionário, é melhor em que circunstâncias. Contudo, neste trabalho, optou-se pela utilização da neuro-evolução, uma das abordagens evolucionárias existentes, por algumas razões:

primeiro porque em ambientes dinâmicos e complexos, é impraticável criar um modelo preciso do mundo e algoritmos evolucionários não precisam de um modelo. Segundo porque redes neurais possuem uma boa generalização e suportam algumas variações no mundo. Terceiro, o armazenamento das políticas de decisão nas redes neurais é econômico, a função de transição é representada por pesos nas conexões da rede, isso significa que mesmo aumentando-se os espaços de ações e estados do mundo, a quantidade de informação armazenada não será alterada.

No capítulo 3, aborda-se a neuro-evolução em maiores detalhes. Nas duas próximas seções, é fornecida uma visão geral de algoritmos genéticos e redes neurais, que constituem os centros de apoio da neuro-evolução.

2.5 Algoritmos Genéticos

Algoritmos genéticos (AG) são métodos de busca e otimização inspirados nos processos da evolução natural dos seres vivos. Foram introduzidos por Holland (1975) e popularizados por um de seus alunos, David Goldberg (1989). Estes algoritmos seguem o princípio da seleção natural e sobrevivência do mais apto declarado, em 1859, pelo naturalista e fisiologista inglês Charles Darwin em seu livro **A origem das espécies**. De acordo com Darwin, “Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes”.

Na natureza, os indivíduos mais adaptados ao ambiente irão vencer a competição por recursos limitados. A capacidade de sobrevivência do indivíduo é determinada por várias características dele. As características de cada indivíduo são, por sua vez, determinadas pelo conteúdo genético do indivíduo. O conjunto de genes que controla estas características forma os cromossomos, que são a chave para a sobrevivência do indivíduo em um ambiente competitivo. A seleção natural e a recombinação do material genético durante a reprodução constituem a força motora da evolução.

Desde que somente os indivíduos com maiores aptidões sobrevivem e se reproduzem, os genes mais fracos irão desaparecer gradualmente. Se o ambiente não sofrer alterações, o processo de evolução levará a um estado onde todos os indivíduos serão constituídos dos melhores genes.

Inspirados neste processo de evolução natural, o uso de analogias do comportamento natural levou ao desenvolvimento dos chamados algoritmos evolucionários. Estes algoritmos são compostos de quatro elementos: uma estrutura de codificação que será replicada, operadores que afetam os indivíduos da população, uma função de aptidão que indica quão bom é um indivíduo, e um mecanismo de seleção. Algoritmos genéticos são um dos principais paradigmas dentro dos algoritmos evolucionários. Eles operam em uma população de indivíduos, onde cada indivíduo representa uma possível solução para o problema. Cada indivíduo recebe uma aptidão baseado na função de aptidão. Um mecanismo de seleção escolhe os indivíduos mais aptos para reprodução. A reprodução ocorre através das técnicas de recombinação e mutação.

Não existem garantias de que AG encontrem o ótimo global, mas eles são geralmente bons em encontrar uma solução aceitável dentro de um tempo razoável³.

³ O tempo necessário para busca irá variar dependendo de fatores como, por exemplo, a complexidade da tarefa e a codificação do cromossomo. Contudo, diversos autores

O primeiro passo de um algoritmo genético típico é a geração de uma população inicial de cromossomos. Esta população é formada por um conjunto aleatório de cromossomos que representam possíveis soluções do problema a ser resolvido. Durante o processo evolutivo, esta população é avaliada e cada cromossomo recebe um valor de aptidão, refletindo quão bem ele resolve o problema. Os cromossomos mais aptos são selecionados e podem sofrer modificações através dos operadores de *recombinação* e *mutação*, gerando descendentes para a próxima geração.

Holland (1975) foi a primeira pessoa que propôs programas de computador inspirados nos processos evolucionários da natureza. Este algoritmo genético é comumente chamado de Algoritmo Genético Simples, vide Figura 2.3.

Seja $S(t)$ a população de cromossomos na geração t :

```
t := 0
Inicializar S(t)
Avaliar S(t)
Faça geracao := 1 para n
  t := geracao
  Selecione S(t) a partir de S(t-1)
  Recombinação(S(t))
  Mutação(S(t))
  Avalie(S(t))
Fim faça
```

Figura 2.3: Algoritmo Genético Simples.

AGs operam em representações codificadas das soluções, assim como os cromossomos dos indivíduos na natureza. É assumido que uma solução potencial pode ser bem representada como um conjunto de parâmetros e codificada como um cromossomo. No Algoritmo genético Simples, Holland (1975) codificou as soluções como cadeias de *bits* de um alfabeto binário.

Cada indivíduo é associado a um valor de aptidão, retornado pela função de aptidão, que reflete quão bom ele é. A seleção modela o mecanismo de sobrevivência do mais apto. O AG simples usa o algoritmo de seleção da roleta para realizar a seleção proporcional.

A idéia do método da Roleta é que indivíduos de uma geração sejam escolhidos para fazer parte da próxima geração, através de um sorteio de roleta. Os indivíduos são representados na roleta proporcionalmente ao seu índice de aptidão. Finalmente, a roleta é girada um determinado número de vezes, dependendo do tamanho da população, e são escolhidos como indivíduos que participarão da próxima geração, aqueles sorteados na roleta.

mostram que a abordagem evolucionária é bastante eficiente quando comparada aos métodos de busca tradicionais (MORIARTY 1997).

Computacionalmente, o método da roleta pode ser implementado da seguinte maneira: ordena-se os indivíduos da população por ordem ascendente de aptidão, calcula-se a soma das aptidões de todos os indivíduos da população. Em seguida, gera-se um número aleatório r (tirado de uma distribuição uniforme) no intervalo $[0, \text{total}]$, onde total é a soma de todas as aptidões. Por fim, o cromossomo selecionado é o primeiro cromossomo que possui um total parcial maior que r , onde total parcial é a soma de sua aptidão com as aptidões dos cromossomos anteriores. A Figura 2.4 ilustra o algoritmo da roleta.

```

Seja  $f_i$  o valor de aptidão do cromossomo  $c_i$ .
    n
total :=  $\sum_{i=1}^n f_i$ 
r := random(0,total)
totalParcial := 0
i := 0
Repita
    i := i+1
    totalParcial := totalParcial +  $f_i$ 
Até totalParcial  $\geq r$ 
Retornar cromossomo  $c_i$ 

```

Figura 2.4: Algoritmo de Seleção da Roleta.

Outro operador de seleção que pode ser utilizado é a seleção por torneio. Na seleção por torneio, n cromossomos são escolhidos aleatoriamente e destes, o cromossomo com maior aptidão é selecionado para reprodução. O processo se repete até que a quantidade de cromossomos selecionados para reprodução seja igual ao tamanho da população. Geralmente, utiliza-se o valor de n igual a 3.

A fase de reprodução do AG é realizada através do mecanismo de recombinação. O método mais simples de recombinação é escolher pares de cromossomos selecionados, em seguida, dividir os cromossomos de cada par em alguma posição escolhida aleatoriamente, e trocar seus segmentos, este mecanismo é conhecido como recombinação de 1-ponto. Uma outra operação, chamada mutação, causa alteração esporádica e aleatória dos *bits* das cadeias, que é uma analogia direta da natureza e faz o papel de re-gerar material genético perdido. A mutação é aplicada aos indivíduos após a recombinação. Um parâmetro, taxa de mutação, dá a probabilidade de um *bit* ser alterado.

Uma opção frequentemente usada em AG é o elitismo. Esta opção permite que os melhores cromossomos sejam propagados para a geração seguinte sem sofrer recombinação ou mutação. Isto garante que o AG terá uma convergência monotônica.

2.5.1 *Exploitation* x *Exploration* em AG

A batalha entre *exploitation* e *exploration* em AG é manifestado principalmente pela pressão de seleção na computação evolucionária. Pressão de seleção é simplesmente uma medida de quão fortemente o princípio Darwiniano de “sobrevivência do mais

apto” é aplicado. Baixa pressão de seleção significa que mesmo indivíduos com baixa aptidão podem reproduzir. Alta pressão de seleção significa que indivíduos com alta aptidão possuem uma chance maior de gerar descendentes que os indivíduos com baixa aptidão. Escolher um indivíduo com alta aptidão para procriar é *exploitative*, enquanto que o casamento de indivíduos com baixa aptidão é *exploratory*.

O balanceamento entre *exploitation* e *exploration* também pode ser realizado por mutação. A mutação adiciona novo material genético à população. Uma alta taxa de mutação será mais *exploratory*.

2.6 Redes Neurais

As redes neurais artificiais são técnicas computacionais que apresentam um modelo inspirado na estrutura neural de organismos inteligentes.

Hoje em dia a maior parte dos pesquisadores concorda que as redes neurais são muito diferentes do cérebro em termos de estrutura. No entanto, como o cérebro, uma rede neural é uma coleção massivamente paralela de unidades de processamento pequenas e simples inter-conectadas. A inteligência da rede é “armazenada” nos pesos destas conexões. Entretanto, em termos de escala, o cérebro é muito maior que qualquer rede neural. Além disso, as unidades usadas na rede neural são tipicamente muito mais simples que os neurônios biológicos e o processo de aprendizado do cérebro (embora ainda desconhecido) é, provavelmente, muito diferente do das redes neurais.

Atualmente, Redes Neurais Artificiais é um dos paradigmas mais indicados para o projeto e análise de sistemas inteligentes adaptativos em uma grande faixa de aplicações em inteligência artificial e modelagem cognitiva, por várias razões: redes neurais podem representar qualquer função computável (HERTZ, 1991), possuem potencial para computação paralela massiva, robustez na presença de ruídos, adaptabilidade na falha de componentes, e generalizam a solução para entradas não previamente testadas.

A primeira vantagem da utilização de redes neurais é o eficiente mecanismo de armazenamento do que foi aprendido. No contexto das tarefas de decisão seqüencial, o mapeamento de estado do mundo a ações é representada de forma distribuída nos pesos e conexões da rede. A medida que novos estados e ações são observados, essa nova informação é distribuída e unida de forma efetiva à informação antiga nos pesos. A estrutura de armazenamento não cresce de forma incontrolável enquanto o agente ganha mais experiência, mas, permanece constante. Redes Neurais são uma forma de armazenamento compacto do que foi aprendido que pode cobrir um espaço enorme de situações de entrada.

Além do armazenamento constante, redes neurais fornecem tempo computacional constante. A complexidade computacional é limitada pelo número de neurônios e conexões dentro da rede. Desde que estes componentes permaneçam constantes, o tempo de computação também permanecerá constante. Esta característica é ainda mais importante em tarefas de decisão de tempo real, onde o tempo gasto para gerar uma decisão pode diminuir o desempenho do sistema. Também, desde que redes neurais são constituídas de muitos elementos computacionais separados, eles podem ser facilmente paralelizados a fim de acelerar a computação.

Porém, talvez a vantagem mais importante de uma representação neural seja a generalização efetiva do que foi aprendido. Ou seja, a capacidade de dar respostas coerentes para dados não apresentados a ela durante o treinamento. Além disso, uma

vez que o espaço de armazenamento é finito, a rede neural precisa consolidar um conhecimento geral baseado em características ou faixas de valores do espaço de entrada em vez de valores de entrada exatos. A generalização é importante em espaços grandes de estado onde um agente não pode realisticamente experimentar todas as situações possíveis do domínio durante o treinamento. Ao generalizar o comportamento, a rede neural pode aplicar decisões para estados não experimentados baseada em características comuns com os estados experimentados.

Adicionalmente, Mcquesten (2002) coloca que, Redes Neurais Artificiais são adequadas para a solução de problemas complexos, pois, podem reconhecer padrões em entradas complexas, e problemas substanciais possuem entradas complexas.

As vantagens são bastante atrativas, mas, é também importante considerar as desvantagens de uma política de decisão expressa em rede neural. Primeiro, existem numerosos parâmetros que devem ser conhecidos *a priori* para garantir um bom comportamento. A arquitetura de rede, o número de neurônios, a função de ativação, são três exemplos de parâmetros para os quais os valores ótimos não são bem compreendidos. A implementação envolve várias tentativas e erros para gerar configurações de parâmetros efetivos. Segundo, desde que a política de decisão é representada nas conexões e pesos, é muito difícil extrair a política de forma mais legível e compreensível. Uma representação lúcida pode ser necessária se for necessário implementar a política de decisão de uma outra maneira ou mesmo para compreender um sistema subjacente. Tradicionalmente, desenvolvedores de redes neurais aceitam sem questionamentos o comportamento final da rede, uma vez que o raciocínio por trás deste comportamento é de difícil extração.

3 NEURO-EVOLUÇÃO

Em tarefas de aprendizagem no mundo real tais como controlar robôs, jogar, perseguir ou escapar de um inimigo, não existe uma forma de especificar as ações corretas para cada situação. Muitas vezes, a ação correta não é conhecida. Para tais problemas, o comportamento ótimo precisa ser aprendido através da exploração de diferentes ações e boas decisões precisam ser recompensadas baseadas num sinal de reforço esparso.

Neuro-evolução é um método de aprendizado por reforço, no qual a busca pelo comportamento ideal é implementada através do aprendizado de redes neurais por algoritmos genéticos. Na neuro-evolução, cromossomos representam parâmetros das redes neurais, que podem ser, por exemplo, pesos, limites, e conectividade. Estes cromossomos são recombinados baseados no princípio de seleção natural com o objetivo de se encontrar uma rede neural satisfatória para um dado problema.

Comparado aos métodos de aprendizagem por reforço tradicionais, a neuro-evolução é mais robusta na presença de ruídos e entradas incompletas, e permite representar estados contínuos e ações naturalmente.

Embora a pesquisa que vem sendo realizada na área de Redes Neurais tenha levado à descoberta de vários resultados teóricos e empíricos e ao desenvolvimento de várias aplicações práticas nas últimas décadas, o projeto de redes neurais para aplicações específicas ainda é um processo de tentativa e erro, onde, geralmente se leva em consideração a experiência passada em aplicações similares.

O desempenho de redes neurais em problemas particulares é criticamente dependente, entre outras coisas, do número de neurônios, da arquitetura e do algoritmo de aprendizagem utilizado (MORIARTY, 1997). Estes fatores tornam o processo de projeto de redes neurais uma tarefa difícil. A falta de bons princípios para projeto constitui o maior obstáculo no desenvolvimento de sistemas de redes neurais em larga escala para uma variedade de problemas práticos.

Algoritmos evolucionários, por sua vez, oferecem uma abordagem aleatória, relativamente eficiente, para busca por soluções quase ótimas em uma variedade de domínios de problemas. O projeto de redes neurais eficientes para classes específicas de problemas é, portanto, um candidato natural para a aplicação de algoritmos evolucionários.

3.1 A abordagem evolucionária

Os processos chaves na abordagem evolucionária para o projeto de arquiteturas neurais são ilustrados na Figura 3.1:

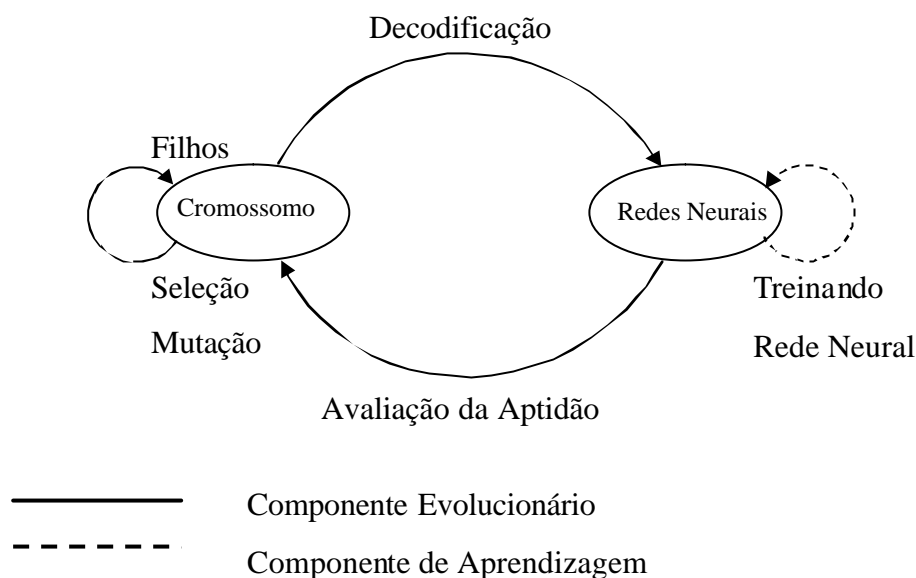


Figura 3.1: Processos da abordagem evolucionária.

Algoritmos evolucionários são modelados, de forma imprecisa, como os processos que parecem ocorrer na evolução biológica. A ideia central de sistemas evolucionários é a de uma população de cromossomos que são elementos de um espaço de busca multidimensional. Em um algoritmo genético simples, cromossomos são, por exemplo, *strings* binárias de um tamanho fixo (n) que codificam os pontos de um espaço de busca booleano n -dimensional.

No contexto deste trabalho, o cromossomo codifica uma classe de atributos de arquiteturas neurais. O processo de codificação/decodificação de redes em cromossomos pode ser extremamente simples ou muito complexo. As redes neurais resultantes podem também ser equipadas com algoritmos de aprendizagem que as treinam usando o estímulo do ambiente ou simplesmente avaliando uma sua execução em uma dada tarefa; os pesos da rede também são determinados pelo mecanismo de codificação/decodificação. A avaliação de desempenho de uma rede neural determina a aptidão do(s) cromossomo(s) correspondente(s).

O procedimento neuro-evolucionário trabalha em uma população de cromossomos, preferencialmente selecionando e reproduzindo aqueles que codificam redes neurais com melhores aptidões. Operadores genéticos, tais como mutação, recombinação, inversão, etc..., são usados para introduzir diversidade na população. Logo, após várias gerações, a população evolui gradualmente para cromossomos que correspondem às melhores redes neurais.

Segundo Moriarty (1997), a combinação de redes neurais e algoritmos genéticos oferece algumas vantagens importantes sobre a maioria dos métodos de aprendizado de redes neurais tradicionais, tais como, *backpropagation* (RUMELHART, 1986) e *cascade correlation* (FAHLMAN, 1990). Nas seções seguintes, são destacados dois dos principais benefícios: treinamento esparsos e tempo de treinamento.

3.1.1 Treinamento esparso

A motivação primária para utilização da neuro-evolução sobre as técnicas mais usuais, tais como a *backpropagation*, é a habilidade de se treinar agentes com reforços mais esparsos (BALAKRISHNAN, 1995). Para muitas tarefas de decisão seqüencial, como, por exemplo, tomar decisões táticas em um jogo, freqüentemente não existe um retorno automático para a avaliação das decisões mais recentes. Em muitos domínios, computar a informação de gradiente sobre cada saída da rede é inviável porque o reforço do sistema não é tão freqüente. Por exemplo, em tarefas de decisão seqüencial, um sinal de reforço pode somente ser dado após a tomada de uma seqüência de decisões. Estas atribuições de crédito são difíceis de serem feitas baseadas em uma métrica de desempenho geral, porque não é sempre óbvio como decisões isoladas afetam a saída final. Uma vez que algoritmos genéticos não requerem atribuições de crédito explícitas para saídas individuais da rede, eles podem resolver uma faixa de problemas, incluindo problemas de reforço esparso.

3.1.2 Tempo de treinamento

Algumas pesquisas mostram que a neuro-evolução é competitiva em tempo de treinamento com os métodos tradicionais de treinamento de redes neurais. Montana (1989) rodou vários experimentos comparando redes neurais evolucionárias com redes *backpropagation* tradicionais na tarefa de classificação de dados. Duas implementações diferentes de redes neurais foram usadas: uma onde os pesos de uma rede *feed-forward* evoluíam usando um único algoritmo evolucionário, e uma onde um algoritmo evolucionário foi implementado com *backpropagation* em uma abordagem híbrida. Montana (1989) não reportou nenhuma vantagem clara em usar a abordagem híbrida sobre a evolucionária isolada. Contudo, reportou resultados superiores no número de iterações de treinamento usando algoritmo evolucionário sobre *backpropagation*.

3.2 Codificação da Rede

Ao se optar pela utilização de redes neurais, uma das questões principais é como será realizada a codificação da rede em cromossomos. A maioria das abordagens de neuro-evolução fixa a arquitetura a ser evoluída e o cromossomo meramente reflete a concatenação de pesos da rede. Fixar a arquitetura e forçar os pesos a corresponderem diretamente a sua localização no cromossomo inibe muito da flexibilidade da abordagem de algoritmos evolucionários (MORIARTY, 1997). Por exemplo, é muito difícil construir estruturas de pesos altamente efetivos se os pesos estão localizados em regiões distantes do cromossomo. Ao fixar os pesos, uma tendência (*bias*) é introduzida considerando que pesos serão combinados em blocos de construção úteis: pesos que ficam próximos um do outro. Tal tendência restringe a liberdade do algoritmo evolucionário para explorar muitos blocos de construção e pode significativamente aumentar o tempo de busca.

A questão de como uma arquitetura neural é representada é crítica. A representação ou codificação usada não somente determina as classes de arquiteturas neurais que poderão possivelmente evoluir, como também podem restringir a escolha do processo de decodificação. Por exemplo, se o problema requer a descoberta de redes neurais com uma estrutura recorrente, de forma a garantir uma probabilidade de sucesso diferente de zero, o esquema de codificação precisa ser bastante preciso para descrever redes neurais

recorrentes, e o mecanismo de decodificação precisa ser capaz de transformar tal descrição em uma rede recorrente apropriada.

3.3 Topologia da Rede

O sucesso de uma arquitetura neural para resolver um problema particular (ou uma classe de problemas) depende criticamente da topologia da rede. Por exemplo, uma rede neural puramente *feed-forward* é incapaz de descobrir ou responder a dependências temporais no seu ambiente; para esta tarefa, seria necessária uma rede recorrente. Similarmente, decisões não-lineares não podem ser descobertas por redes de apenas uma camada; são necessários *perceptrons* multi-camada.

Topologias de redes neurais podem ser classificadas basicamente em dois tipos: redes *feed-forwards* e redes recorrentes.

Cada um dos tipos básicos de topologia pode ainda ser classificado como redes multi-camadas, conectadas aleatoriamente, esparsamente conectadas, regular, irregular, modular, hierárquica, etc...

3.4 Variáveis da Evolução

Redes neurais são tipicamente especificadas em termos de topologia (ou padrão de conectividade), funções computadas pelos neurônios (*sigmoid*, *threshold*,...) e os pesos da conexão (ou, um algoritmo de aprendizagem que atribua valores a esses pesos). Uma descrição mais completa de uma arquitetura neural requer a especificação das estruturas de controle e aprendizagem. Virtualmente, qualquer subconjunto destas variáveis é candidato a ser operado por processos evolucionários. Por exemplo, um sistema A pode evoluir a conectividade da rede tanto quanto os pesos (enquanto mantém todo o resto constante). Similarmente, um sistema B pode evoluir somente a conectividade, confiando talvez em uma busca local mais eficiente por pesos dentro de cada rede. O tempo/desempenho para os dois sistemas, em um dado problema, será diferente, tornando a escolha das variáveis, sujeitas à evolução, um fator extremamente crítico.

Em adição à conectividade da rede e os pesos, é possível evoluir o algoritmo de aprendizagem, as funções de controle ou reguladoras, as funções computadas por vários neurônios, a distribuição de diferentes tipos de neurônio, densidades relativas de conexões, parâmetros (e/ou processos) governando a decodificação de um genótipo em um fenótipo, e assim sucessivamente.

3.5 Métodos de Neuro-Evolução

Nesta seção, são descritos alguns dos métodos de neuro-evolução existentes na literatura. Na seção 3.5.1, descrevemos o *Symbiotic, Adaptive Neuro-Evolution* (SANE), um método de neuro-evolução que evolui os neurônios da camada oculta da rede neural, cada neurônio codifica os pesos das conexões dele com os outros neurônios aos quais está conectado. Na seção 3.5.2, o *Enforced Subpopulations* (ESP) é descrito. O ESP é uma extensão do SANE, e é a base do modelo proposto neste trabalho. Assim como o SANE, o ESP também considera redes neurais com topologias fixas. Por último, na seção 3.5.3, apresenta-se o *NeuroEvolution of Augmenting Topologies* (NEAT), que ao contrário de outros métodos, como o SANE e o ESP, evolui não só os pesos das conexões da rede, como também sua topologia.

3.5.1 *Symbiotic, Adaptive Neuro-Evolution*

Symbiotic, Adaptive Neuro-Evolution (MORIARTY, 1996; MORIARTY, 1997), ou SANE, é um método de neuro-evolução que evolui uma população de neurônios que se interconectam para formar uma rede neural completa. Mais especificamente, SANE evolui uma população de neurônios da camada oculta de uma rede para um dado tipo de arquitetura tal como uma rede de duas camadas.

Os passos básicos do SANE podem ser descritos na Figura 3.2. Durante o estágio de evolução, subpopulações aleatórias de neurônios de tamanho **S** são selecionados e combinados para formar uma rede neural. A rede é avaliada na tarefa e recebe um crédito, que é subsequentemente adicionado à variável de aptidão de cada neurônio. O processo continua até que cada neurônio tenha participado em um número suficiente de redes. A aptidão média de cada neurônio é calculada dividindo-se a soma de suas aptidões pelo número de redes em que ele participou. Os neurônios que possuem uma aptidão mais alta cooperaram melhor que os outros neurônios da população. Neurônios que não cooperaram e são prejudiciais para as redes das quais eles fizeram parte recebem um escore mais baixo e são menos selecionados.

1. Limpe os valores de aptidão de todos os neurônios
2. Monte a rede neural do agente
3. Avalie o agente na execução da tarefa
4. Distribua a aptidão do agente para os neurônios que participaram da rede avaliada
5. Repita os passos de 2 à 4 um número suficiente de vezes
6. Calcule a aptidão média de cada neurônio, dividindo seu valor total de aptidão pelo número de redes em que ele participou.
7. Desempenhe operações de recombinação na população baseada no valor de aptidão média de cada neurônio.

Figura 3.2: Passos básicos em uma geração do SANE.

A Figura 3.3 ilustra como é realizado o processo de neuro-evolução no SANE. Neurônios são escolhidos aleatoriamente, da população de neurônios ocultos, para compor a rede neural. A rede é avaliada e sua aptidão na execução da tarefa é atribuída à todos os neurônios que a constituíram. Novos neurônios são escolhidos para montar uma nova rede e assim sucessivamente até que todos os neurônios tenham sido avaliados.

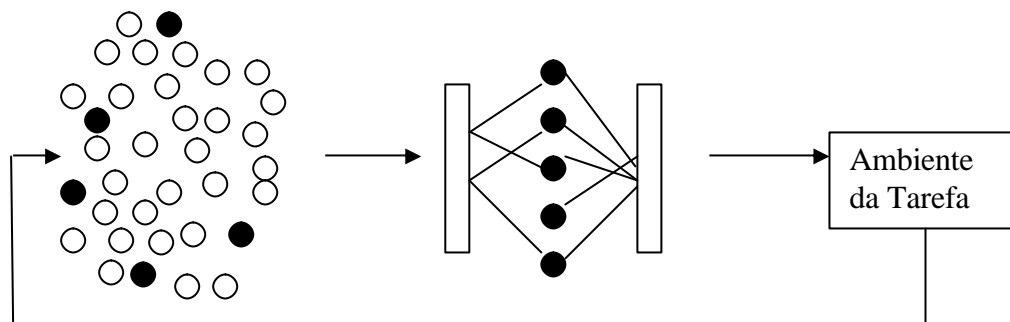


Figura 3.3: Neuro-evolução no SANE.

Uma vez que cada neurônio possui um valor de aptidão, operações de recombinação são usadas para combinar os cromossomos em neurônios com melhor desempenho. Mutação é utilizada em uma taxa baixa para introduzir material genético que pode estar faltando nas populações iniciais ou pode ter sido perdido durante as operações de recombinação.

Cada neurônio oculto é definido como um cromossomo onde os *bits* codificam uma série de definições de conexão. A rede não é totalmente conectada. Cada neurônio oculto se conecta somente com alguns dos neurônios de entrada e alguns dos neurônios de saída. Cada definição de conexão identifica a posição da conexão e o peso da mesma; são utilizados 8 *bits* como campo rótulo e 16 *bits* como campo peso. Não existem conexões entre os neurônios da camada oculta.

O uso de uma precisão limitada (16 bits como campo peso) certamente tem um impacto significativo sobre o algoritmo de aprendizado e a evolução dos pesos.

O valor do rótulo determina a posição da conexão. Se o valor decimal do rótulo D é maior que 127, então a conexão é realizada na unidade de saída $D \bmod O$, onde O é o número total de unidades de saída. Similarmente, se D é menor ou igual a 127, a conexão é realizada na unidade $D \bmod I$, onde I é o número total de unidades de entrada. O campo peso codifica um peso em ponto flutuante para a conexão. A Figura 3.4 mostra como uma rede neural é formada a partir da definição de três neurônios da camada oculta. No exemplo, a primeira definição de conexão para o neurônio A é (15,1.242). Como 15 é menor que 127, a conexão liga o neurônio A ao neurônio 7 ($15 \bmod 8$) da camada de entrada. O peso desta conexão é 1.242.

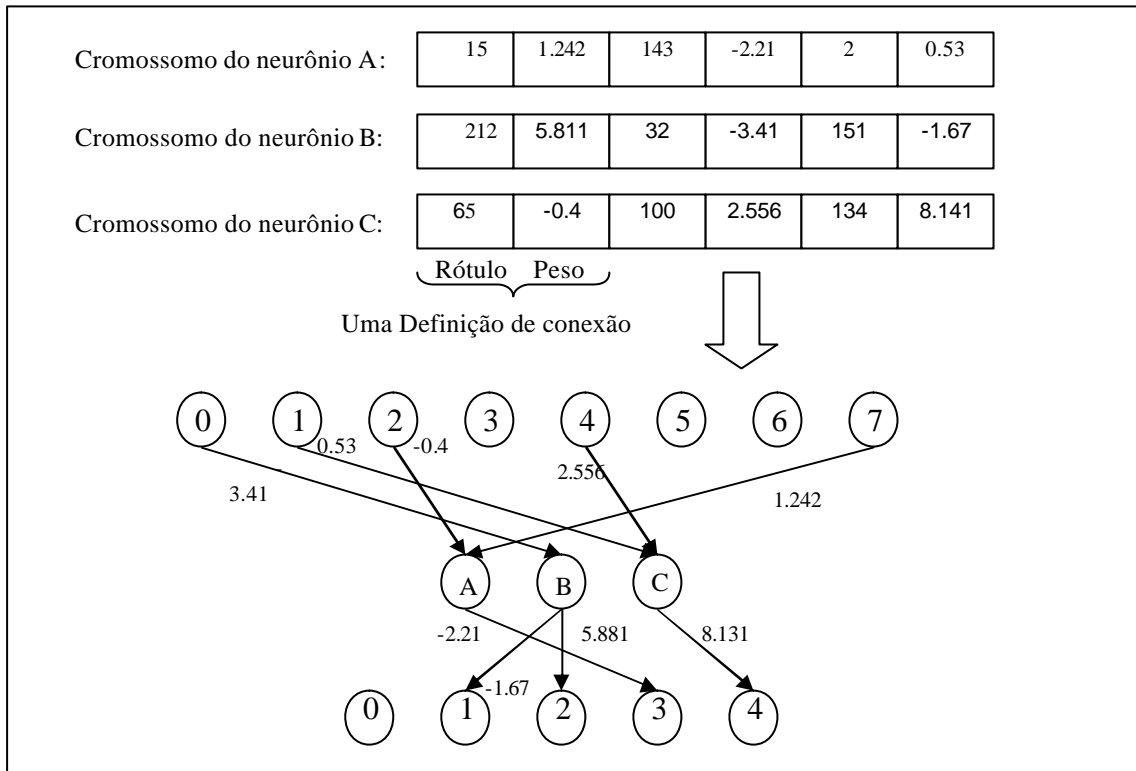


Figura 3.4: Rede neural formada a partir dos cromossomos que definem a camada oculta da rede.

Uma vez que cada neurônio tenha participado de um número suficiente de redes, a população é ordenada de acordo com valores de aptidão média. Os 25% melhores neurônios são recombinados com outros neurônios de igual ou maior valor de função de aptidão. Um operador de recombinação de um ponto é usado para combinar os cromossomos dos dois neurônios, criando-se dois filhos por casamento/combinção. A prole substitui os neurônios com pior desempenho na população. Mutação é usada em uma taxa baixa de 0,1% na nova prole como o último passo da evolução.

A seleção por *rank* é empregada ao invés das seleções padrões de proporcionalidade de aptidões para garantir um *bias* para os neurônios com melhor desempenho. Nas seleções com proporcionalidade de aptidões, uma string *s* é selecionada para casamento com probabilidade f_s/F , onde f_s é a aptidão da cadeia e F é a média de aptidão da população. A medida que a aptidão média das cadeias aumenta, a variação na aptidão diminui. Sem variação suficiente entre as cadeias que desempenham melhor e pior, o algoritmo genético será incapaz de atribuir *bias* significativa em prol das melhores cadeias. Selecionando-se cadeias baseadas no seu *rank* na população, as melhores cadeias sempre irão receber *bias* significativa sobre as piores cadeias mesmo quando as diferenças no desempenho forem pequenas.

3.5.2 Enforced Sub-Populations

O *Enforced Sub-Populations* ESP (GOMEZ, 1997) é uma extensão do SANE, e assim como no SANE, a população consiste de neurônios individuais em vez de redes inteiras, e a rede é formada por um subconjunto de neurônios. Contudo, ESP aloca uma

população separada para cada um dos u neurônios na rede, e um neurônio pode ser recombinado somente com neurônios de sua própria sub-população. Vide Figura 3.5.

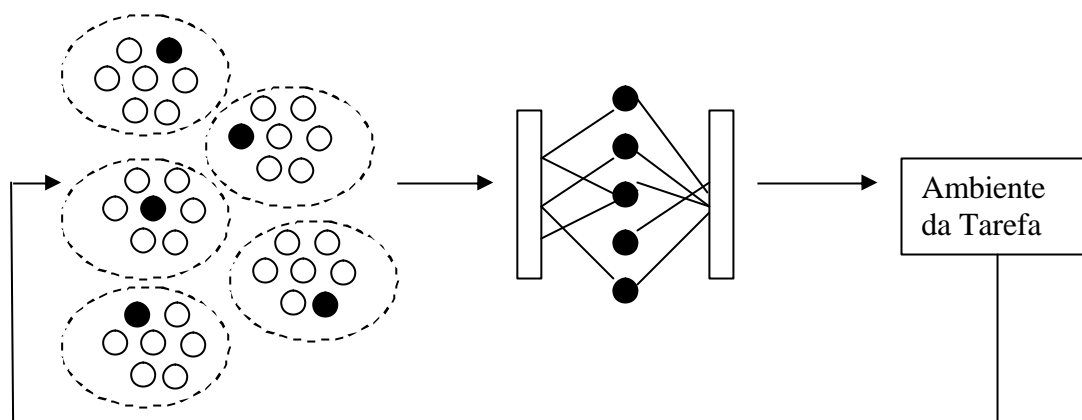


Figura 3.5: Neuro-evolução em ESP. A rede é constituída por um neurônio de cada população de neurônios.

Cada população de neurônios tende a convergir para um papel que apresenta as melhores aptidões quando a rede é avaliada. Desta forma, ESP decompõe o problema de encontrar uma rede satisfatória em vários problemas menores. Segundo Yong (2001), esta abordagem resulta em uma evolução mais eficiente.

ESP acelera a evolução SANE por duas razões: as especializações progressivas dos neurônios não são ocultadas pela recombinação através de especializações que normalmente completam de forma relativamente ortogonal os papéis na rede. Segundo, as redes formadas por ESP sempre consistem de um representante de cada especialização que evolui, um neurônio é sempre avaliado em quão bem ele desempenha seu papel no contexto de outros neurônios. Em SANE, redes podem conter múltiplos membros de alguma especialização e omitir membros de outras, e suas avaliações são, logo, menos consistentes.

Desde que SANE forma redes aleatoriamente através da seleção de neurônios dentro de uma mesma população, um neurônio não pode confiar em ser combinado com neurônios similares em quaisquer duas rodadas. Um neurônio que se comporta de uma forma em uma rodada pode se comportar muito diferentemente em uma outra, resultando em avaliações de aptidão de neurônios que são muito ruidosas. A arquitetura de sub-população de ESP torna a avaliação dos neurônios mais consistente. A medida que as subpopulações se especializam, neurônios evoluem supondo, com uma certeza cada vez maior, os tipos de neurônios com os quais eles serão conectados.

A medida que a evolução progride, cada sub-população irá declinar em diversidade. Isto é um problema, especialmente em evolução incremental, pois, uma população que já convergiu pode não se adaptar facilmente a uma nova tarefa. Para acoplar a transferência de tarefas a despeito da convergência, ESP é combinado com uma técnica de busca interativa conhecida como *Delta-Coding*.

3.5.2.1 *Delta-Coding*

A idéia do *Delta-Coding* (WHITLEY, 1991) é buscar modificações ótimas para a melhor solução corrente. Quando a população de soluções candidatas converge, *Delta-*

Coding salva a melhor solução corrente e inicializa uma população de cromossomos chamadas Δ -cromossomos. Os Δ -cromossomos possuem a mesma quantidade de genes que os cromossomos da melhor solução corrente, e eles consistem de Δ -valores que representam a diferença da melhor solução. Uma nova população é evoluída, selecionando-se Δ -cromossomos, adicionando seus Δ -valores á melhor solução corrente e avaliando o resultado. Os Δ -cromossomos que melhoram o resultado são selecionados para reprodução. Logo, *Delta-Coding* explora a vizinhança da melhor solução. *Delta-Coding* pode ser aplicada várias vezes, com Δ -populações sucessivas representando as diferenças da melhor solução anterior.

3.5.3 NeuroEvolution of Augmenting Topologies

Uma das principais questões em sistemas de neuro-evolução é se a evolução de pesos juntamente com a topologia da rede pode melhorar o desempenho da neuro-evolução. Por um lado, evoluir a topologia juntamente com os pesos torna a busca mais difícil. Por outro lado, evoluir topologias pode economizar tempo de ter que se encontrar o número certo de neurônios ocultos para um problema particular.

Ao contrário da maioria dos sistemas de neuro-evolução que evoluem arquiteturas de redes neurais fixas, como é o caso do SANE e do ESP, o *NeuroEvolution of Augmenting Topologies* (NEAT) (STANLEY, 2002) evolui não só os pesos das conexões da rede, como sua própria topologia, buscando um aprendizado mais rápido.

No NEAT, cromossomos são representações lineares da conectividade da rede. Cada genoma inclui uma lista de genes de conexão. Cada gene de conexão especifica o nó de entrada, o nó de saída, o peso da conexão, se a conexão entre os nós está habilitada ou não, e um número inovador, que permite encontrar genes correspondentes.

Na Figura 3.6 é apresentada o mapeamento de um genótipo para um fenótipo no NEAT. Observe que o segundo gene está desabilitado, logo, a conexão que ele especifica não é observada no fenótipo.

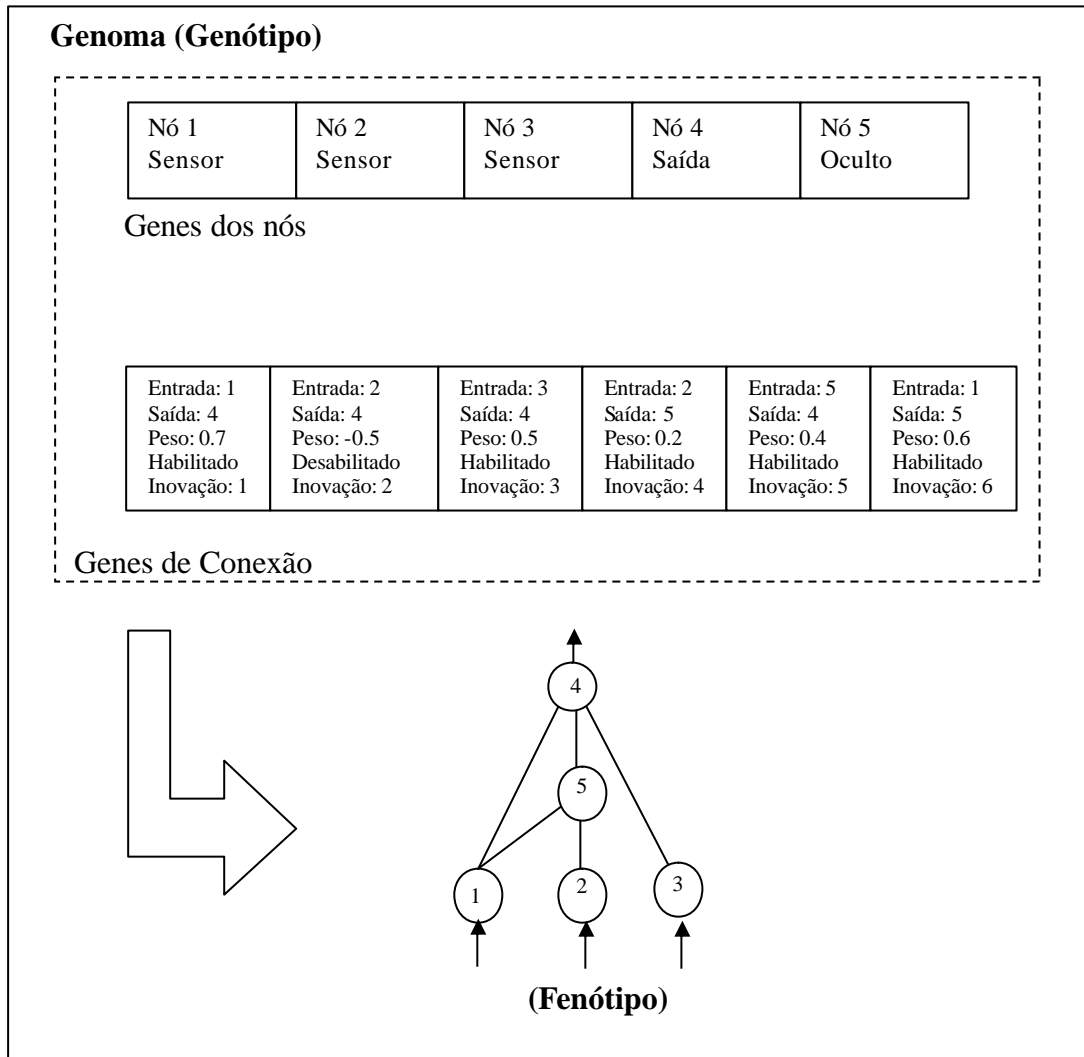


Figura 3.6: Mapeamento do genótipo para o fenótipo.

A mutação em NEAT pode alterar tanto os pesos das conexões quanto a estrutura da rede. Os pesos da conexão sofrem mutação como em qualquer sistema de neuro-evolução, com cada conexão sendo perturbada ou não em cada geração. Mutações estruturais ocorrem de duas formas. Na mutação de adição de conexões, uma nova conexão é adicionada conectando dois nós previamente desconectados. Na mutação de adição de nós, uma conexão existente é dividida e um novo nó é colocado onde a conexão antiga existia. A conexão antiga é desabilitada e duas novas conexões são adicionadas ao genoma. Este método de adição de nós integra, imediatamente, novos nós à rede.

Através da mutação, os cromossomos em NEAT ficarão gradualmente maiores. Cromossomos de tamanhos variados irão resultar, algumas vezes com conexões completamente diferentes nas mesmas posições.

Na Figura 3.7, são apresentados os dois tipos de mutação no NEAT, adição de nós e adição de conexões. O número no topo de cada genoma é o número de inovação. Os números de inovação são marcadores históricos que identificam o ancestral de cada gene. Cada novo gene recebe um novo número de inovação.

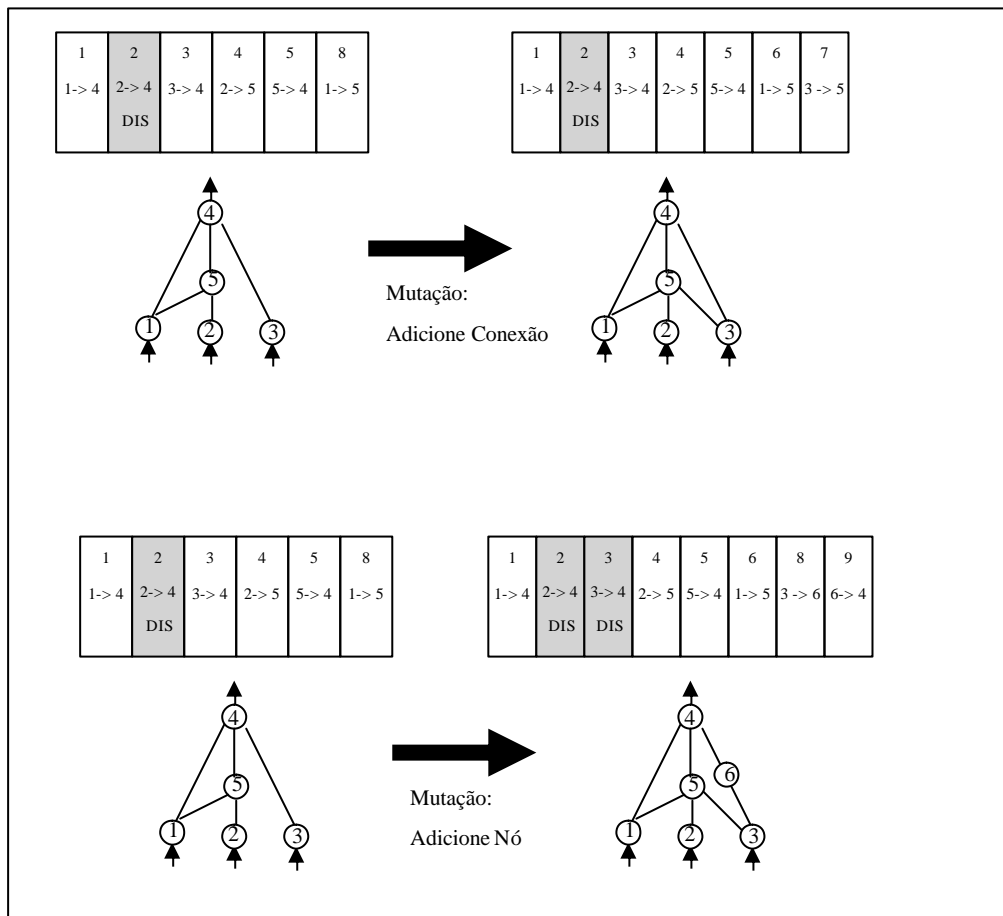


Figura 3.7: Os dois tipos de mutação estrutural no NEAT.

Para poder identificar quais genes combinam ao realizar recombinação, o NEAT mantém um número de inovação global que guarda a origem histórica de cada gene no sistema. Sempre que um novo gene é criado, o número de inovação global é incrementado e atribuído a ele; o número de inovação global representa a cronologia do aparecimento de cada gene no sistema. Como exemplo, suponha que as duas mutações na Figura 3.8 ocorreram uma após a outra no sistema. O novo gene de conexão criado na primeira mutação recebe o número 7, e os dois novos genes de conexão adicionados durante a mutação de adição de nó, recebem os números 8 e 9. No futuro, se estes genes vierem a se combinar, os filhos herdarão o mesmo número de inovação global de cada gene; números de inovação nunca são alterados.

Ao realizar recombinação, os genes em que os cromossomos possuem o mesmo número de inovação global são alinhados. Genes que não combinam são chamados disjuntos(D) ou excedentes (E), dependendo se eles ocorrem dentro ou fora da faixa de valores dos números de inovação do outro pai. Eles representam estruturas que não estão presentes no outro genoma. Ao compor os filhos, genes são aleatoriamente escolhidos dos pais cujos genes combinam, enquanto todos os genes excedentes ou disjuntos são incluídos do pai com maior aptidão ou de ambos os pais se estes possuem aptidões equiparáveis.

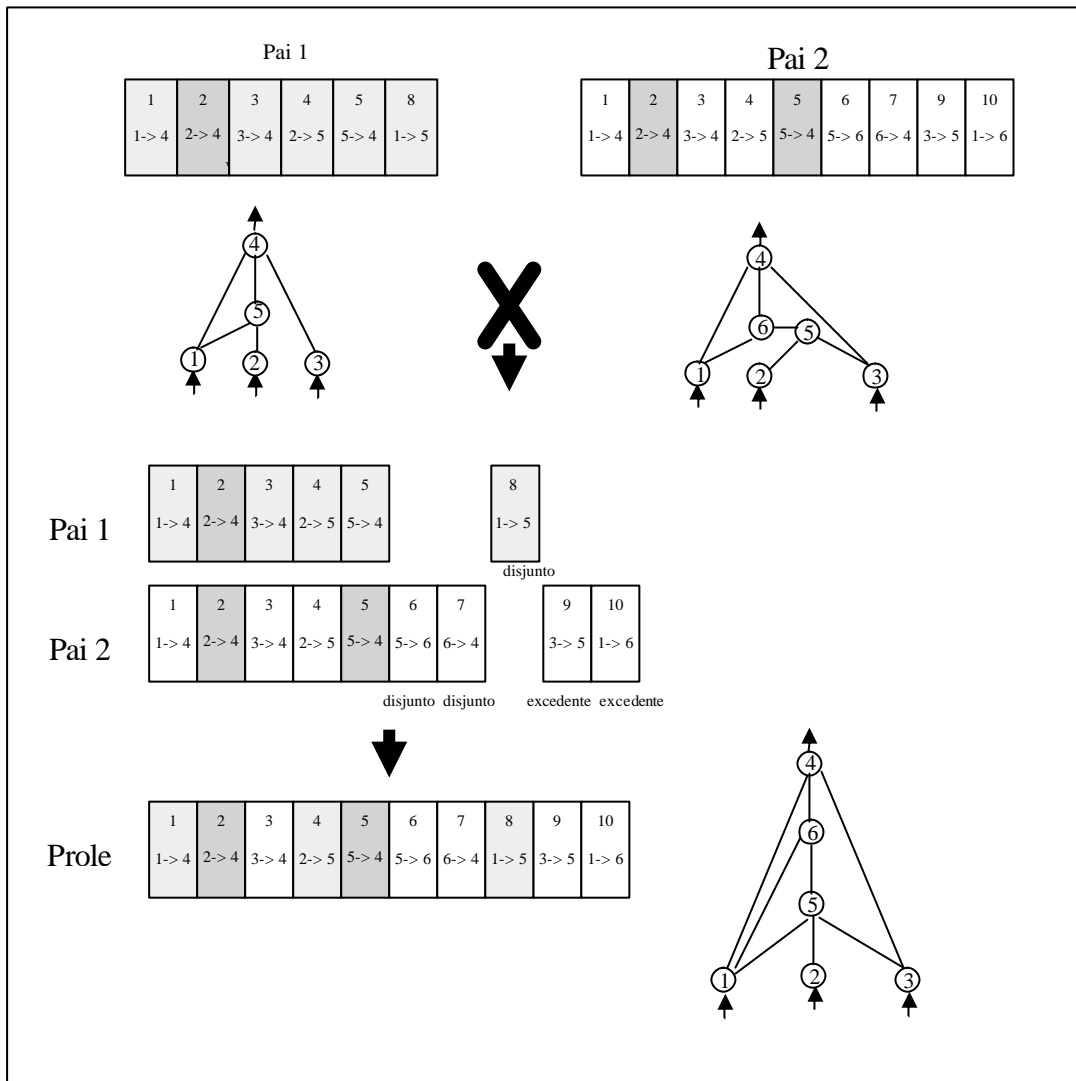


Figura 3.8: Recombinação de duas redes usando o NEAT.

Ao adicionar novos genes à população e casar cromossomos que representam estruturas diferentes, o sistema pode formar uma população de topologias diversas. Contudo, estruturas menores otimizam mais rápido que estruturas maiores, e a adição de nós e conexões normalmente diminui a aptidão da rede inicialmente. Estruturas aumentadas recentemente possuem pouca chance de sobreviver por mais que uma geração mesmo que as inovações que elas representam sejam cruciais para resolver a tarefa em questão. A solução para proteger a inovação é usar classificação.

A idéia é dividir a população em espécies, de forma que topologias similares pertençam a mesma espécie. O número de genes disjuntos e excedentes entre um par de genes é uma medida natural da sua compatibilidade. Quanto mais disjuntos dois cromossomos são, menos história evolucionária eles compartilham, e logo, menos compatíveis eles são. Logo, pode-se medir a compatibilidade de duas estruturas diferentes no NEAT como uma simples combinação linear do número de genes

excedentes (**E**) e disjuntos(**D**), tanto quanto, a diferença média dos pesos dos genes que combinam (**W**):

$$d = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot W$$

Os coeficientes, c_1 , c_2 e c_3 permitem ajustar a importância dos três fatores, e o fator N , o número de genes no genoma maior, normaliza o tamanho do genoma (N pode ser 1 se ambos os cromossomos são pequenos, isto é, se consistem de menos de 20 genes).

A distância d permite classificar usando um limite de compatibilidade. Cromossomos são comparados uns com os outros uma vez; se a distância dos cromossomos para membros escolhidos aleatoriamente é menor que este limite, ele é colocado dentro da espécie. Cada genoma é localizado dentro da primeira espécie para a qual esta condição é satisfeita, de forma que um genoma nunca está em mais de uma espécie. A medida d para um par de cromossomos é linear no número de conexões mesmo se d expressa precisamente a compatibilidade entre topologias multidimensionais.

Como mecanismo de reprodução, é usado compartilhamento de aptidões, onde organismos da mesma espécie compartilham a mesma aptidão do seu nicho.

NEAT guia a busca através de espaços mini-dimensionais começando sempre com uma população uniforme de redes com zeros neurônios ocultos. Novas estruturas são inseridas incrementalmente como ocorrências de mutações estruturais, e somente aquelas estruturas consideradas úteis através da avaliação de aptidão sobrevivem. Em outras palavras, as elaborações estruturais que ocorrem no NEAT são sempre justificadas. Desde que a população começa minimamente, a dimensionalidade do espaço de busca é minimizado, isto provê uma vantagem em desempenho quando comparado com outras abordagens.

4 MODELO DE COORDENAÇÃO MULTIAGENTE NEURO-EVOLUTIVO

Neste capítulo o modelo proposto é descrito. Na seção 4.1, será apresentado o algoritmo utilizado para a evolução do comportamento dos agentes. Em seguida, será apresentado o diagrama de classes do modelo desenvolvido. Por fim, na seção 4.5, descreve-se o ambiente de simulação desenvolvido.

4.1 Algoritmo evolucionário

A base do nosso modelo de coordenação adaptativa é uma extensão do ESP, método de neuro-evolução descrito no capítulo anterior.

Na nossa extensão, cada neurônio da camada oculta se conecta com todos os neurônios da camada de entrada e todos os neurônios da camada de saída, ou seja, a rede que descreve o comportamento de cada agente é uma *2-layer-feedforward* totalmente conectada.

Desta forma, o genótipo de cada neurônio é mantido “limpo”, isto é, cada cromossomo é uma cadeia dos pesos das conexões do neurônio oculto atual com os outros neurônios das camadas de entrada e de saída, não havendo necessidade de armazenar qualquer informação rótulo, como acontece no SANE e no ESP.

Contudo, a diferença mais importante do presente modelo é que a topologia da rede não é fixa. É possível definir o tamanho máximo de neurônios na camada oculta. Porém, o algoritmo evolucionário proposto busca uma quantidade de neurônios ocultos que possibilite aos agentes executarem bem a tarefa. Desta forma, a topologia da rede é mantida mais simples. Adicionalmente, alguns experimentos (vide capítulo 5) comprovaram que o tempo de treinamento é reduzido significativamente a medida que diminuimos a quantidade de neurônios na camada oculta.

No passo 1 do algoritmo evolucionário proposto, descrito na Figura 4.1, uma nova rede neural experimental é construída para cada agente. No sub-passo **b**, é definido, aleatoriamente, quantos s neurônios devem existir na camada oculta. Posteriormente, no passo **c**, um neurônio é escolhido aleatoriamente da população correspondente. Isto é, o neurônio i é escolhido aleatoriamente da população de neurônios i , onde i varia de 1 a s .

1. Crie p populações de neurônios ocultos para a rede neural de cada agente, onde p é o tamanho máximo da camada oculta.
2. Para cada agente
 - a. Zere os valores de aptidão de cada neurônio
 - b. Gere um inteiro $s \in [0, p]$
 - c. Crie uma rede neural com s neurônios ocultos obtidos aleatoriamente da respectiva população
3. Avalie os agentes de acordo com uma função de aptidão apropriada para a tarefa.
4. Para cada agente
 - a. Adicione a aptidão para a variável de aptidão de cada neurônio que participou da rede
5. Repita os passos de 2 a 4 um número suficiente de vezes
6. Calcule a aptidão média de cada neurônio, dividindo seu valor total de aptidão pelo número de redes em que ele participou.
7. Execute operações de recombinação na população baseada no valor de aptidão média de cada neurônio.

Figura 4.1: Algoritmo básico de treinamento.

A depender da complexidade da tarefa, algumas vezes não é possível obter um comportamento coerente do agente com evolução direta. Gomez (1997) propõe que uma abordagem em que comportamentos complexos sejam obtidos através de um aprendizado incremental. Isto é, os agentes são inicialmente treinados em tarefas mais simples e a medida que seus comportamentos convergem, aumenta-se o grau de dificuldade da tarefa, tornando-se cada vez mais complexa.

Utilizou-se o algoritmo de *Delta-Coding*, apresentado na seção 3.5.2.1, para implementar as transições para tarefas cada vez mais complexas.

$$t_1 \overset{\Delta}{?} t_2 \overset{\Delta}{?} t_3 \overset{\Delta}{?} \dots \overset{\Delta}{?} t_n$$

Em cada transição de tarefa, a melhor solução é salva, uma população de Δ -cromossomo é inicializada e esta população é evoluída para que o agente se adapte a nova tarefa. Quando a melhor solução da tarefa t_i evolui para a tarefa t_{i+1} , os Δ -valores são adicionado ao melhor resultado do agente, formando um novo melhor resultado. A Figura 4.2, ilustra o algoritmo de *Delta-Coding*.

1. Crie p população de Δ -valores, onde p é o tamanho máximo da camada oculta.
2. Para cada agente
 - a. Zere os valores de aptidão de cada Δ -valor
 - b. Altere o conjunto de Δ -valores que serão adicionados aos pesos da rede
 - c. Crie uma rede neural, somando os Δ -valores selecionados acima ao melhor resultado corrente.
3. Avalie os agentes de acordo com uma função de aptidão apropriada para a tarefa.
4. Para cada agente
 - a. Adicione a aptidão para a variável de aptidão de cada Δ -valor somado aos pesos da rede
 - b. Se o desempenho do agente melhorou com estes Δ -valores, some estes valores aos pesos do agente.
5. Repita os passos de 2 a 4 um número suficiente de vezes
6. Calcule a aptidão média de cada Δ -valor, dividindo seu valor total de aptidão pelo número de redes em que ele participou.
7. Desempenhe operações de recombinação na população baseada no valor de aptidão média de cada Δ -valor.

Figura 4.2: Algoritmo de *Delta-Coding*.

4.2 Características do Modelo Proposto

Nesta seção, são apresentados alguns detalhes do modelo proposto. Na seção 4.2.1, é abordada a codificação dos neurônios em cromossomos binários e reais. Na seção 4.2.2, são apresentados os operadores genéticos utilizados. Na seção 4.2.3 apresenta-se as informações utilizadas como entrada para as redes neurais dos agentes, bem como, o significado da saída da rede.

4.2.1 Cromossomos

Cada cromossomo corresponde a um neurônio da camada oculta e armazena os pesos das conexões que o neurônio que ele representa tem com os outros neurônios das camadas de entrada e saída. Inicialmente, representou-se cada neurônio da rede como um cromossomo binário. Posteriormente, foi possibilitado que o neurônio fosse codificado como uma cadeia de números reais.

Considerou-se que as redes são completamente conectadas, de forma que se uma rede possui, por exemplo, dois neurônios na camada de entrada e quatro na camada de saída, o neurônio oculto armazena um total de seis pesos. Vide Figura 4.3.

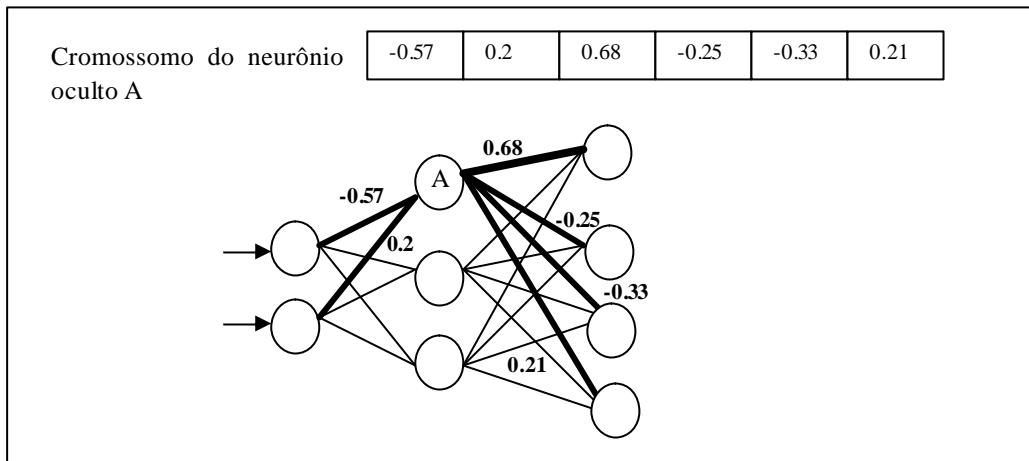


Figura 4.3: Codificação do neurônio no modelo proposto.

Na representação binária, é necessário especificar quantos genes codificam um valor real. Desta forma, para decodificar os pesos do neurônio, dividi-se a cadeia de genes em subcadeias, e para cada subcadeia, calcula-se o valor real que ela representa.

Na Figura 4.4, considere que o cromossomo binário descrito representa o neurônio 2 da rede neural. Para decodificar os pesos das conexões do neurônio 2 com os outros neurônios da rede, foram seguidos os passos descritos na Figura 4.4:

Partindo do Cromossomo binário abaixo:

001111100000110110101001001101100000101101101110111100111100

E considerando que cada dez *bits* correspondem a um valor real, subdividimos a cadeia binária em subcadeias, cada uma consistindo de 10 bits:

0011111000 0011011010 1001001101 1000001011 0110111011 1100111100

Para cada subcadeia é aplicada a fórmula abaixo:

$$X = \min + (\max - \min) * \text{base10}(\text{subcadeia}) / 2^l - 1, \text{ onde } \min = -1, \max = 1^4, l = 10$$

No final, é obtido uma cadeia de reais : -0.515 -0.573 0.1515 0.0224 -0.133 0.6187

É possível visualizar os pesos na rede abaixo

Figura 4.4: Codificação da Rede Neural partindo de Cromossomos binários.

⁴ Restringir os pesos entre -1 e +1 impactam na aprendizagem, podendo justificar o não aprendido para entradas mais elaboradas (vide seção 5.2.1).

Alternativamente, pode-se representar o cromossomo como uma cadeia de números reais. Neste caso, cada elemento da cadeia já representa o peso da conexão correspondente. O comportamento evolutivo pode ser

4.2.2 Operadores Genéticos

O operador de seleção utilizado foi o algoritmo da roleta, já descrito na seção 2.3.

Foram utilizados dois operadores de recombinação, a depender da codificação utilizada para o cromossomo (binária ou real).

Para a recombinação de cromossomos binários, utilizou-se um operador padrão com um ponto de corte.

Para a recombinação de cromossomos reais, foi utilizado o operador de recombinação aritmético, onde o gene de cada filho é obtido pela combinação linear dos genes correspondentes nos pais.

$$\text{geneFilho}_1 = a * (\text{genePai}_1) + (1 - a) * \text{genePai}_2;$$

$$\text{geneFilho}_2 = (1 - a) * \text{genePai}_1 + a * \text{genePai}_2;$$

Exemplo:

Partindo dos pais

$$\text{pai}_1 = -0.0292 \ -0.4206 \ -0.7125 \ -0.6795 \ 0.4273$$

$$\text{pai}_2 = -0.3454 \ -0.4604 \ 0.3289 \ -0.4708 \ 0.4278$$

e considerando $a = 0.3$, tem-se:

$$\text{filho}_1 = -0.2505 \ -0.4485 \ 0.0165 \ -0.5334 \ 0.4277$$

$$\text{filho}_2 = -0.1241 \ -0.4325 \ -0.4000 \ -0.6169 \ 0.4275$$

Para mutação, foram utilizados também dois operadores. Quando o cromossomo é codificado como uma cadeia de números binários, o operador de mutação é aplicado com dada probabilidade, em cada um dos filhos gerado. Essa probabilidade é parametrizada, mas, recomenda-se que seja um valor baixo, como 0.2, por exemplo. Na mutação binária, inverte-se um alelo, escolhido aleatoriamente (0 passa para 1 e 1 passa para 0).

Na mutação aritmética, o operador de mutação também é aplicado com dada probabilidade em cada filho gerado. Porém aqui, um alelo, é substituído por um número real gerado aleatoriamente pertencente ao intervalo $[-1, +1]$.

4.2.3 Entrada e Saída das Redes Neurais

A entrada fornecida para a rede neural, bem como a saída retornada pela mesma, é altamente dependente do problema que se busca resolver com o modelo de coordenação proposto. Na seção 4.3, apresenta-se o domínio de aplicação utilizado como estudo de caso.

Por hora, pode-se adiantar que se trata de um grupo de agentes que se locomove em um ambiente bidimensional.

Avaliou-se seis entradas possíveis para o estudo de caso utilizado. As entradas variaram tanto em termos de valor quanto em termos de percepção. Em termos de percepção, considerou-se entradas onde o agente recebia apenas parte do estado do mundo e entradas onde ele recebia o estado total do mundo. Em termos de valor, as entradas variaram entre posições absolutas, posições relativas e sinais da direção. Para mais detalhes sobre as entradas utilizadas, consulte a seção 5.1.1.

No estudo de caso, cada agente pode se mover em uma das quatro direções: N, S, L, O. O tamanho da camada de saída das redes neurais dos predadores é sempre igual a quatro. Para saber qual a próxima direção do agente, foi aplicado a política do “vencedor leva tudo” (*winner takes all*). Ou seja, a direção escolhida é aquela cujo neurônio correspondente apresenta o maior valor de saída.

4.3 Domínio de Aplicação

A tarefa de captura de uma presa é um caso especial de problemas de perseguição-evasão (MILLER, 1994). Tais tarefas consistem de um ambiente com uma ou mais presas e um ou mais predadores. Os predadores se movem ao redor do ambiente tentando pegar as presas, e as presas tentam escapar dos predadores. Tarefas de perseguição-evasão são interessantes porque existem no mundo real, e oferecem um objetivo claro que requer coordenação complexa no que diz respeito ao ambiente, outros agentes com o mesmo objetivo e agentes adversários. Eles são verdadeiros desafios mesmo para os melhores sistemas de aprendizagem, permitindo medição precisa, análise e visualização das estratégias evoluídas.

No presente trabalho explorou-se várias tarefas para o domínio da presa e do predador. Dentre as quais, pode-se destacar: a variação da quantidade de agentes predadores, a velocidade da presa, a posição inicial da presa, o algoritmo utilizado pela presa para fugir dos predadores e o formato do mundo (toroidal ou limitado).

O mundo é constituído por uma presa e até quatro predadores. A presa é controlada por um algoritmo simples. Os comportamentos disponíveis para a presa são realizar um movimento aleatório, caminhar sempre em uma mesma direção ou fugir do predador mais próximo. Os predadores são controlados por redes neurais. O objetivo é evoluir as redes neurais para formar um time eficiente na tarefa de capturar a presa.

O ambiente é bidimensional e sua dimensão pode ser configurada, na maioria dos experimentos realizados, foi utilizada a dimensão (30,30). O mundo pode ser configurado para se comportar de forma toroidal ou limitado. No mundo toroidal, quando o agente atinge a última célula do mundo, ele automaticamente passa para a primeira na mesma direção contrária. Em outras palavras, a célula seguinte a última abscissa/coordenada do mundo é a primeira abscissa/coordenada do mundo. No mundo limitado, ao atingir a última abscissa/coordenada do mundo, o agente é impedido de prosseguir naquela direção. Em qualquer das situações, não existem obstáculos. Todos os agentes podem se mover em quatro direções: **N** (Norte), **S** (Sul), **L** (Leste), **O** (Oeste). A velocidade e a posição inicial da presa podem ser configuradas de forma a facilitar/difícultar o problema. A presa pode se mover tão rápido quanto os predadores.

Na Figura 4.5, é ilustrado o algoritmo evolucionário resultante para o problema de captura da presa.

1. Para cada agente
 - a. Zere os valores de aptidão de cada neurônio
 - b. Gere um inteiro s ? [0, Qtde máxima de neurônios ocultos]
 - c. Crie uma rede neural com s neurônios ocultos
 - d. Posicione o agente no mundo bidimensional
2. Posicione a presa de acordo com a posição inicial especificada
3. Para os cenários de 1 à k
 - a. Posicione os predadores em um extremo aleatório
 - b. Para os passos de 1 à n
 - i. Para cada agente, execute o próximo movimento
 - ii. Movimente a presa de acordo com a velocidade especificada
 - c. Calcule a distância média final d_f entre os predadores e a presa
 - d. Avalie o time de acordo com a função abaixo:

$$\text{aptidao} += 100 / d_f$$
4. Calcule a aptidão média do grupo da seguinte maneira:

$$\text{aptidaoMedia} = \text{aptidao} / k;$$
5. Para cada agente
 - a. Adicione a aptidão para a variável de aptidão de cada neurônio que participou da rede
6. Repita os passos de 2 à 5 um número suficiente de vezes
7. Calcule a aptidão média de cada neurônio, dividindo seu valor total de aptidão pelo número de redes em que ele participou.
8. Desempenhe operações de recombinação na população baseada no valor de aptidão média de cada neurônio.

Figura 4.5: Ciclo evolucionário para a captura da presa.

Observe que a tarefa de captura de uma presa não-estacionária é uma tarefa de decisão seqüencial. Só é possível avaliar se o time de agentes possui uma boa política de decisão após a execução de uma seqüência de passos. Considere, por exemplo, que o ambiente possui a seguinte configuração: mundo toroidal, onde a presa se locomove na mesma velocidade que os predadores, e fugindo sempre do predador mais próximo. A primeira vista, poderia se pensar que uma boa estratégia seria que a cada passo de execução, os agentes se aproximassem da presa. Contudo, como o mundo considerado é toroidal, os predadores nascem todos em um mesmo extremo e a presa se locomove na mesma velocidade que os predadores, a presa nunca seria capturada, mesmo que sua distância inicial para o predador mais próximo fosse 1.

Os agentes precisam “aprender” uma estratégia mais inteligente, uns tem que impedir o avanço da presa em alguma direção enquanto os outros tentam captura-la.

Em cada ciclo evolucionário, para avaliar se a estratégia do grupo de predadores é boa, foram criados diferentes cenários. Cada cenário consiste na alteração das posições iniciais dos predadores. Para cada cenário criado, é permitido que os agentes se locomovam por n passos antes de seus comportamentos serem avaliados.

4.4 Diagrama de Classes

Nesta seção, apresenta-se alguns diagramas de classes do modelo desenvolvido.

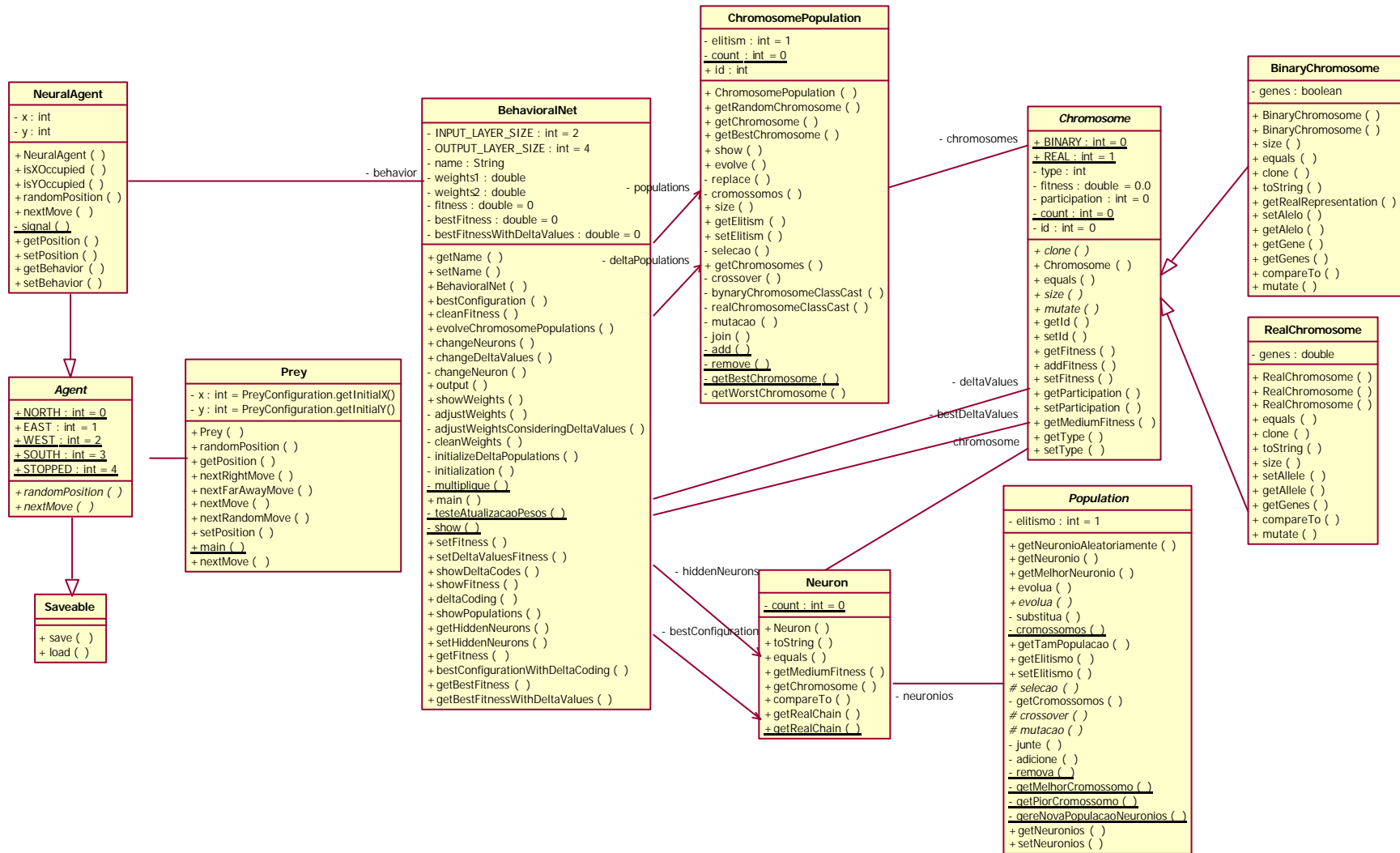


Figura 4.6: Principais classes do modelo de coordenação evolucionário.

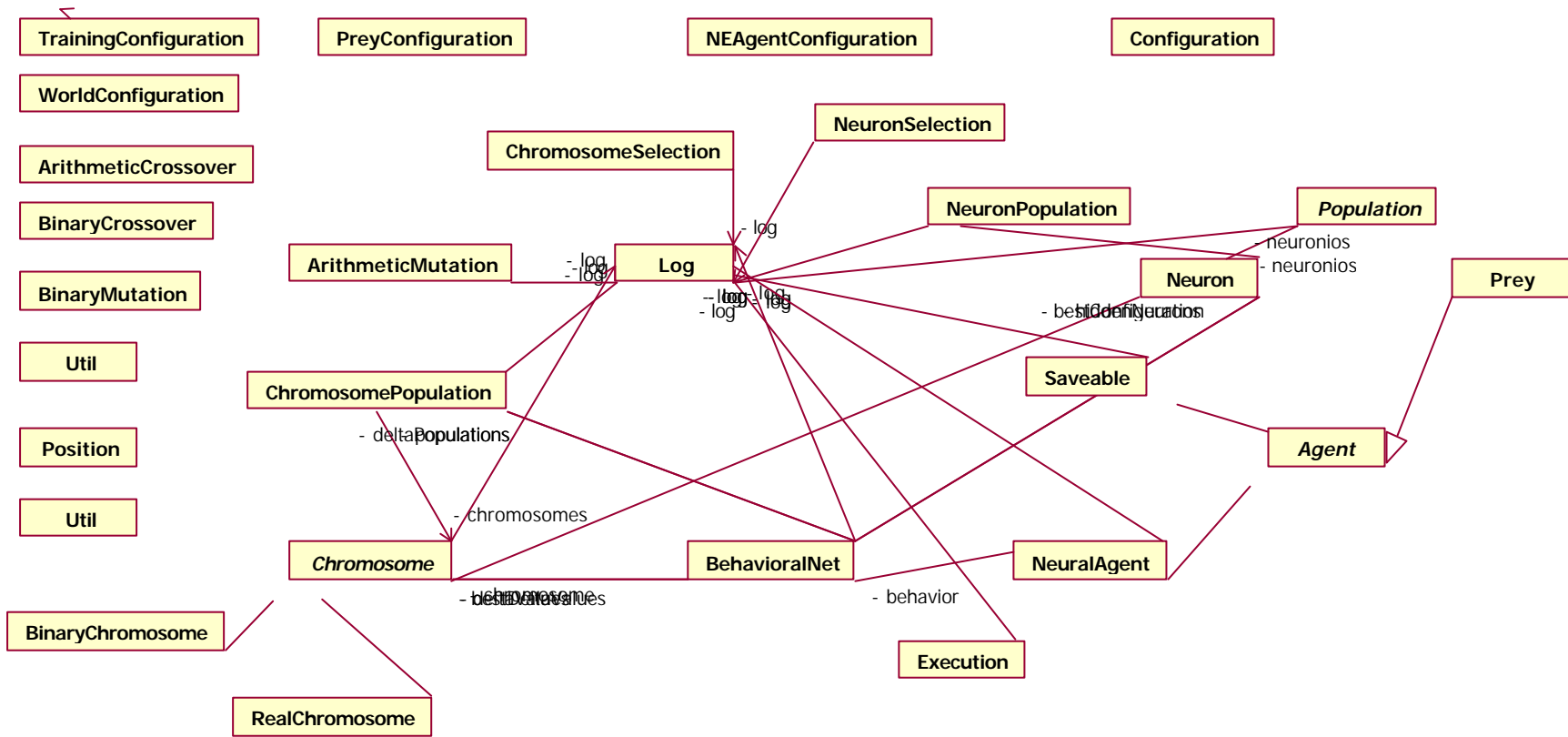


Figura 4.7: Todas as classes do modelo evolucionário.

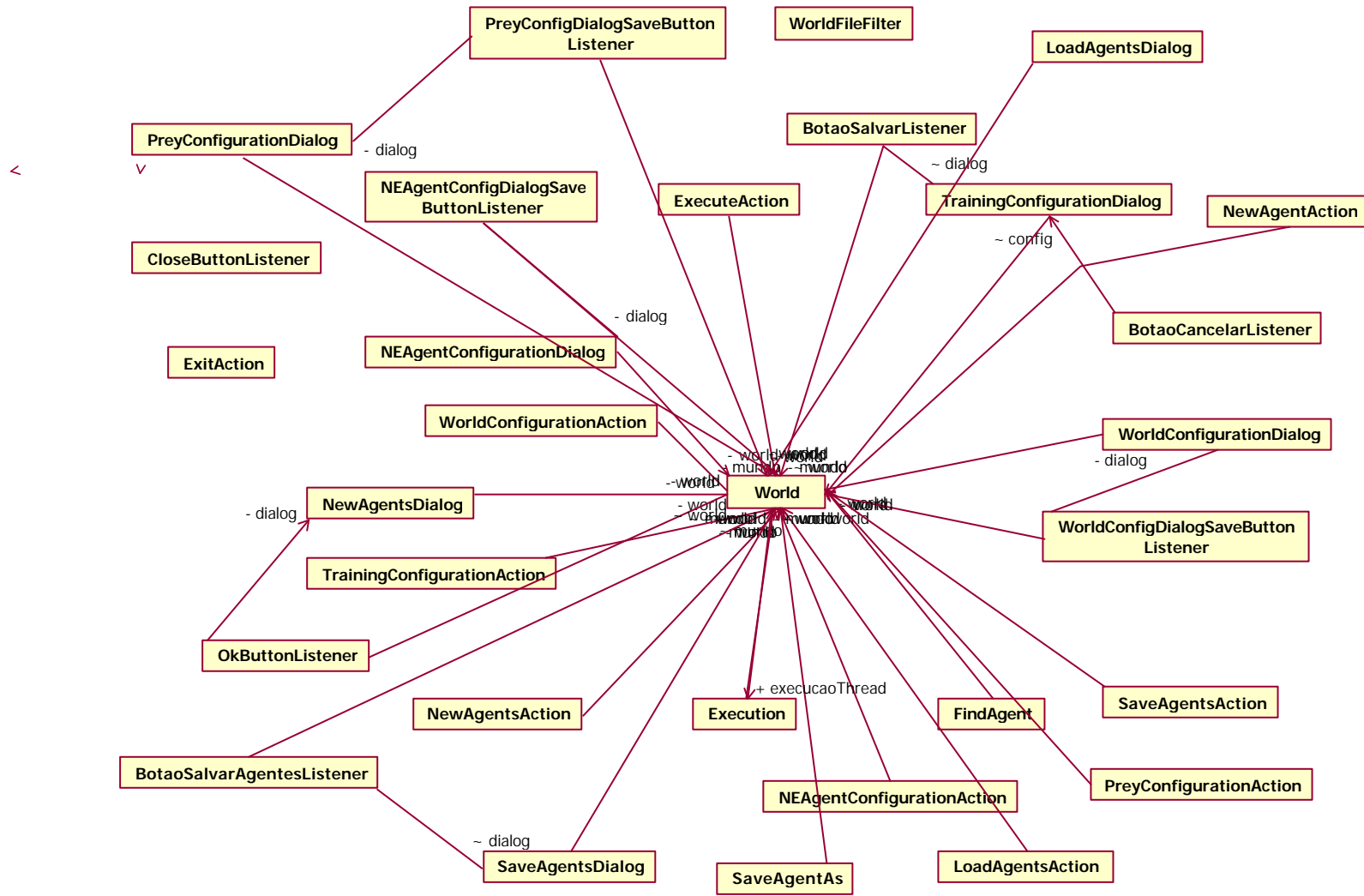


Figura 4.8: Classes do ambiente de simulação.

4.5 Ambiente de Simulação

O modelo proposto foi implementado com JSDK1.5. Utilizou-se o Eclipse 3.0 como ambiente de desenvolvimento. Além da codificação do modelo proposto de forma parametrizável, desenvolveu-se também um ambiente de simulação.

O ambiente de simulação que permite várias configurações do ambiente, a execução do treinamento, o armazenamento de comportamentos aprendidos, a simulação propriamente dita e a visualização dos resultados.

O ambiente possui uma interface visual intuitiva e facilita a execução de várias simulações, inclusive, dos experimentos relatados no capítulo 5.

4.5.1 Manipulação dos Comportamentos

Comportamentos podem ser gerados, armazenados e recuperados através do menu Predadores, conforme ilustrado na Figura 4.9.

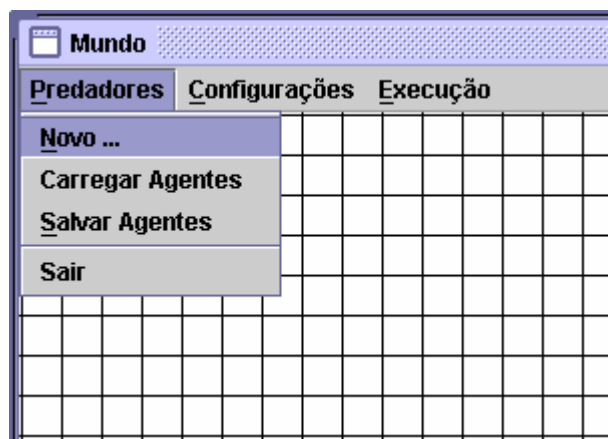


Figura 4.9: Manipulação de comportamentos no ambiente.

As opções do menu são:

- “Novo”: permite especificar quantos predadores existirão no mundo.
- “Carregar Agentes”: especifica qual o comportamento, previamente salvo, de cada predador.
- “Salvar Agentes”: salva o comportamento corrente de cada predador do sistema.

As Figuras 4.10 e 4.11 ilustram como carregar e salvar o comportamento dos agentes.

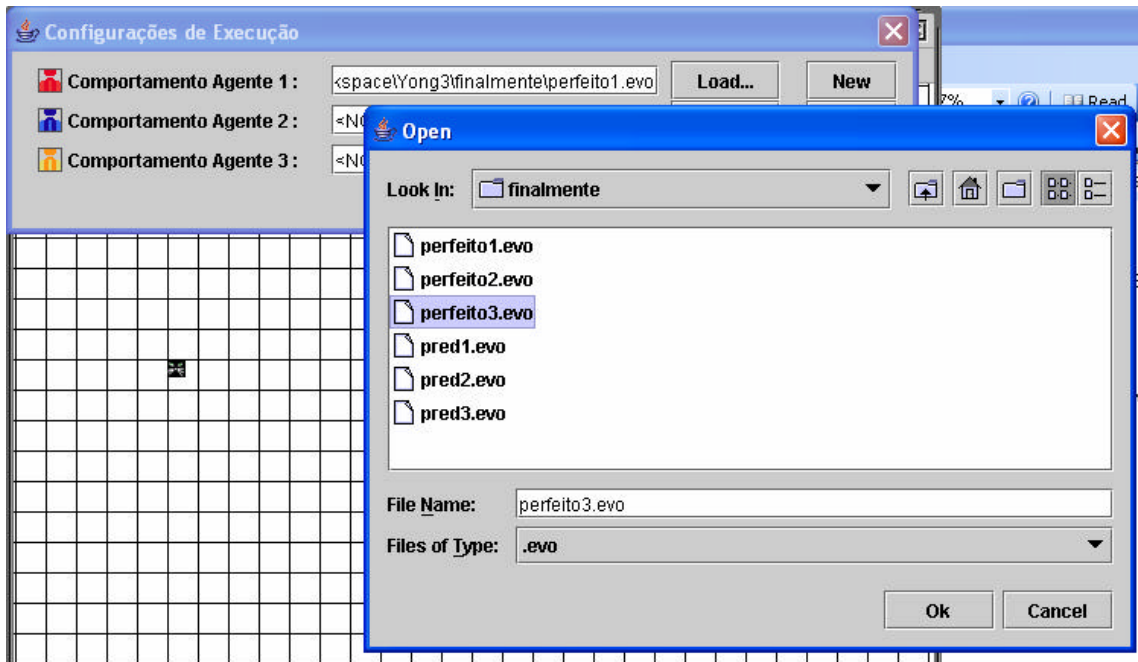


Figura 4.10: Carrega o comportamento dos predadores.

Ao escolher a opção “Salvar agentes”, virá a indicação de quantos neurônios o agente possui na camada oculta e será solicitado que o usuário informe onde o comportamento deve ser salvo.

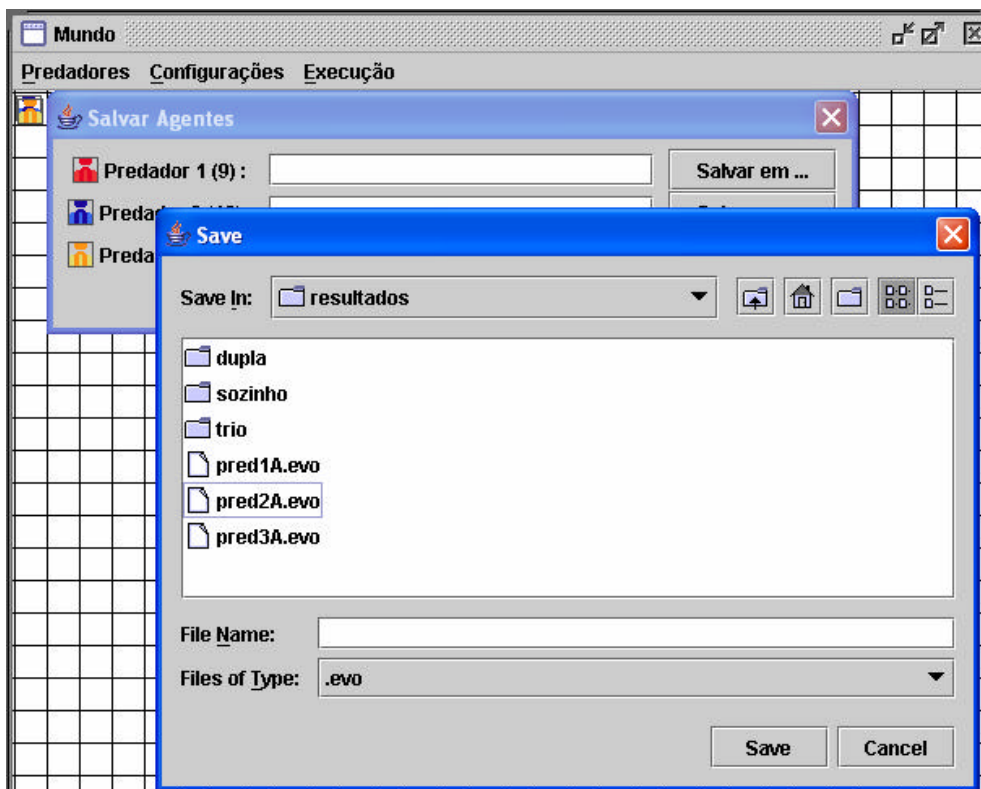
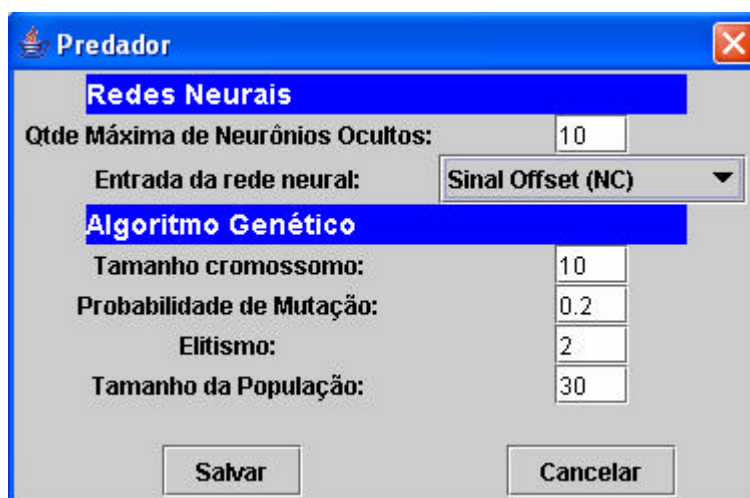


Figura 4.11: Salva o comportamento atual dos agentes.

4.5.2 Configurações

O menu de configurações, permite que se configure o predador, a presa, o mundo e o treinamento.

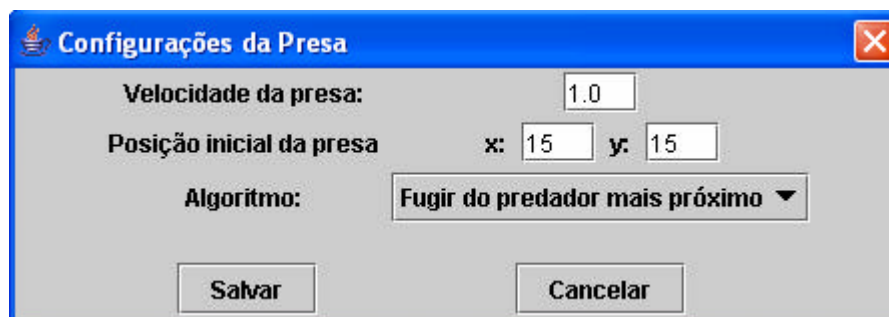
Para o predador, é possível configurar: a quantidade máxima de neurônios na camada oculta, a entrada da rede neural, o tamanho do cromossomo binário, a probabilidade de mutação, o elitismo e o tamanho das populações de neurônios. Vide Figura 4.12.



The image shows a Windows-style dialog box titled "Predador". It has a blue title bar with a close button (X) in the top right corner. The dialog is divided into two sections. The first section, "Redes Neurais", has a blue header and contains two settings: "Qtde Máxima de Neurônios Ocultos:" with a text box containing "10", and "Entrada da rede neural:" with a dropdown menu showing "Sinal Offset (NC)". The second section, "Algoritmo Genético", also has a blue header and contains four settings: "Tamanho cromossomo:" with a text box containing "10", "Probabilidade de Mutação:" with a text box containing "0.2", "Elitismo:" with a text box containing "2", and "Tamanho da População:" with a text box containing "30". At the bottom of the dialog are two buttons: "Salvar" on the left and "Cancelar" on the right.

Figura 4.12: Configuração dos predadores.

Para a presa é possível configurar: a posição inicial, a velocidade (ou melhor, a probabilidade dela realizar um movimento a cada passo de execução) e o algoritmo a ser utilizado (fugir do predador mais próximo, realizar movimentos aleatório e caminhar sempre na mesma direção). A Figura 4.13 ilustra as configurações possíveis para a presa.



The image shows a Windows-style dialog box titled "Configurações da Presa". It has a blue title bar with a close button (X) in the top right corner. The dialog contains three settings: "Velocidade da presa:" with a text box containing "1.0", "Posição inicial da presa" with two text boxes for "x:" and "y:" both containing "15", and "Algoritmo:" with a dropdown menu showing "Fugir do predador mais próximo". At the bottom of the dialog are two buttons: "Salvar" on the left and "Cancelar" on the right.

Figura 4.13: Configuração da presa.

Para cada treinamento é possível configurar a quantidade de ciclos, de experimentos, de avaliações e de passos por avaliação. O ciclo determina a quantidade de ciclos evolucionários utilizados no treinamento, isto é, quantas vezes as populações de neurônios evoluam, através das aplicações dos operadores genéticos. Os experimentos determinam quantas redes são montadas em cada ciclo evolucionário. Quanto maior a quantidade de redes montadas, maior será a quantidade de neurônios avaliados em cada população e mais precisa será a aptidão média de cada neurônio. A aptidão média de cada neurônio é dada pela razão entre a aptidão total do neurônio pela quantidade de redes em que ele participou. A avaliação indica quantos cenários serão criados para avaliar a rede montada no experimento atual. Por fim, os passos indicam quantos movimentos os agentes devem fazer antes de terem suas estratégias avaliadas. Vide Figura 4.14.



Figura 4.14: Configuração do treinamento.

Para o mundo é possível configurar o tipo (limitado ou toroidal), a dimensão (altura e largura), grau de dificuldade (fácil: basta que um agente pegue a presa, difícil: os agentes precisam encurralar a presa) e regras de posicionamento (é permitido ou não mais de um agente na mesma célula). Observe as configurações possíveis na Figura 4.15. Todos estes fatores contribuem para a configuração da complexidade da tarefa.

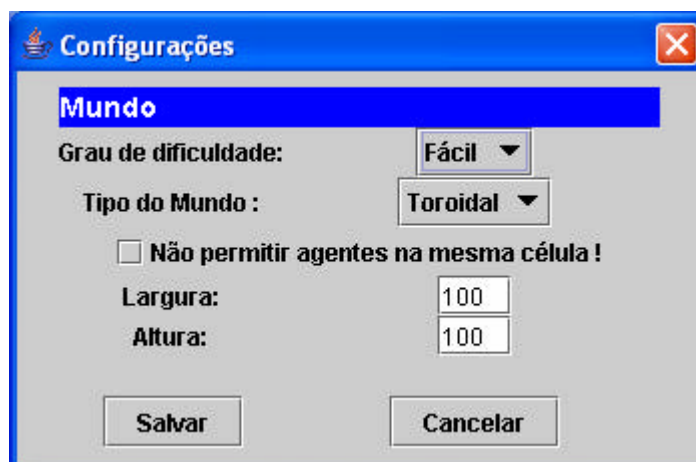


Figura 4.15: Configurações do mundo.

O menu Execução, permite que se controle a execução do treinamento, a execução do algoritmo de *Delta-coding* e a execução da estratégia aprendida. A Figura 4.16 ilustra o menu Execução.



Figura 4.16: Menu Execução.

4.5.3 Visualização das estratégias

A janela principal da ferramenta possibilita a fácil visualização das estratégias aprendidas pelos agentes. Vide Figura 4.17.

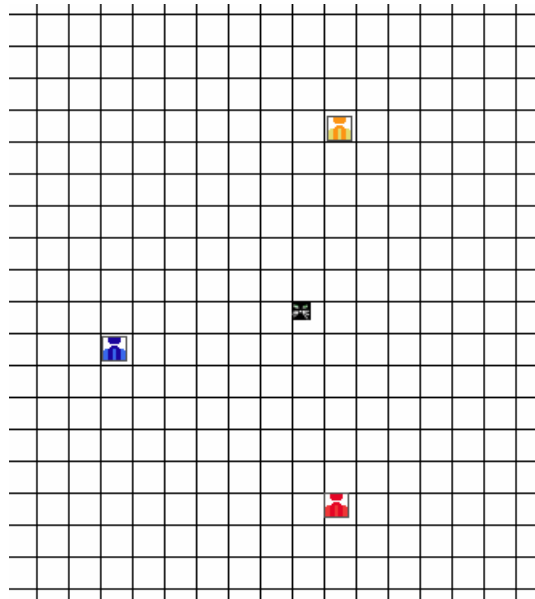


Figura 4.17: Visualização das estratégias aprendidas.

Neste capítulo, foram apresentadas as partes constituintes do modelo de coordenação proposto, dentre as quais, pode-se destacar: o algoritmo evolucionário, a codificação da rede neural em cromossomos, os operadores genéticos considerados e a topologia de rede utilizada. Adicionalmente, apresentou-se o domínio de aplicação onde o modelo foi testado, o diagrama de classes da implementação do modelo e o ambiente de simulação. No próximo capítulo, serão apresentados diversos experimentos realizados tanto para a validação do modelo quanto para a identificação dos valores ideais para os parâmetros do modelo, considerando-se a tarefa da presa-predador.

5 EXPERIMENTOS

Foram realizados dois conjuntos de experimentos de forma a avaliar o modelo proposto. No primeiro conjunto de experimentos, descritos na seção 5.1, foi realizada uma comparação entre o modelo proposto e outros modelos existentes na literatura. No segundo conjunto de experimentos, foi realizado um estudo sobre valores satisfatórios para os diversos parâmetros do modelo.

Em cada experimento descrito neste capítulo, o treinamento foi realizado de forma incremental. Os predadores eram inicialmente treinados com a presa estacionária. Em cada fase subsequente do treinamento, a velocidade da presa era incrementada de 0,1 até que atingisse a velocidade de 1,0. Em cada troca de velocidade da presa, *Delta-Coding* era aplicado.

5.1 Experimentos Comparativos

Yong(2001) utilizou *ESP/Delta-Coding* para evoluir o comportamento dos predadores, em uma tarefa de perseguição-evasão, onde, a presa era considerada capturada quando um dos predadores tocassem nela.

A estratégia padrão utilizada pela presa nos experimentos de Yong(2001) era fugir do predador mais próximo. Os predadores de Yong(2001), assim como os predadores que tiveram seus comportamentos evoluídos com o modelo proposto no presente trabalho, são capazes de capturar sempre a presa desde que o comportamento utilizado por ela, em tempo de execução, seja o mesmo usado no treinamento.

Yong(2001) realizou experimentos a fim de avaliar a adaptabilidade dos comportamentos dos predadores à mudança de estratégia da presa, em tempo de execução. Nos experimentos reportados, os predadores foram treinados para capturar uma presa cuja estratégia era fugir do predador mais próximo. Em tempo de execução, a estratégia da presa era alterada para se movimentar sempre à direita.

Neste trabalho, foi realizado um experimento similar. Utilizou-se uma configuração para treinamento parecida com a que foi adota por Yong(2001) no treinamento de seus agentes predadores. A comparação das configurações adotadas pode ser visualizada na Tabela 5.1. O tempo gasto para a realização de um experimento completo (treinamento inicial com presa estacionária e dez treinamentos com *Delta-Coding* incrementando-se a velocidade da presa a cada novo treinamento) durou cerca de 10 horas. A máquina utilizada para o treinamento possui um processador AMD Athlon™ Xp 2200+ 1.80GHz e 1.00GB de memória RAM.

Tabela 5.1: Comparação das configurações adotadas para treinamento dos agentes.

	Configuração Utilizada em Yong (2001)	Configuração do Modelo proposto
Tamanho de cada População de Neurônios	100	100
Elitismo	50	20
Tamanho (Máximo) da Camada Oculta	10	9
Quantidade de ciclos evolucionários	400	400
Quantidade de avaliações/cenários por ciclo	6	6
Quantidade de Experimentações (redes formadas por ciclo evolucionário)	1000	1000
Dimensão do Mundo	(100,100)	(100,100)

As configurações adotadas no presente trabalho divergiram sutilmente em dois parâmetros: o elitismo e a quantidade de neurônios presentes na camada oculta de cada agente. No algoritmo evolucionário utilizado por Yong(2001), em cada ciclo de evolução, ele substituía apenas os 50% piores neurônios de cada população, usando, desta forma, uma alta taxa de elitismo. No experimento realizado no presente trabalho, optou-se por uma taxa de elitismo de 20%.

Em Yong(2001), a quantidade de neurônios presente na camada oculta era fixa e igual a 10. Nos comportamentos evoluídos neste experimento, utilizando o modelo proposto, existiam nove neurônios presentes na camada oculta de cada predador.

A Tabela 5.2 compara os resultados do modelo proposto com o modelo utilizado por Yong(2001). Executou-se 10 simulações para os comportamentos evoluídos. Em cada simulação, a presa iniciava em uma posição diferente no mundo, sempre com uma distância de, no mínimo, dez casas do predador mais próximo.

Tabela 5.2: Comparação de resultados com o ESP.

	Taxa de captura quando a presa troca de comportamento em tempo de execução
ESP	14,5 %
Modelo Proposto	80 %

Não foram encontradas referências da utilização de SANE ou NEAT para problemas similares aos utilizados neste trabalho, no domínio da presa-predador.

5.2 Experimentos Variando Diversos Parâmetros

Na abordagem neuro-evolucionária, são muitos os parâmetros a serem configurados. Realizou-se uma série de experimentos a fim de avaliar quais os melhores parâmetros para a tarefa proposta.

Foi considerado um valor padrão para cada parâmetro. Na maioria dos experimentos, realizados nesta seção, alterou-se apenas um dos parâmetros.

Os parâmetros podem ser categorizados como parâmetros dos predadores, parâmetros da presa, parâmetros do mundo e parâmetros de treinamento.

Os valores adotados foram obtidos através de experimentos preliminares. Para os parâmetros dos predadores considerou-se como padrão, os valores abaixo:

- Quantidade máxima de neurônio na camada oculta: 10
- Entrada da rede neural do predador: sinal do *offset* entre o predador atual e a presa.
- Probabilidade de mutação do neurônio: 0.2
- Elitismo na população de cada neurônio: 2
- Tamanho da população de cada neurônio: 30

Os valores adotados, por padrão, para os parâmetros da presa foram:

- Posição Inicial: (15,15)
- Algoritmo: Fugir do predador mais próximo

Os valores adotados, por padrão, para os parâmetros do treinamento foram:

- Quantidade de ciclos evolucionários: 300
- Quantidade de experimentações por ciclo: 50

Experimentação, neste contexto, significa quantidade de redes que são testadas por ciclo evolucionário. Em outras palavras, em cada experimentação, novos neurônios são escolhidos, de cada população de neurônio, para montar a rede neural do agente.

- Quantidade de avaliações por tentativa: 20
- Quantidade de passos por avaliação: 15

Foram assumidas as seguintes suposições sobre o mundo:

- A dimensão do mundo foi: (30,30)
- É suficiente que um agente toque na presa para que ela seja considerada capturada
- O mundo é toroidal

- É permitido que quaisquer dois agentes ocupem a mesma célula
- Os predadores sempre iniciam no canto superior esquerdo do mundo.

5.2.1 Entrada do Agente

Foram avaliados seis tipos de entrada para os predadores:

Pode-se categorizar a percepção que o agente tem do mundo como parcial ou total. Na percepção parcial, o agente recebe informações relativas a ele a presa. Na percepção total, o agente recebe informações relativas a ele e ao resto do grupo de predadores:

Entradas de tamanho dois:

- Sinal do *offset* [x,y] entre o agente atual e a presa
- *Offset* [x,y] entre o agente e a presa

Entradas de tamanho quatro:

- Coordenada absolutas [x_a,y_a,x_p,y_p] do agente atual e da presa

Entradas de tamanho n:

- Sinal do *offset* [x,y] entre o agente atual e a presa e o agente atual e os outros n-1 predadores
- *Offset* [x,y] entre o agente atual e a presa e o agente atual e os outros n-1 predadores

Entradas de tamanho 2*(n+1)

- Coordenada absolutas [x_{a1},y_{a2},...,x_{an},y_{an},x_p,y_p] de todos os agentes do sistema

Percepção Parcial		
Sinal do <i>Offset</i> (1)	<i>Offset</i> (2)	Coordenadas (3)
Percepção Total		
Sinal do <i>Offset</i> (4)	<i>Offset</i> (5)	Coordenadas (6)

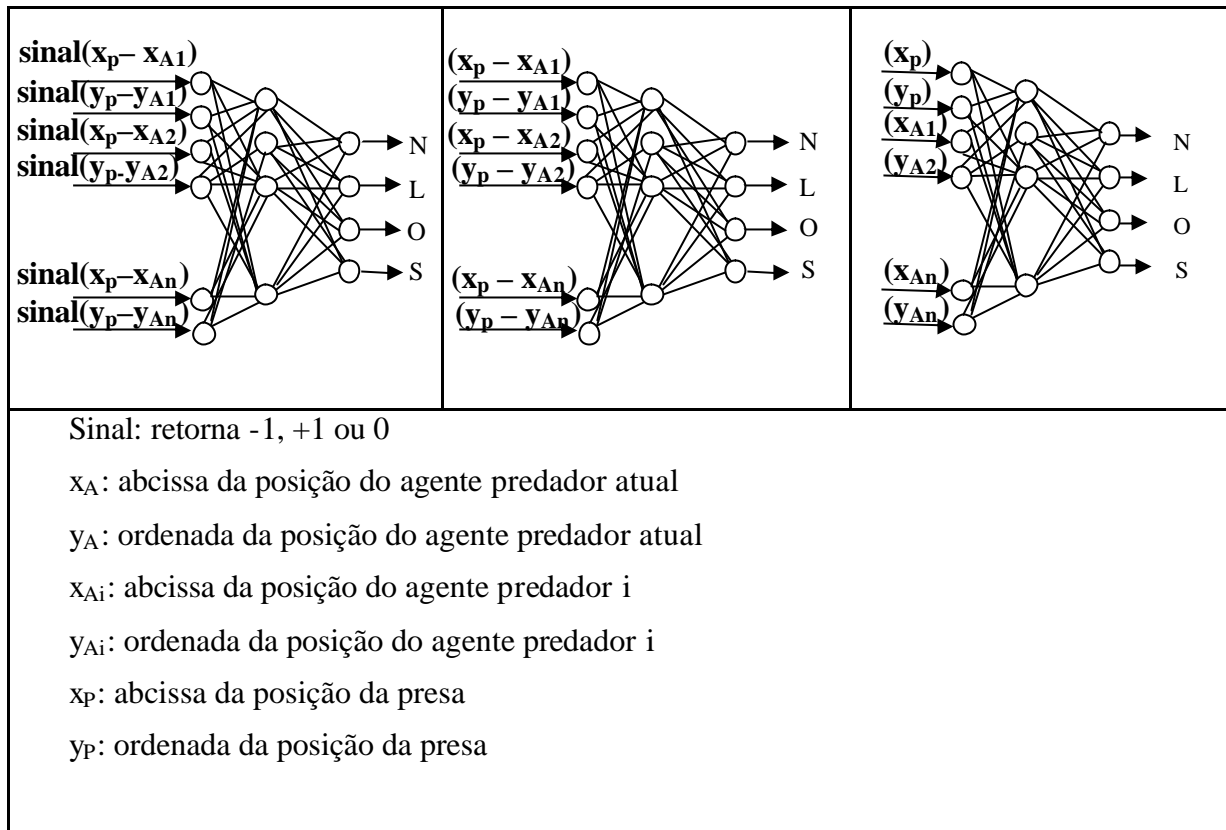


Figura 5.1: Entradas das Redes Neurais dos agentes.

Os agentes só se mostraram capazes de aprender como capturar a presa quando as entradas correspondiam ao sinal do *offset*. Informações envolvendo distâncias e posições absolutas são entradas mais complexas e dificultam o aprendizado da tarefa.

Para este experimento, não foi utilizado *Delta-Coding*, apenas, o treinamento inicial com a presa estacionária. O time com a percepção parcial do mundo, conseguia alcançar a presa com uma quantidade menor de passos. Observe o resultado de 10 simulações consecutivas, cada simulação constituída de um novo treinamento e execução, na Tabela 5.3:

Tabela 5.3: Passos para captura da presa.

Execução	Time com Percepção Total do Mundo	Time com percepção parcial do Mundo
1	35	29
2	33	29
3	43	29
4	53	31
5	37	33
6	35	31

7	34	31
8	41	31
9	55	29
10	47	31
Média	41.3	30.4

5.2.2 Quantidade de Neurônios na Camada Oculta

Foi avaliada a taxa de captura da presa, em função da quantidade máxima de neurônios na camada oculta. Foram realizadas 10 execuções para cada quantidade máxima de neurônio na camada oculta. O resultado destas execuções pode ser visualizado na Tabela 5.4.

Tabela 5.4: Taxa de captura por tamanho da camada oculta.

Quantidade de Neurônios	Taxa de Captura (%)
3	80
4	90
5	100
6	100
7	90
8	100
9	90
10	90

Avaliou-se o tempo necessário para treinamento em função do tamanho da camada oculta. Para este experimento, foi considerado apenas o treinamento inicial com a presa estacionária, sem *Delta-Coding*.

Na Tabela 5.5, mostra-se o tempo de treinamento por quantidade de neurônios na camada oculta. A camada oculta com menor quantidade de neurônios (3) possui um tempo de treinamento 20% menor que o da camada com a maior quantidade de neurônios (10).

Tabela 5.5: Tempo de treinamento por tamanho da camada oculta.

Quantidade de Neurônios	Tempo Médio de Treinamento (ms)
3	56367
4	59649
5	60955
6	59917
7	60528
8	62840
9	67503
10	70568

5.2.3 Quantidade de ciclos evolucionários

Neste conjunto de experimentos, buscou-se identificar a quantidade adequada de ciclos evolucionários.

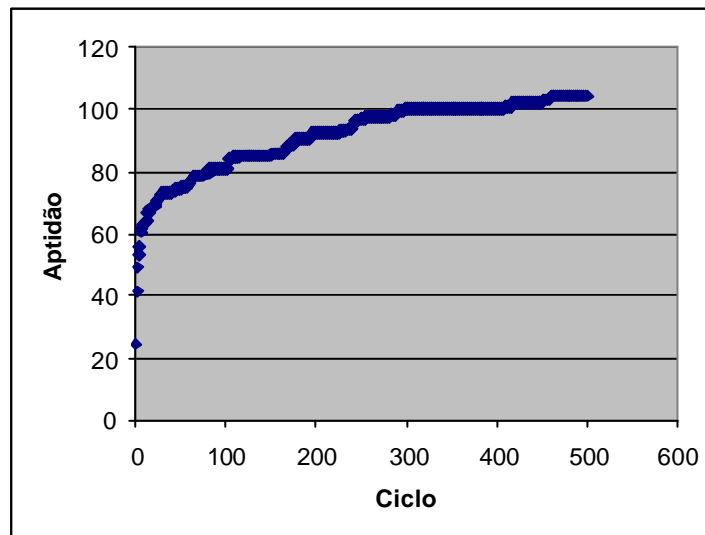


Figura 5.2: Média das melhores aptidões dos agentes por ciclo evolucionário. Time constituído por 3 agentes.

Foram realizadas algumas execuções com um número maior de agentes (4). Os valores de aptidão foram maiores. Em ambos os experimentos, as aptidões dos agentes se estabilizaram a partir do ciclo 300. Vide Figuras 5.2 e 5.3.

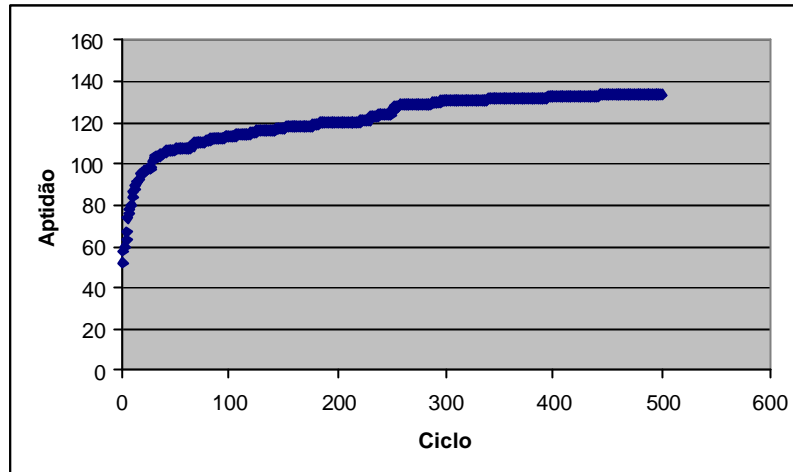


Figura 5.3: Média das melhores aptidões dos agentes por ciclo evolucionário. Time constituído por 4 agentes.

5.2.4 Função de Aptidão

A função de aptidão escolhida é um dos fatores que mais influenciam a aprendizagem do agente, pois, é o que indica quão bom é o comportamento do agente.

As funções de aptidão podem variar muito em termos de complexidade e quantidade de parâmetros levados em consideração.

Utilizou-se uma função de aptidão simples. A aptidão é inversamente proporcional à distância final dos predadores para a presa.

Foram realizadas algumas execuções com variações desta função. Porém, os resultados não foram satisfatórios.

5.2.5 Mundo toroidal x limitado

Realizou-se diversos experimentos considerando um mundo limitado. Inicialmente, esta tarefa parecia mais fácil, pois, os predadores teriam a “ajuda” de obstáculos (limite do mundo) na tarefa de captura da presa. Contudo, esta tarefa se mostrou mais difícil, pois, em nenhuma das simulações realizadas, a presa foi capturada com o mundo limitado. A causa deste insucesso pode estar relacionada com a mudança abrupta de resposta que o ambiente dá como retorno para os agentes. Por exemplo, o agente recebe um retorno positivo ao caminhar para a direita. Contudo, ao barrar na parede, o retorno fica “de repente” estagnado, pela visão do agente. Uma hipótese é que se o agente recebesse mais informações sobre o mundo, como, por exemplo, o tamanho do mesmo e “recordasse” de suas ações passadas, ele seria capaz de aprender a tarefa.

5.2.6 Grau de dificuldade da tarefa-alvo

A tarefa alvo considerada foi a captura da presa em um mundo toroidal, onde a presa se movimenta na mesma velocidade que os predadores. A presa é considerada capturada quando um dos predadores toca nela.

Em nenhuma das simulações executadas, a presa foi capturada quando treinada diretamente na tarefa alvo. Partir de uma tarefa simples, isto é, capturar uma presa

estacionária e tornar a tarefa progressivamente mais complexa, utilizando *Delta-Coding* para os treinamentos subsequentes, foi essencial para o aprendizado da tarefa alvo. O grau de dificuldade da tarefa era incrementado com o aumento da velocidade da presa.

Quando foi imposta a restrição de que a presa só seria considerada capturada quando todos os agentes a cercassem, não foi obtido sucesso em nenhuma das simulações realizadas.

6 CONCLUSÕES

Em ambientes dinâmicos e complexos, a política ótima de coordenação não pode ser derivada analiticamente, mas deve ser aprendida através da interação direta com o ambiente. Neste trabalho, propomos um modelo de coordenação baseado em neuro-evolução, que, de acordo com vários pesquisadores (MORIARTY, 1997; MCQUESTEN, 2002; YONG, 2001; STANLEY, 2002) é um dos métodos mais promissores para aprendizagem em ambientes estocásticos.

Para testar o modelo, utilizou-se a tarefa da presa-predador, um caso especial de uma classe de problemas conhecidos como perseguição-evasão. Nesta tarefa, uma presa tem por objetivo fugir de um ou mais predadores, enquanto que os predadores precisam se coordenar de forma a capturá-la.

A tarefa da presa-predador é praticamente um *benchmark* para modelos de coordenação, por basicamente duas razões: primeiro por simularem situações que ocorrem no mundo real e segundo porque, a depender da configuração do ambiente e da estratégia de fuga da presa, exige elevado grau de coordenação entre os predadores.

Foi realizado um estudo sobre os modelos de neuro-evolução existentes na literatura, e proposto uma extensão do método neuro-evolutivo conhecido como ESP. Ao contrário do ESP, que trabalha com uma topologia fixa para as redes neurais dos agentes, o nosso modelo permite que o algoritmo evolucionário encontre, em tempo de treinamento, a topologia da rede neural de cada agente.

Além de propiciar flexibilidade, o nosso modelo possibilita uma otimização do tempo necessário para o treinamento. Nossos experimentos mostraram que a quantidade de neurônios presentes na camada oculta de cada agente, no final do treinamento, é, freqüentemente, inferior à quantidade máxima especificada. Adicionalmente, os experimentos também mostraram que o tempo de treinamento aumenta proporcionalmente com o tamanho da camada oculta. Nos experimentos foi constatado que o tempo de treinamento quando o agente possui três neurônios na camada oculta é 20% menor que o tempo necessário para o treinamento com 10 neurônios na camada oculta. Este último resultado já era esperado, uma vez que uma quantidade maior de neurônios implica em maior complexidade computacional.

A variação na topologia da rede dos agentes, em tempo de treinamento, levou a um outro resultado: a construção de um time heterogêneo de agentes, no que diz respeito à topologia da rede neural de cada agente, o que é mais coerente com o papel diferenciado que cada agente possui no grupo. Considere, por exemplo, a tarefa mais difícil para a qual obteve-se sucesso com o modelo proposto: um mundo toroidal, onde a presa se movimenta na mesma velocidade que os predadores, fugindo sempre do predador mais próximo. Se todos os agentes possuíssem o papel de correr atrás da presa, eles nunca

iriam conseguir capturá-la, pois, todos os agentes (presa e predadores) ficariam girando na mesma direção ao redor do mundo. Desta forma, os agentes precisam evoluir comportamentos diferenciados, isto é, parte dos agentes persegue a presa enquanto outra parte impede que a presa avance em determinada direção.

Os melhores comportamentos foram obtidos através de um processo de aprendizagem incremental, isto é, começando-se com uma tarefa simples: capturar uma presa estacionária e finalizando com uma tarefa mais complexa, por exemplo, capturar uma presa que se movimenta na mesma velocidade que os predadores, fugindo sempre do predador mais próximo.

A série de experimentos realizados neste trabalho mostra como a escolha dos valores para os diversos parâmetros do modelo influencia o resultado do processo de aprendizagem. Abrindo espaço para uma pesquisa sobre um modelo auto-ajustável de parâmetros por tipo de tarefa. Incluindo variações de parâmetros não avaliados neste trabalho, como, por exemplo, o algoritmo utilizado para seleção e outras funções e aptidão.

Nos experimentos apresentados no capítulo anterior, mostrou-se que o modelo proposto tem boa capacidade de adaptação em função da alteração no comportamento da presa. Em trabalhos futuros, pretendemos desenvolver um modelo em que os agentes sejam capazes de compreender possíveis alterações no mundo e alterar seu comportamento, em tempo de execução, de forma a suportar estas alterações sem a necessidade de um novo treinamento. Uma forma possível de realizar tal procedimento seria fazer com que os agentes armazenassem as várias estratégias aprendidas para as diversas variações do mundo e em tempo de execução, identificassem qual estratégia melhor se aplica à percepção atual do mundo. Essa identificação poderia ser realizada por uma rede neural a parte.

O ambiente de simulação desenvolvido, como parte deste trabalho, tanto facilita a configuração dos diversos parâmetros do modelo como permite visualizar graficamente as estratégias evoluídas para a tarefa da presa-predador.

Pretende-se estender o ambiente de simulação de forma que seja capaz de suportar o treinamento e simulação do comportamento dos agentes em outros domínios de aplicação, além do domínio da presa-predador, utilizado neste trabalho.

REFERÊNCIAS

AGOGINO, A.; MIIKKULAINEN, R. Efficient Allele Fitness Assignment with Self-organizing Multi-agent System. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2004. **Genetic and Evolutionary Computation**. New York: Springer-Verlag, 2004.

ALDEN, M.; KESTEREN, A; MIIKKULAINEN, R. Eugenic Evolution Utilizing A Domain Model. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2002. **Genetic and Evolutionary Computation**. New York: Springer-Verlag, 2002, p. 279-286.

BALAKRISHNAN, K; HONAVAR, V. **Evolutionary Design of Neural Architectures – A Preliminary Taxonomy and Guide to Literature**. Ames, Iowa: Department of Computer Science, Iowa State University, 1995.

BARRETO, A. M. S. **Algoritmo Genético dos Mínimos Quadrados Ortogonal para o Treinamento de Redes RBF**. 2003. Dissertação (Mestrado em Engenharia Civil) - COPPE, UFRJ, Rio de Janeiro.

BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. **IEEE Transactions on Systems, Man, and Cybernetics**, [S.l.], v. SMC-13, p. 834-846, 1983.

BARTO, A. G.; SUTTON, R. S.; WATKINS, C. J. C. H. Learning and sequential decision making. In GABRIEL, M.; MOORE, J. W. (Ed.). **Learning and Computational Neuroscience**. Cambridge, MA: MIT Press, 1990.

BELEW, R. K. Interposing an ontogenic model between genetic algorithms and neural networks. In: HAN-SON, S. J.; COWAN. J. D.; GILES; C. L. **Advances in Neural Information Processing Systems (NIPS)**. San Mateo: Morgan Kaufmann, 1993.

BERENJI, H. R.; VENGEROV, D. Advantages of Cooperation Between Reinforcement Learning Agents in Difficult Stochastic Problems. In: IEEE INTERNATIONAL

CONFERENCE ON FUZZY SYSTEMS, FUZZ-IEEE, 9., 2000. **Proceedings...** [S.l.]: IEEE, 2000.

BRUCE, J.; MIIKKULAINEN, R. Evolving Populations Of Expert Neural Networks. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2001. **Genetic and Evolutionary Computation: proceedings**. San Francisco, CA: Kaufmann, 2001, p. 251-257.

BRYANT, B. D.; MIIKKULAINEN, R. Neuroevolution for Adaptive Teams. In: CONGRESS ON EVOLUTIONARY COMPUTATION, CEC, 2003, Camberra. **Proceedins...** Camberra, Australia: [s.n.], 2003.

CLIFF, D.; MILLER, G Co-evolution of Pursuit and Evasion II: Simulation Methods and Results. In: INTERNATIONAL CONFERENCE ON SIMULATION OF ADAPTATIVE BEHAVIOUR, SAB, 4., 1996. **Proceedings...** [S.l.]: SAB, 1996.

FAN, J.; LAU, R.; MIIKKULAINEN, R. Utilizing Domain Knowledge in Neuroevolution. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ICML, 20., 2003. **Proceedings...** Washington, DC: ICML, 2003

FAHLMAN, S. E.; LEBIERE, C. The cascade-correlation learning architecture. In: **Advances in Neural Information Processing Systems**. San Mateo: Morgan Kaufmann, 1990.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. Reading, MA: Addison-Wesley, 1989.

GOMEZ, F.; MIIKKULAIEN, R. Incremental evolution of complex general behavior. **Adaptive Behavior**, [S.l.], v. 5, p. 317-342, 1997.

GOMEZ, F. J. **Robust Non-Linear Control through Neuroevolution**. 2003. PhD Thesis. Department of Computer Sciences, The University of Texas, Austin.

GOMEZ, F. J.; MIIKKULAINEN, R. Transfer of Neuroevolved Controllers in Unstable Domains. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2004. **Genetic and Evolutionary Computation**. New York: Springer-Verlag, 2004.

GRASEMANN, U.; MIIKKULAINEN, R. Evolving Wavelets using a Coevolutionary Genetic Algorithm and Lifting. In: GENETIC AND EVOLUTIONARY

COMPUTATION CONFERENCE, GECCO, 2004. **Genetic and Evolutionary Computation**. New York: Springer-Verlag, 2004.

GREFENSTETTE, J. J.; RAMSEY, C. L.; SCHULTZ, A. C. Learning sequential decision rules using simulation models and competition. **Machine Learning**, [S.l.], v.5, p. 355-381, 1990.

GRUAU, F.; WHITLEY, D.; PYEATT, L. **A comparison between cellular encoding and direct encoding for genetic neural networks**. Genetic Programming 1996: proceedings of the First Annual Conference. Cambridge, MA: MIT Press, 1996. p. 81-89.

HAYNES, T.; SEN, S. Evolving Behavioral Strategies in Predators and Prey. In: **Adaptation and Learning in Multiagent Systems**. Berlin: Springer-Verlag, 1996.

HAYNES, T.; WAINWRIGHT, R.; SEN S. Evolving Cooperation Strategies. In: INTERNATIONAL CONFERENCE ON MULTIAGENT SYSTEMS, ICMAS, 1., 1995, San Francisco. **Proceedings...** San Francisco, CA: MIT Press, 1995.

HERTZ, J.; KROGH, A.; PALMER, R. G. **Introduction to the Theory of Neural Computation**. Reading, MA: Addison-Wesley, 1991.

HOLLAND, J. H. **Adaptation in natural artificial systems**. Ann Arbor: University of Michigan Press, 1975.

O'HARE, G.M.P.; JENNINGS, N. R. **Foundations of Artificial Intelligence**. [S.l.]: John Wiley & Sons, Inc, 1996.

KAHN, J.; KATZ, R.H.; PISTER, K. Emerging Challenges: Mobile Networking for "Smart Dust". **J. Comm. Networks**, [S.l.], p. 188-196, Sept. 2000.

KAELBLING, L.; LITTMAN, M.; MOORE, A. Reinforcement Learning: A Survey. **Journal of Artificial Intelligence Research**, [S.l.], v.4, p. 237-285, May 1996.

LESSER, V. R. Multiagent Systems: An Emerging Subdiscipline of AI. **ACM Computing Surveys**, New York, v. 27, n. 3, Sept. 1995.

MALONE, T. W.; CROWSTON, K. The Interdisciplinary Study of Coordination. **ACM Computing Surveys**, New York, v.26, n.1, p. 87-119, 1994.

MCQUESTEN, P. **Cultural Enhancement of Neuroevolution**. 2002. Ph.D. Thesis. Department of Computer Sciences, The University of Texas, Austin, TX. (Tech Report AI-02-295).

MILLER, G.; CLIFF, D. **Co-evolution of pursuit and evasion i**: Biological and game-theoretic foundations. Brighton, UK: School of Cognitive and Computing Sciences, University of Sussex, 1994.

MINSKY, M. Steps toward artificial intelligence. In: FEIGENBAUM, E. A.; FELDMAN, J. A. (Ed.). **Computers and Thought**. New York: McGraw-Hill, 1963, p. 406–450.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, MA: MIT Press, 1996.

MONTANA, D. J.; DAVIS, L. Training feedforward neural networks using genetic algorithms. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 11., 1989, San Mateo. **Proceedings...**CA: Morgan Kaufmann, 1989. p. 762-767.

MORIARTY, D.; MIKKULAINEN, R. Efficient Reinforcement Learning through Symbiotic Evolution. **Machine Learning**, [S.l.], v. 22, p. 11-33, 1996.

MORIARTY, D. E. **Symbiotic Evolution Of Neural Networks In Sequential Decision Tasks**. 1997. Ph.D. Dissertation, Department of Computer Sciences, University of Texas, Austin.

NOLFI, S.; PARISI, D. Desired answers do not correspond ecological neural networks. **Neural Processing Letters**, [S.l.], n. 2, p. 1–4, 1994.

NSF (National Science Foundation). **Review Panel for Research on Coordination Theory and Technology**. 1989, A report by NSF-IRIS. NSSF Forms and Publication Unit, National Science Foundation, Washington, D.C.

PARDOE, D.; RYOO, M.; MIKKULAINEN, R. Evolving Neural Network Ensembles for Control Problems. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2005. **Genetic and Evolutionary Computation: proceedings**. [S.l.: s.n.], 2005. p. 1379-1384.

POTTER, M. A. A genetic cascade-correlation learning algorithm. In: INTERNATIONAL WORKSHOP ON COMBINATIONS OF GENETIC ALGORITHMS AND NEURAL NETWORKS, COGANN, 1992, Baltimore, MD. **Proceedings...**[S.l.]: IEEE Computer Society Press, 1992. p. 123-133.

REISINGER, J.; STANLEY, K. O.; MIIKKULAINEN, R. Evolving Reusable Neural Modules. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2004. **Genetic and Evolutionary Computation**. New York: Springer-Verlag, 2004.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. In: **Parallel Distributed Processing**: Exploration in the Microstructure of Cognition. Cambridge MA: MIT Press, 1986.

SINGH, B.; REIN, G. L. **Role Interaction Nets (RINs)**: A Process Definition Formalism. [S.l.]: MCC, 1992. (Technical Report n. CT-083-92).

STANLEY, K. O.; MIIKKULAINEN, R. Efficient Reinforcement Learning through Evolving Neural Network Topologies. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2002. **Genetic and Evolutionary Computation**: proceedings. San Francisco, CA: Morgan Kaufmann, 2002. (Winner of the Best Paper Award in Genetic Algorithms).

STANLEY, K. O.; MIIKKULAINEN, R. Efficient Evolution Of Neural Network Topologies. In: CONGRESS ON EVOLUTIONARY COMPUTATION, CEC, 2002, Piscataway. **Proceedings...** NJ: IEEE, 2002.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving Neural Networks Through Augmenting Topologies. **Evolutionary Computation**, [S.l.], v. 10, n. 2, p. 99-127, 2002.

STANLEY, K. O.; MIIKKULAINEN, R. A Taxonomy for Artificial Embryogeny. **Artificial Life**, [S.l.], v. 9, n. 2, p. 93-130, 2003.

STANLEY, K. O.; MIIKKULAINEN, R. Achieving High-Level Functionality through Evolutionary Complexification. In: AAI-2003 SPRING SYMPOSIUM ON COMPUTATIONAL SYNTHESIS, 2003, Stanford. **Proceedings...**CA: AAI Press, 2003.

STANLEY, K. O.; BRYANT, B. D.; MIIKKULAINEN, R. Evolving Adaptive Neural Networks with and Without Adaptive Synapses. In: CONGRESS ON

EVOLUTIONARY COMPUTATION, CEC, 2003. **Proceedings...** Camberra, Australia, 2003. p. 2557-2564.

STANLEY, K. O.; MIIKKULAINEN, R. Competitive Coevolution through Evolutionary Complexification. **Journal of Artificial Intelligence Research**, [S.l.], v.21, p. 63-100, 2004.

STANLEY, K.; BRYANT, B.; MIIKKULAINEN. Evolving Neural Networks Agents in the NERO Video Game. In: IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE AND GAMES, CIG, 2005, Piscataway. **Proceedings...** NJ: IEEE, 2005.

STANLEY, K. et al. Neuroevolution of an Automobile Crash Warning System. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2005, Washington, D.C. USA. **Genetic and Evolutionary Computation: proceedings**. [S.l.:s.n.], 2005, p. 1977-1984.

SUTTON, R. S. Learning to predict by the methods of temporal differences. **Machine Learning**, [S.l.], v. 3, p. 9-44, 1988.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. Cambridge, MA: MIT Press, 1998.

TEWS, A.; LISTER, R. **Self-Organisation in a Simple Pursuit Game**. Disponível em: <<http://life.csu.edu.au/complex/ci/vol6/tews/>>. Acesso em: set. 2005.

THOMPSON, J. **Organizations in action: social sciences bases of administrative theory**. New York: McGraw-Hill Book Co, 1967.

VENGEROV, D.; BERENJI, R.; VENGEROV, A. Emergent Coordination Among Fuzzy Reinforcement Learning Agents. In: LOIA, V. (Ed.). **Soft Computing Agents: A New Perspective for Dynamic Information Systems**. Amsterdam, The Netherlands: IOS Press, 2003.

YANNAKAKIS, G.; LEVINE, J.; HALLAM, J. An Evolutionary Approach for Interactive Computer Games. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, 2004, Portland, Oregon, USA. **Proceedings...** [S.l.:s.n.], 2004.

YANNAKAKIS, G.; HALLAM, J. Evolving Opponents for Interesting Interactive Computer Games. In: INTERNATIONAL CONFERENCE ON THE SIMULATION

OF ADAPTIVE BEHAVIOR, SAB, 8., 2004. **From Animals to Animats 8:** proceedings. Cambridge: MIT, 2004. p. 499-508.

YONG, C.; MIIKKULAIEN, R **Cooperative Coevolution of Multi-Agent Systems.** Austin: Department of Computer Sciences, The University of Texas at Austin, 2001. (Technical Report AI01-287).

WATKINS, C. J. C. H. **Learning from Delayed Rewards.** 1989. PhD thesis, University of Cambridge, England.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, [S.l.], v. 8, n. 3, p. 279-292, 1992.

WHITESON, S. et al. Automatic Feature Selection in Neuroevolution. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, GECCO, 2005, Washington, D.C. USA. **Genetic and Evolutionary Computation:** proceedings. [S.l.:s.n.], 2005. p. 1225-1232.

WHITLEY, D.; MATHIAS, K.; FITZHORN, P. Delta-coding: An iterative search strategy for genetic algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 4., 1991. **Proceedings...** Los Altos, CA: Morgan Kaufmann, 1991.