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

Inteligência Artificial Popperiana

por

JOÃO PAULO SCHWARZ SCHÜLER

Dissertação submetida à
avaliação, como requisito parcial para a
obtenção do grau de Mestre em
Ciência da Computação

Prof. Luis Otávio Campos Alvares
Orientador

Porto Alegre, dezembro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Schüler, João Paulo Schwarz

Inteligência Artificial Popperiana / por João Paulo Schwarz Schüler. - Porto Alegre: PPGC da UFRGS, 2002.

105 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de pós-graduação, Porto Alegre, BR – RS, 2002. Orientador: Alvares, Luis Otávio Campos.

1. Inteligência Artificial. 2. Agentes. 3. Criaturas popperianas. 4. Redes neurais. I Alvares, Luis Otávio Campos. II Título

AGRADECIMENTOS

Agradeço à CAPES por ter financiado a minha pesquisa, ao meu orientador por sua grande atenção dedicada ao presente trabalho, ao meu pai e a todos os familiares que contribuíram para a realização desta pesquisa.

Sumário

Lista de Abreviaturas

CNM	combinatorial neural model
DFD	diagrama de fluxo de dados
DNA	deoxiribonucleic acid
NARS	non-axiomatic reasoning system
UCP	unidade central de processamento
TP	tamanho da procura
TR	tabela de relação
WAE	Webmind AI Engine

Lista de Figuras

Lista de Tabelas

Resumo

A inteligência tem sido estudada como fruto de evolução biológica. Nas últimas centenas de milhões de anos, a inteligência tem evoluído juntamente com a biologia. Essa conclusão pode ser obtida ao analisar o comportamento das criaturas que emergiram assim como a sua capacidade de armazenar e processar informação. A evolução gerou criaturas possuidoras de cérebros com grande poder de adaptação. Partindo-se do pressuposto que a inteligência humana é resultado de um processo evolutivo paulatino que ocorreu ao longo de milhões de anos, faz sentido tentar repetir os mesmos passos dados ao longo da evolução da inteligência artificialmente. A evolução oferece uma rota que vai desde tipos de mentes simples até tipos de mentes mais complexas apresentando um caminho de características e capacidades que evoluíram ao longo do tempo. No presente trabalho, acredita-se que esse caminho seguido pela evolução é uma boa fonte de inspiração para a geração de inteligência artificial. De acordo com Dennett, um tipo de mente que apareceu ao longo da evolução é a mente popperiana que aprende as regras do ambiente e tem a capacidade de imaginar ou planejar estados futuros permitindo que ela se adapte com facilidade a novas e inesperadas situações. Sendo assim, modela-se e implementa-se um agente popperiano capaz de aprender as regras do seu ambiente e planejar ações futuras baseando-se no seu aprendizado. Por fim, são implementados dois protótipos de agentes popperianos para resolver problemas distintos e observa-se a capacidade dos agentes popperianos em se adaptar às condições do seu meio para alcançar seus objetivos.

Palavras-chaves: inteligência artificial, agentes, criaturas popperianas, redes neurais.

TITLE: “POPPERIAN ARTIFICIAL INTELLIGENCE”

Abstract

Intelligence has been science's subject of study as a result of biological evolution. In the last hundred million years, intelligence has evolved together with biology. One may get to this conclusion by analyzing the behavior of creatures that have surged as well as their ability to store and process information. Evolution has generated creatures having brains with great adaptive capacity. Assuming that human intelligence has evolved through a long and slow process that took place along several million years, it would make sense to try and replicate artificially the same steps taken in this process. Evolution shows us a path that goes from the simplest to the most complex minds presenting the features and abilities that have evolved along time. On the present work it is believed that the way evolution goes is a good source of inspiration to artificial intelligence. According to Dennett, a kind of mind that appeared along evolution is the Popperian mind capable of imagining, planning future states and learning from environment presenting great capacity to adapt to new and unexpected situations. A Popperian agent is modeled and implemented to learn from environment rules and to plan future actions based on self knowledge. Finally, two prototypes of Popperian agents are implemented to solve distinct problems and it can be observed the capacity of the Popperian agents to adapt to environment conditions in order to accomplish own objectives.

Keywords: artificial intelligence, agents, popperian creatures, neural networks.

1 Introdução

A inspiração biológica é freqüentemente usada no estudo da inteligência artificial especialmente nas áreas de algoritmos evolutivos e redes neurais artificiais. No presente trabalho, a evolução da vida e da inteligência é usada como ponto de partida para o desenvolvimento de agentes artificiais.

A vida terrestre pode ser vista como um sistema onde a ordem e a complexidade crescem ao longo dos milhões de anos[DAR87]. A inteligência é algo que emergiu e é selecionada pela evolução natural . Sendo assim, pode-se discutir o que é inteligência sob o aspecto evolutivo. Essa discussão pode inclusive começar com a origem da vida e ir até a consciência humana.

A evolução da inteligência pode ser estudada esquecendo-se os meios biológicos que a tornam possível conforme faz Daniel Dennett. Dennett preocupa-se com a evolução do processo informacional com que as criaturas resolvem seu problemas e não com a evolução do substrato usado nessas soluções. Por outro lado, estudar o impacto evolutivo e comportamental que ocorreu com a evolução dos primeiros seres vivos possuidores de neurônios assim como a função neuronal não é menos interessante. De fato, os enfoques biológico[SAG77] e filosófico[DEN97] sobre a evolução da inteligência são discutidos aqui.

O estudo da evolução da inteligência justifica-se no presente trabalho tendo em vista que ele é usado como base para o desenvolvimento de um tipo específico de inteligência que emergiu durante a evolução: a inteligência popperiana nos moldes que sugere Dennett. Ainda que as criaturas de Dennett sejam discutidas em profundidade na presente dissertação, vale ressaltar que a criatura popperiana é uma criatura que aprende as regras do seu meio ambiente e usando essas regras aprendidas planeja o seu futuro podendo planejar seqüências de ações.

A criatura popperiana tem características marcantes como o fato de que os planos que ela gera para alcançar a sua própria satisfação são difíceis de serem previstos por outra criatura. A experiência vivida por duas criaturas independentes raramente é a mesma. Sendo assim, seu aprendizado raramente é o mesmo implicando em planejamentos feitos sobre aprendizados distintos. O objetivo da presente dissertação é a construção de agentes ou criaturas artificiais com grande capacidade de adaptação na busca de seus objetivos inspirada na evolução da vida e da inteligência. A capacidade da criatura popperiana de aprender e planejar confere grande capacidade adaptativa na busca da satisfação.

Pode-se questionar o porquê do uso da teoria da evolução como fonte de inspiração para a criação de criaturas extremamente adaptáveis ao seu meio. Considerando que a evolução vai do mais simples para o mais complexo, sob o aspecto de engenharia de software, parece fazer sentido começar com modelos de inteligência simples que são facilmente entendidos e implementáveis artificialmente e gradualmente avançar para modelos mais complexos seguindo os mesmos passos de complexificação encontrados na evolução biológica. Para tanto, na presente dissertação, discute-se os seguintes assuntos:

- Genética e evolução da vida.
- Aparecimento e função dos neurônios.
- Evolução da inteligência e das criaturas de Daniel Dennett.

- Experimentos já feitos na área.
- Modelo, implementação e testes do agente popperiano.

1.1 Evolução Biológica

Nos primeiros milhões de anos após a formação da Terra, os relâmpagos e a luz ultravioleta do sol estavam decompondo as moléculas da atmosfera primitiva rica em hidrogênio. Novas moléculas de complexidade gradualmente maior combinavam-se na atmosfera. Os produtos da química atmosférica precipitavam-se no oceano que conseqüentemente se tornava quimicamente mais complexo [SAG 83]. O Jardim do Éden molecular estava pronto. Caoticamente, a primeira molécula capaz de fazer cópias grosseiras de si mesma é formada. Toda uma estrutura de herança e evolução e toda atividade biológica que tivemos contato está baseada em uma molécula ancestral do ácido desoxirribonucléico, o DNA. Molécula principal da vida na Terra, o DNA tem a forma de uma escada em que cada degrau pode conter um de quatro tipos de nucleotídeos diferentes que são as quatro letras do alfabeto genético. Considerando que são necessários dois bits para representar uma de quatro alternativas possíveis, cada letra do alfabeto genético carrega 2 bits de informação.

A vida é uma conseqüência da química orgânica. Na origem do universo e na origem da química, não foram definidas as regras da biologia. As regras da biologia emergem das regras da química. As regras do todo emergem das regras das partes. O comportamento do todo emerge do comportamento das partes. Considerando a química um sistema simples e a biologia um sistema complexo, o sistema complexo emerge do sistema simples. O aparecimento da vida é um exemplo de auto-organização.

No oceano primitivo, não existiam predadores; existiam somente moléculas se duplicando. A evolução ao nível molecular seguia de forma implacável. Com o passar de milhões de anos, moléculas especializadas se agruparam formando as primeiras células primitivas [SAG 83].

A vida na Terra apareceu logo depois de sua formação. A Terra foi formada há 4.6 bilhões de anos atrás enquanto que a vida apareceu há 4 bilhões de anos. Considerando as dimensões de tempo, a vida apareceu pouco depois da formação da terra. Toda a vida na Terra possui código genético descrito na mesma cadeia de DNA com 4 nucleotídeos [SAG 83]. Seres humanos, árvores e bactérias são descritos pelo mesmo alfabeto genético. Darwin não poderia estar mais certo. O motivo pelo qual os organismos são diferentes está no fato de que as instruções em seus códigos genéticos são diferentes ainda que seus alfabetos sejam os mesmos [SAG 77].

É interessante observar que o DNA é um sistema que armazena informação de forma digital. A seleção natural escolheu um sistema de armazenamento de informação digital. Para efeito de cópia ou replicação, a informação digital é regenerada enquanto que a informação analógica é amplificada. Quando a informação analógica é replicada, os ruídos ou erros presentes na informação analógica são igualmente replicados ou até amplificados. De forma contrária, durante a replicação, a informação digital pode ser regenerada e os ruídos podem ser filtrados. Sendo assim, a informação digital é mais resistente contra erros se queremos transportá-la através do espaço ou do tempo.

Os segredos da evolução são mutação, tempo, replicação e morte[SAG77]. A mutação é uma alteração de um nucleotídeo que é passado para a geração seguinte. A maior parte das mutações são desfavoráveis para o indivíduo enquanto que uma

pequeníssima parte torna o indivíduo melhor adaptado melhorando suas chances de propagar seu DNA. A morte é responsável pela eliminação dos indivíduos menos adaptados contribuindo para a evolução. Considerando que o código genético é digital, a mutação também ocorre de forma digital.

Para Manfred Eigen [EIG 97], todo sistema que possua auto-replicação, mutação e metabolismo está sujeito a evolução. Sem a auto-replicação, a informação seria perdida a cada geração. Sem a mutação, a informação seria inalterável e não poderia emergir. Sem o metabolismo, o sistema cairia em equilíbrio significando a morte do ser vivo.

De forma mais abstrata, se queremos implementar um algoritmo de otimização, basta implementar a replicação e mutação de soluções e alguma pressão seletiva que selecione as soluções de maior interesse. A evolução é um processo de otimização. O ambiente em que os seres vivos habitam impõem uma pressão que seleciona os mais adaptados naturalmente. A pressão seletiva implicará na evolução das soluções.

Os segredos da evolução não dependem de um determinado substrato tendo em vista que eles podem ser encontrados na natureza e em algoritmos evolutivos desenvolvidos para computadores. Para que a vida evolua, basta que exista reprodução, mutação e seleção. É importante observar que a evolução não depende de espaço e tempo contínuos tendo em vista que a continuidade não é necessária para a reprodução, morte e mutação.

Moléculas com funções especializadas formaram colônias que resultaram nas primeiras células. Há 3 bilhões de anos, os primeiros seres multicelulares evoluíram a partir de células vegetais. É importante observar que árvores e seres humanos são colônias de células. Cada célula também é uma colônia de seres vivos descendentes dos seres vivos que existiram no mar primitivo da Terra. O fato de que as mitocôndrias tenham seu próprio DNA sugere fortemente que elas sejam descendentes de organismos independentes do mar primitivo da Terra. Somos uma colônia de células em que cada célula também é uma colônia.

Há 1 bilhão de anos atrás, a atmosfera da Terra foi alterada de forma irreversível. Os organismos vegetais primitivos da Terra fabricaram quantidades enormes de oxigênio molecular que foi lançado na atmosfera. As células vegetais possuem cloroplastos que são responsáveis por converter luz solar, água e dióxido de carbono em carboidratos e oxigênio. A atmosfera perdia suas características primitivas como a presença abundante de hidrogênio. O oxigênio, molécula altamente reativa com diversas moléculas orgânicas, foi um gás letal para a maior parte dos seres vivos desprotegidos [SAG 83].

Há 600 milhões de anos, o domínio das algas sobre o planeta foi perdido na revolução cambriana para uma enorme diversidade de novos e mais complexos seres vivos. Até então, a evolução ocorria principalmente no nível de estrutura celular e bioquímica. Pouco depois, apareceu o primeiro peixe e o primeiro vertebrado. Plantas migraram para a terra e a evolução seguiu seu curso [SAG 83].

Bactérias e outras formas de vida simples que baseiam suas regras de comportamento em DNA são agentes biológicos com pouca capacidade de adaptação. O momento da evolução que gerou formas de vida em que o comportamento poderia se modificar durante a vida do agente sem alteração do seu código genético foi um tremendo salto evolutivo.

No início da vida na Terra, a evolução dependia da mutação e da seleção natural. Todo o conhecimento (informação usada para resolver problemas) dos seres primitivos

estava armazenado no seu DNA. A evolução seguia de forma lenta [SAG77]. Uma bactéria usa o conhecimento do seu DNA para construir uma cópia de si mesma. Uma bactéria é uma máquina auto-replicativa.

Com o aparecimento dos cérebros, a memória dos eventos ocorridos durante a existência dos seres poderia ser usada como base de conhecimento para resolver novos problemas. A medida que a evolução transcorre, agentes biológicos cada vez mais adaptáveis ao ambiente e cada vez mais distantes da noção de agente reativo evoluem. Nesse momento, os seres vivos que possuíam cérebro tinham uma ferramenta que incrementava e adaptava o seu conhecimento e comportamento de forma extremamente dinâmica.

Os agentes biológicos que possuem cérebro possuem memória e regras de comportamento extragenéticas (fora do genético). Na próxima seção, é introduzida a unidade funcional do cérebro: o neurônio.

1.2 Neurônios Naturais

Para entender a maneira que um neurônio biológico processa informação, deve-se entender algumas de suas propriedades químicas e elétricas encontradas na bibliografia [DOW98] [GUY91] [KAN97] [NET92]. Sob certas condições, um neurônio sofre um distúrbio no seu potencial elétrico do tipo “tudo ou nada” que se propaga conduzindo informação para outros neurônios. Esse distúrbio no potencial elétrico é chamado de potencial de ação ou simplesmente disparo. Neste capítulo, primeiramente, aborda-se o neurônio no estado de repouso. Posteriormente, estuda-se o disparo do neurônio.

1.2.1 Neurônio em Repouso

A membrana de um neurônio é permeável aos íons $\mathbf{Na}^+$ e $\mathbf{K}^+$ entre outros. Os canais de repouso são responsáveis pela passagem passiva de determinadas substâncias químicas pela membrana. Um canal de repouso de $\mathbf{Na}^+$ é um canal onde somente passa $\mathbf{Na}^+$ sem gasto de energia por parte do neurônio.

A força que promove a passagem de íons através da membrana é a força eletroquímica. A força eletroquímica é resultado das forças químicas e elétricas. A força química é resultado de gradientes de concentração de elementos químicos enquanto que a força elétrica é resultado de gradientes de potenciais elétricos.

No interior do neurônio biológico, a concentração de $\mathbf{K}^+$ é muito maior que fora dele. Sendo assim, em um neurônio em repouso, existe força química que força os íons $\mathbf{K}^+$ a saírem do neurônio pelos canais de repouso de $\mathbf{K}^+$. Por ser o $\mathbf{K}^+$ eletricamente positivo, a medida que os íons de $\mathbf{K}^+$ saem do neurônio, o neurônio fica eletricamente negativo. A medida que o neurônio fica eletricamente negativo, a força elétrica que força o $\mathbf{K}^+$ positivo a entrar no neurônio cresce. O potencial elétrico de equilíbrio para o $\mathbf{K}^+$ é de -75mV.

No exterior do neurônio, a concentração de $\mathbf{Na}^+$ é maior do que no interior dele. Sendo assim, forças químicas e elétricas impulsionam os íons de $\mathbf{Na}^+$ para o interior do neurônio através dos canais de repouso de $\mathbf{Na}^+$. Com a passagem de $\mathbf{Na}^+$ para o interior do neurônio, o neurônio tende a ficar eletricamente positivo. O ponto de equilíbrio de potencial elétrico para o $\mathbf{Na}^+$ é de +55mV. Existem mais canais de repouso de $\mathbf{K}^+$ do que canais de repouso de $\mathbf{Na}^+$. Sendo assim, em repouso e em equilíbrio, a diferença de potencial registrada entre o interior e o exterior da membrana do neurônio é de -60mV.

Em um neurônio em equilíbrio de potencial elétrico, entram e saem a mesma quantidade de $\mathbf{K}^+$ e $\mathbf{Na}^+$ mantendo a quantidade desses íons quase constante. O fluxo passivo de $\mathbf{K}^+$ saindo e $\mathbf{Na}^+$ entrando é compensado pelas bombas de $\mathbf{K}^+$,que bombeiam $\mathbf{K}^+$ para o interior, e pelas bombas de $\mathbf{Na}^+$,que bombeiam $\mathbf{Na}^+$ para o exterior do neurônio.

Se, por algum motivo, o neurônio apresentar um potencial elétrico de -50mV, sairão mais íons de $\mathbf{K}^+$ pelo canal de $\mathbf{K}^+$ do que entrarão pela bomba de $\mathbf{K}^+$ fazendo com que o potencial do neurônio volte para o seu potencial de equilíbrio de -60mV.

1.2.2 Disparo do Neurônio

Na membrana dos neurônios, além dos canais de repouso e das bombas, existem os canais voltagem-dependentes. Como o próprio nome sugere, os canais voltagem-dependentes abrem-se quando submetidos a uma determinada voltagem. Os canais voltagem-dependentes de $\mathbf{Na}^+$ abrem-se a medida que a voltagem torna-se mais positiva ultrapassando o limiar que é de cerca de -35mV.

Quanto maior for o número de canais de $\mathbf{Na}^+$ abertos, maior será o número de íons de $\mathbf{Na}^+$ que ingressarão no interior do neurônio tornando-o mais eletricamente positivo. Quanto mais eletricamente positivo for o neurônio, mais canais de $\mathbf{Na}^+$ se abrirão. Nesse processo, o potencial no interior do neurônio cresce até atingir cerca de +50mV atingindo o potencial máximo.

Quando o potencial necessário para abertura dos canais voltagem-dependentes de $\mathbf{Na}^+$ é atingido na zona de gatilho do neurônio, o gatilho dispara e os canais voltagem-dependentes do axônio começam a se abrir um após o outro conduzindo o sinal elétrico (variação brusca de potencial elétrico ou potencial de ação) por todo axônio até as sinapses com outras células. Após algum tempo, os canais de $\mathbf{Na}^+$ voltam a se fechar permitindo que o neurônio volte ao seu estado de repouso. A zona de gatilho pode ser vista na figura 1.1.

A chegada do potencial de ação nas sinapses químicas provoca de forma indireta a entrada de íons eletricamente positivos (excitatórios) ou a entrada de íons eletricamente negativos (inibitórios) nos neurônios alvo.

Quando um neurônio é alvo de um potencial elétrico que resulta em um estímulo inibitório através de sua sinapse, ele fica mais eletricamente negativo. Ao contrário, se ele recebe um estímulo excitatório, ele fica mais eletricamente positivo. Entre outros fatores, o potencial elétrico é resultado da soma dos estímulos excitatórios e inibitórios que recebe em pontos espacialmente diferentes. Nesse aspecto, os neurônios são somadores espaciais. Um único neurônio pode receber estímulos de milhares de neurônios diferentes.

Quando um neurônio é excitado mas não chega a disparar, ele não volta imediatamente ao seu estado de repouso. Sendo assim, o potencial elétrico instantâneo de um neurônio é resultado dos estímulos que recebeu em seu passado recente na ordem de 10^{-3}s. O neurônio é um somador temporal ao considerar que o potencial do neurônio é resultado de estímulos que ocorrem ao longo do tempo. O disparo do neurônio pode ser causado por estímulos que ocorrem ao longo do tempo. Sendo assim, ***os neurônios são somadores espaciais e temporais*** [GUY91] [KAN97] [NET92].

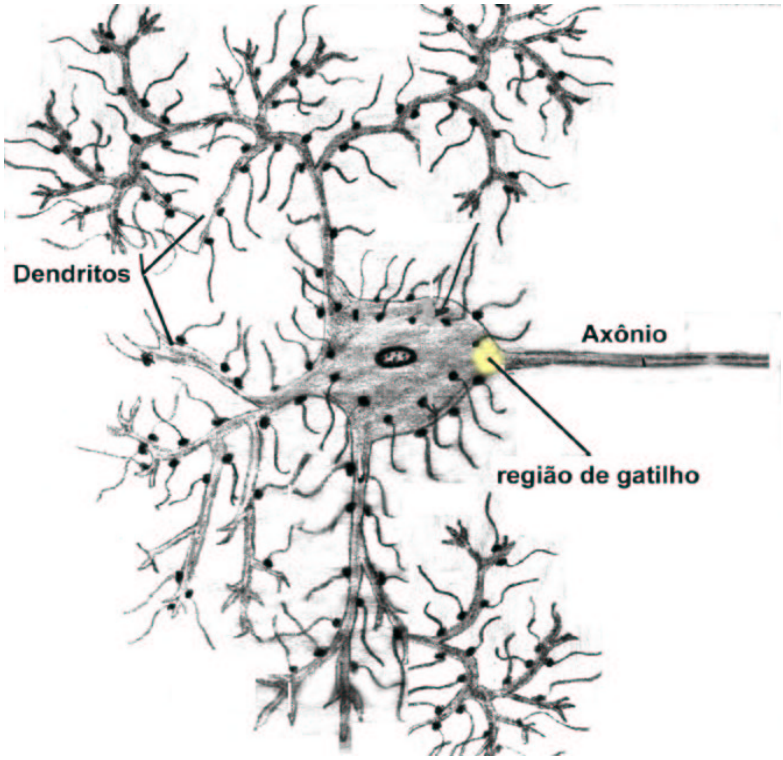

FIGURA 1.1 – Neurônio Natural

Outro aspecto interessante relacionado ao disparo do neurônio é o fato de que o neurônio não dispara com a mesma intensidade (variação de potencial) quando consecutivamente acionado. Isto se deve ao grande número de canais de $\mathbf{Na}^+$ que se mantêm inativos durante alguns milissegundos após o disparo. Sendo assim, dados dois disparos espaçados por uma curta quantidade de tempo, o segundo disparo dispõem de menos canais de $\mathbf{Na}^+$ para se propagar resultando em um disparo de menor intensidade.

1.3 A Necessidade da Evolução e Adaptação do Conhecimento

No início da evolução, os seres vivos possuíam todo o seu comportamento definido no seu DNA, sem capacidade de aprendizado e comportando-se como agentes reativos. Com a evolução dos neurônios, surge uma maneira para as criaturas aprenderem com a experiência.

Mais adiante, quando os seres humanos começam a desenhar e escrever pondo os frutos de sua inteligência fora de seu cérebro, surge o conhecimento extrassomático. A grosso modo, o conhecimento extrassomático é o conhecimento que está armazenado fora do corpo. No momento em que se passa a depositar o conhecimento em máquinas de forma que elas passam a resolver problemas de forma autônoma, surge a inteligência extrassomática que é usualmente chamada de inteligência artificial.

Vale lembrar que conhecimento é a informação usada para resolver problemas. O DNA armazena o conhecimento de uma célula. Segue a lista com os passos evolutivos mais importantes no tratamento do conhecimento:

1. DNA.
2. Cérebro ou meio de armazenamento de conhecimento fora do DNA ou conhecimento extragenético.

3. Meio de armazenamento de conhecimento fora do corpo ou conhecimento extrassomático.
4. Meio de processamento de conhecimento fora do corpo ou inteligência extrassomática.

Observando-se a lista acima, observa-se que a inteligência extrassomática é o passo evolutivo óbvio a ser dado. Sistemas especialistas são sistemas artificiais que simulam as decisões de um especialista humano em determinada área do conhecimento. Indo ainda mais longe na implementação de inteligência extracorpórea, sob o sonho distante de construir uma mente artificial, as redes neurais artificiais são implementadas com inspiração nas redes neurais naturais. Neste ponto, a inteligência extrassomática não apenas é uma forma artificial de processar o conhecimento como também é inspirada no processador natural de conhecimento: o cérebro.

Além da forma artificial de resolver problemas, será que é possível construir criatividade artificial? Usando algoritmos genéticos como forma de resolver problemas, as soluções encontradas não são fruto de atividade consciente ou engenharia e sim resultado da evolução de soluções parciais. Considerando a incerteza que encontra-se nos métodos evolutivos de computação, não é possível prever qual será a solução que a máquina encontrará. Pode-se apenas afirmar que a máquina encontrará uma boa solução. Mesmo em execuções sucessivas, o mesmo algoritmo evolutivo pode responder com soluções boas e ao mesmo tempo significativamente distintas. As leis de Darwin para a evolução explicam as mais diversas formas de vida que conhecemos. De certa forma, as leis de Darwin são criativas e correm no sentido contrário da entropia. As leis da evolução criaram sistemas vivos que ao longo do tempo ganharam em complexidade. Se criatividade é a capacidade de criar coisas novas, então as leis da evolução biológica e da evolução artificial, que são as mesmas, estão repletas de criatividade.

1.4 Inteligência

O significado do termo inteligência evoluiu muito ao longo do tempo absorvendo o significado de outros termos. Morais [MOR 90] entende que o termo inteligência é um dos termos mais equívocos da filosofia e psicologia atual sobretudo devido à diversidade de conteúdos que carrega. Por vezes, o termo inteligência aparece significando entendimento, intelecto, conjunto de funções cognitivas, razão e capacidade de resolver problemas.

Orador e político romano, Marcus Tullius Cicero [LAR 95] (106 a.C - 46 a.C) traduziu do grego para o latim o termo *intelligentia* que significa penetrar dentro das coisas, captar a sua intimidade, discernir ou entender a essência.

Modernamente, não existe consenso sobre uma definição de inteligência. Spearman desabafa: *"Nos sucessivos congressos com o objetivo de definir a inteligência, discutem-se muitas teorias, ouvem-se brilhantes oradores, mas cada um expõe a sua própria opinião... irreconciliável com as demais"* [MOR 90]. Ainda assim, muitos autores dividem a inteligência em dois tipos: (1) capacidade de compreender as relações que existem entre os elementos de uma situação; (2) maneira de aplicar utilmente as experiências adquiridas e retidas na memória.

Na tradução de Cicero, o termo inteligência refere-se a uma atividade passiva em relação ao meio. Aquele que possui a inteligência apenas capta, entende e discerne não

tomando um papel ativo. Modernamente, além da atividade passiva, o ser inteligente exibe um comportamento ativo tentando resolver problemas com base em sua memória ou sendo criativo.

Robert Sternberg [STE 95] propõe a existência de três tipos de inteligência: (1) inteligência analítica que é responsável pela análise e entendimento dos objetos e suas relações; (2) inteligência criativa que é responsável pelas idéias, criações e capacidade de reagir a eventos desconhecidos; (3) inteligência prática que é responsável pela resolução de problemas que são diariamente apresentados. De forma resumida, a inteligência serve para análise e resolução de problemas. Os três tipos de inteligência reunidos devem resultar para aquele que a detém em sucesso na vida real.

Pode ser traçado um paralelo entre a definição de inteligência de Robert Sternberg e Howard Gardner [GAR 94] tendo em vista que Gardner concorda que a inteligência serve para resolver problemas. Para Gardner, a inteligência humana é fruto da combinação de capacidades intelectuais que resolvem problemas específicos. No seu livro Estruturas da Mente, Gardner escreve: "*Uma inteligência é a capacidade de resolver problemas ou de criar produtos que sejam valorizados dentro de um ou mais cenários culturais ... indico que há evidências persuasivas para a existência de diversas competências intelectuais humanas relativamente autônomas abreviadas daqui em diante como inteligências humanas*".

Assim como Sternberg e Gardner, no livro Como a Mente Funciona, Steven Pinker [PIN 97] concorda no aspecto relativo à resolução de problemas da inteligência: "*a mente é um sistema de órgãos de computação, projetados pela seleção natural para resolver os tipos de problemas que nossos ancestrais enfrentavam em sua vida de coletores de alimentos, em especial entender e superar em estratégia os objetos, animais, plantas e outras pessoas*".

Para que serve a inteligência? Por que ela evoluiu com a seleção natural? Com o aparecimento dos cérebros e da inteligência, a memória e análise dos eventos ocorridos durante a existência dos seres poderia ser usada como base de conhecimento para analisar e resolver novos problemas. Baseada na análise e resolução de problemas, essa resposta é muito semelhante ao que propõe Sternberg para inteligência. Na próxima seção, é abordada a visão de Dennett para a evolução da inteligência ao longo da evolução das espécies encontrada no capítulo 4 do seu livro Tipos de Mentes [DEN97].

Nessa mesma linha de pensamento que vê a evolução da inteligência juntamente com a evolução da biologia, Luc Steels [STE 96] acredita que os sistemas inteligentes apresentam quatro propriedades também encontradas nos seres vivos: manutenção própria, adaptatividade, preservação da informação e complexidade incremental. Para Steels, uma rede neural deve apresentar essas quatro propriedades.

1.5 Criaturas de Dennett

Primeiramente, Dennett observa que quanto mais precisa for a informação e quanto maior for a região de onde uma criatura coleta informação, mais provável será o seu sucesso. No início da evolução das espécies, apareceu o olfato que é uma forma de coletar informação do ambiente próximo. Posteriormente, apareceu a audição permitindo a coleta de informação a distâncias maiores.

Da mesma forma, a visão permite a coleta de informação de alta qualidade a

grandes distâncias. A percepção de eventos a grande distância no espaço eventualmente contribui para a predição de eventos a grandes distâncias no tempo. Por exemplo, a visão permite à presa ver o seu predador a grande distância e prever o possível ataque do predador. Na evolução da vida, observa-se a evolução da capacidade perceptiva.

Assim como ocorreu evolução da coleta de informação, ocorreu evolução na maneira pela qual os seres vivos processam essa informação. Para Dennett, os sistemas de coleta de informação são compostos por um grande número de detectores. Cada detector detecta um determinado evento produzindo resposta do tipo SIM ou NÃO para a pergunta “percebo o evento agora?”. Células individuais da retina do olho humano são detectores. Cada detector apresenta a sua intencionalidade mínima: a intencionalidade de detectar um tipo muito específico de evento.

Seguindo o mesmo raciocínio, os neurônios que disparam ou não disparam estão respondendo SIM ou NÃO a determinados estímulos da mesma forma que os detectores abordados anteriormente respondem SIM ou NÃO. Sendo assim, as células que coletam informação do meio exterior da criatura são funcionalmente semelhantes as células destinadas ao processamento de informação encontradas no cérebro. Dennett chama tais células de agentes.

Como os agentes se organizam em sistemas mais complexos capazes de sustentar tipos de intencionalidade cada vez mais sofisticada? Dadas várias opções de organizações geradas por mutações, as melhores serão selecionadas por seleção natural. Dennett chama a geração e seleção de **Torre de Gerar e Testar**.

A Torre de Gerar e Testar é um modelo muito simplificado no qual pode-se classificar várias opções de organização de cérebros para estudar onde o seu poder se origina.

1.5.1 Criatura Darwiniana

No início da evolução, diversos tipos de criaturas foram gerados por mutação. Os tipos melhor adaptados foram selecionados pelo ambiente. Nesta etapa da evolução, as criaturas não tem memória do passado e nenhum tipo de aprendizado. Para essas criaturas, a única possibilidade de alteração do comportamento é mutação genética durante a reprodução. Essas criaturas são chamadas de criaturas Darwinianas e estão inteiramente definidas no momento do nascimento. As criaturas Darwinianas constituem a base da Torre de Gerar e Testar.

Em diversos aspectos, as criaturas Darwinianas assemelham-se aos agentes reativos. Assim como os agentes reativos, as criaturas Darwinianas não possuem memória dos eventos do passado e não constroem representação do ambiente. Sendo assim, as criaturas Darwinianas agem de acordo com a sua percepção instantânea.

A única diferença marcante entre as criaturas Darwinianas e os agentes reativos está no fato de que as criaturas Darwinianas reproduzem-se, modificam-se por mutação e são selecionadas pelo ambiente. É interessante observar que a reprodução é uma diferença que se aplica mais a estrutura orgânica da criatura do que a sua inteligência propriamente dita. A forma como as criaturas Darwinianas e os agentes reativos processam a informação coletada do ambiente é análoga.

1.5.2 Criatura Skinneriana

Durante a evolução, apareceram as criaturas com a propriedade da plasticidade fenótica chamadas de criaturas Skinnerianas. As criaturas Skinnerianas não estão totalmente definidas no momento de seu nascimento e podem se modificar durante a sua vida.

As criaturas Skinnerianas testam ao acaso diversas ações. As ações que resultarem em sucesso serão reforçadas. As criaturas devem avaliar as suas ações para medir o seu sucesso e então reforçar ou não as suas ações. As criaturas que tiverem melhores métodos de avaliação terão mais chances de sobrevivência.

O reforço de algumas ações sob determinadas condições em detrimento de outras ações é um tipo de aprendizado que Dennett chama de aprendizado ABC. O aprendizado ABC é resultado de um processo longo de treinamento e freqüentemente encontrado em animais. No entendimento do autor do presente texto, o aprendizado que Dennett chama de aprendizado ABC é muito semelhante ao aprendizado por reforço encontrado na bibliografia de inteligência artificial [KAE96] [SUT98]. Uma criatura que possui aprendizado por reforço sente recompensa ao tomar uma ação correta ou punição ao tomar uma ação incorreta. Com o tempo, o aprendizado por reforço leva a criatura a tomar as ações que propiciem recompensa.

Uma característica importante do aprendizado ABC é que ele é resultado de tentativa e erro de ações. Muitas tentativas são feitas e as que resultarem em sucesso são reforçadas. O perigo do aprendizado ABC é a realização de uma tentativa que provoque dano sério ou a morte do indivíduo. O aprendizado ABC não possui nenhum sistema de predição de ações não testadas que provoquem a morte do indivíduo. Sendo assim, a criatura Skinneriana pode fazer uma tentativa e acabar morrendo.

A criatura Skinneriana difere do agente reativo por ter memória das ações bem e mal sucedidas. Por outro lado, a criatura Skinneriana está muito longe de ser um agente cognitivo. A criatura Skinneriana não tem representação do ambiente, não negocia, não planeja e não delibera.

A criatura Skinneriana não é um agente reativo nem um agente cognitivo. Ela é um agente reativo exceto pelo fato de ter memória das ações bem e mal sucedidas. A criatura Skinneriana é um pouco mais evoluída que o agente reativo e está longe de ser um agente cognitivo.

Para evitar a morte da criatura em uma tentativa perigosa, um sistema de predição de conseqüência de ações evoluiu. As criaturas que possuem esse sistema de predição são chamadas de criaturas Popperianas e formam a próxima classe de criaturas a serem estudadas no presente texto na Torre de Gerar e Testar.

1.5.3 Criatura Popperiana

As criaturas Popperianas constroem dentro de si um modelo de mundo com base nas suas percepções ao longo de sua existência. Fazendo uso do seu modelo de mundo, a criatura Popperiana pode realizar simulações no modelo de mundo e avaliar as conseqüências antes de realizar as ações do mundo real. Quando uma criatura faz algo que resulta em insucesso ou sucesso no modelo de mundo, a simulação deve gerar algum tipo de insatisfação ou satisfação respectivamente. Essa insatisfação ou satisfação é a avaliação da ação ou conjunto de ações experimentadas no modelo de mundo.

Ações testadas no modelo de mundo competem entre si. As ações vencedoras na

avaliação da criatura tem mais probabilidade de serem usadas no mundo real. As criaturas Popperianas sentem medo[1] de certas situações apresentadas tanto no modelo de mundo quanto no mundo real. O medo faz com que muitas ações perigosas sejam descartadas no modelo de mundo antes de serem testadas no mundo real.

Além de ter memória dos eventos que ocorreram no passado, a criatura popperiana pode projetar as ações que realizará no futuro diferindo-se bastante de um agente reativo por essas duas características.

Existem características que são comuns às criaturas popperianas e aos agentes reativos. Por não possuir e não poder desenvolver linguagem complexa, as criaturas popperianas não possuem qualquer maneira de expor ou representar o seu conhecimento de forma explícita. Entende-se por linguagem complexa uma linguagem que possua alto poder de descrição e ao mesmo tempo sirva como ferramenta de inferência. Português, inglês e as linguagens humanas em geral são consideradas linguagens complexas.

Pode-se discutir se as criaturas popperianas possuem conhecimento ou apenas informação. Considerando que elas não tem uma maneira de explicitar as suas intenções, desejos e medos, pode-se apenas inferir a forma com que as criaturas processam a sua informação estudando o seu comportamento. Com o tempo, uma criatura popperiana pode aprender que o evento A provoca o evento B. Sabendo que o evento B provoca o evento C, a criatura popperiana tem condições de prever o evento C com a ocorrência do evento A podendo fazer uso dessa predição para alcançar a sua satisfação. Sendo assim, ela está atribuindo o significado ao evento A de que A provoca C.

A criatura popperiana tem informação sobre eventos que ocorreram no passado, atribui semântica a estes eventos e usa esta informação para alcançar sua satisfação. Por tais motivos, acredita-se que a criatura popperiana possua conhecimento mesmo sem poder revelá-lo ao exterior usando linguagem complexa.

A criatura popperiana não tem condições para negociar com outras criaturas não podendo resolver conflitos ou definir estratégia de trabalho em grupo. Ela não tem condição de perceber a intencionalidade de outros agentes.

A criatura popperiana desconhece a própria existência. Ela não pode inferir nada do tipo "*eu existo*". A falta da noção do "*eu*" a impede de inferir sobre o que ela faria na posição do outro para tentar descobrir o que o outro faria. De certa forma, a percepção do "*outro*" somente ocorre quando há a percepção do "*eu*". Sem o "*eu*", não se pode perceber o "*outro*".

1.5.4 Criatura Gregoriana

A criatura gregoriana está no topo da Torre de Gerar e Testar apresentando o nível mais alto de inteligência entre as criaturas de Dennett. A criatura gregoriana apresenta diversas soluções às debilidades encontradas nas criaturas popperianas possuindo percepção do "*eu*", capacidade para aprender e desenvolver linguagem complexa, capacidade para usar e construir ferramentas, capacidade para negociar e trabalhar em equipe entre outras características.

Em uma conferência em 1925, Wolfgang Köhler [KÖH26] descreveu as suas experiências com chimpanzés: "*o antropóide estava preso numa jaula gradeada, observando-me. Fora do alcance dos seus braços, cavei um buraco, coloquei algumas*

1 Entende-se por medo o resultado desfavorável de uma função de avaliação frente a uma situação indesejada.

frutas e cobri tudo - buraco e arredores - com areia. O chimpanzé não conseguia alcançar o alimento desejado, porque o buraco havia sido cavado longe de sua jaula ... quarenta e cinco minutos depois, joguei uma vara dentro da jaula no lado oposto ao do buraco, ... o antropóide imediatamente se apossou dela, ... começou a escavar a areia no ponto exato onde estavam as frutas. Conseguiu desenterrá-las e puxá-las para si". O mesmo experimento foi repetido com diversos graus de dificuldade com chimpanzés diferentes e foram encontrados resultados semelhantes. Não é interesse discutir as conclusões de Köler com os experimentos por serem conclusões voltadas a psicologia. Ainda assim, é interessante observar que os chimpanzés usaram ferramentas para resolver o problema o que é uma característica de criatura gregoriana.

Além das características já citadas, uma característica marcante da criatura gregoriana é a cultura. Entende-se por cultura o conhecimento passado para a geração seguinte por meio de aprendizado de forma que os membros mais velhos do grupo passem informação para os membros mais novos. A cultura é certamente um meio de propagação de informação não genético.

A cultura é uma característica obviamente encontrada na espécie humana; porém, a espécie humana não é a única espécie na qual observa-se a propagação de cultura. Chimpanzés aprendem com seus semelhantes sendo esta a razão para explicar o comportamento diverso de grupos de chimpanzés. A cultura é importante para a propagação de hábitos, uso e construção de ferramentas entre outros aspectos. Chimpanzés aprendem com seus semelhantes a construir pequenas varas de madeira usando galhos para perfurar cupinzeiros, retirar os cupins e come-los.

É surpreendente que chimpanzés apresentem qualidades que até pouco tempo seriam classificadas como exclusividade humanas. Alguns grupos de chimpanzés não desenvolveram a técnica de coletar cupins usando ferramentas construídas por eles próprios. Existe um longo caminho até associar que perfurar um cupinzeiro e continuamente inserir uma vara pode resultar em comida. Mais ainda, aprimorar a técnica escolhendo os galhos certos e os quebrando para que fiquem da forma ideal. Chimpanzés conseguem usar e construir ferramentas e ainda passar o seu conhecimento às gerações seguintes.

Um chimpanzé não é uma criatura gregoriana completa. Conforme o próprio Dennett explica, as criaturas constantes na Torre de Gerar e Testar são modelos simplificados de criaturas encontradas ao longo da evolução biológica da inteligência. Um chimpanzé não possui linguagem complexa, o que impede que ele desenvolva diversas habilidades superiores.

Para Dennett, a intencionalidade é baseada em crenças e desejos. Dennett propõe a classificação da intencionalidade pela sua ordem. Por exemplo, o pensamento "*quero comida*" revela intencionalidade primária; o pensamento "*quero que ele queira comida*" revela intencionalidade de segunda ordem; da mesma forma, o pensamento "*acredito que ele quer que eu acredite em p*" revela intencionalidade de terceira ordem. Espera-se encontrar intencionalidade de alta ordem em uma criatura gregoriana.

A intencionalidade de ordem maior do que um é uma importante ferramenta para esconder informação. Por exemplo, "*acredito que existe comida atrás da porta e quero que ele não acredite que exista comida atrás da porta*" é pensamento de uma criatura que deseja esconder a informação sobre a localização da comida envolvendo intencionalidade de segunda ordem.

É esperado que a intencionalidade de alta ordem surja evolutivamente apenas se

ela for útil para a criatura que a detém. Esconder informação é uma tarefa útil somente se a criatura em questão está inserida em um ambiente onde ela pode adquirir informação que os outros não tenham adquirido. Em espécies onde todos os individuos convivem em grupo, todo o evento percebido por um indivíduo é logo percebido por todos os outros indivíduos do grupo. Nesse caso, esconder informação é algo inútil. Por outro lado, em espécies onde seus indivíduos vagueiam sozinhos, um indivíduo pode encontrar comida em um lugar e desejar esconder essa informação fazendo com que os outros não compartilhem a informação.

É improvável que capacidades atribuídas à inteligência emerjam e sejam mantidas ao longo da evolução biológica se elas não forem úteis à criatura que a possuir. Se a percepção do "eu" apareceu com a evolução, deve haver algum benefício dessa percepção para a criatura que a detiver.

Até aqui, na presente dissertação, foram introduzidos assuntos que não pertencem diretamente a ciência de computação e que servirão de base para o desenvolvimento do agente popperiano. Nas próximas seções, são estudados sistemas computacionais desenvolvidos ou em desenvolvimento que possam oferecer subsídios para o que é pretendido. Posteriormente, aborda-se o modelo e implementação do agente popperiano.

1.6 Non-Axiomatic Reasoning System

1.6.1 Introdução

O NARS [WAN93] é um sistema desenvolvido por Pei Wang [WAN2001] de raciocínio inteligente onde inteligência significa *trabalho e adaptação com recursos e informação insuficientes*. É interessante observar que o ambiente que os seres humanos habitam é cheio de incertezas o que resulta em conhecimento freqüentemente incerto. O NARS é um sistema para coletar, armazenar e processar conhecimentos incertos.

Wang não pretende construir uma definição universal de inteligência. Wang apenas desenvolveu um conceito de inteligência a ser seguido no desenvolvimento do NARS.

No NARS, não existem verdades absolutas, evidências universalmente aceitas ou axiomas. O conhecimento no NARS pode ser refutado e adaptado. O NARS opõe-se ao modelo de sistema de raciocínio artificial baseado em conhecimento completo e por isso mesmo não adaptável. A inspiração do NARS vem do fato de que muitas decisões humanas não são baseadas em conhecimento completo.

O NARS pode coletar informação do ambiente que entre em conflito com sua base de conhecimento. Ao contrário de emitir uma mensagem de erro, o NARS deve aproveitar esse conflito para se adaptar.

No NARS, a relação "*ser*" representada por "$\subset$" é entendida como uma relação de herança. A sentença " $S \subset P$ " deve ser entendida como "*S é um tipo de P*" ou "*S apresenta as propriedades de P*".

1.6.2 O Valor Verdade

Um problema importante a ser estudado é a representação da veracidade ou incerteza da sentença " $S \subset P$ ". Sendo S e P conjuntos, existem *n* instâncias de S e *m* instâncias de S que pertencem ao conjunto P, a razão $f = m/n$ proporciona um índice de

veracidade na sentença “ $S \subset P$ ”.

Para efeito de exemplo, sejam os conjuntos:

S = { “*gatinho*”, “*gatinha*”, “*gato pequeno*”, “*cachorro*” }

Gatos = conjunto de todos os gatos

Pode-se afirmar que “ $S \subset$ Gatos ” significa “ *as instâncias de S são gatos*”:

n = número de instâncias de S = 4.

m = número de instâncias de S que pertencem ao conjunto dos gatos = 3 .

f = *m*/*n* = 0,75.

O exemplo anterior é mais didático do que acurado. Uma criatura que não conhece todos os gatos do mundo e nem todos os elementos do conjunto S não pode realizar o cálculo com total grau de confiança. Além de calcular *f*, é necessário calcular o grau de confiança para o valor obtido.

Supondo que o NARS tenha informação de apenas um único elemento de S chamado “gatinho” que pertence ao conjunto Gatos, calcula-se:

n = 1
m = 1
f = *m*/*n* = 1/1

O problema é que uma única observação do ambiente que resulta em um único elemento na amostra é muito pouco para ter uma idéia confiante de “ $S \subset$ Gatos ”. O valor *f* normalmente referido como freqüência ou força deve ser acompanhado de outro valor *c* para denotar o grau de confiança que se tem sobre o valor *f*.

O grau de confiança *c* é a razão entre o número de evidências observadas e o número de evidências a serem observadas: $c = \frac{n}{n+k}$. O valor de *k* é uma constante no sistema e indica o *tamanho do futuro próximo*. Para efeito de exemplo, com um único exemplo na amostra e *k* = 1 encontraremos a confiança $c = \frac{1}{1+1} = 0{,}5$.

Pode-se calcular os valores máximo *z* e mínimo *a* que *f* pode alcançar no futuro próximo. O valor máximo *z* de *f* é alcançado quando todas as observações do futuro próximo forem positivas: $z = \frac{m+k}{n+k}$. Da mesma forma, o valor mínimo é calculado prevendo que todas as observações do futuro próximo serão negativas como segue: $a = \frac{m}{n+k}$.

Intuitivamente, pode-se afirmar que quanto maior for o intervalo [*a*,*z*] que *f* pode ocupar no futuro próximo, menor é o grau de confiança que se tem sobre *f*. De fato, pode-se verificar que: $z - a = \frac{k}{n+k} = 1 - c$. Essa equação afirma que o tamanho do intervalo [*a*,*z*] é o complemento para 1 da confiança *c*.

Sendo assim, existem três tipos de valores verdade equivalentes que podem ser transformados uns nos outros: {*m*,*n*} , < *f*,*c* >, [*a*,*z*].

1.6.3 Crença

Abordando o exemplo " S ⊂ P < 1 , 0.5 > ", observa-se que a freqüência de S em P é 1 com um grau de confiança de 0.5. Apesar da freqüência ser alta, o grau de confiança é 0.5 o que nos impede de crer que "S ⊂ P" com muita intensidade. É importante extrair um número do par <*f,c*> que dê uma idéia sobre o quanto podemos crer em "S ⊂ P".

A crença identificada pela letra *d* é calculada como a média entre *a* e *z* como segue:

$$d = \frac{a+z}{2} = \frac{m+\frac{k}{2}}{n+k}$$

Com *k=1*, o par < 1, 0.5 > pode ser transformado em { 1,1 } ou [0.5 , 1]. Sendo assim, d = (0.5+1)/2 = 0.75 .

1.6.4 Gramática

A unidade básica do conhecimento do NARS é o julgamento. O julgamento é uma sentença e um valor de veracidade. Um julgamento representa a opinião do sistema sobre como um termo pode ser usado como outro. Por exemplo, " S ⊂ P < 1 , 0.5 > " é um julgamento de que S pode ser usado como P com valor verdade < 1 , 0.5 >.

O NARS aceita dois tipos de perguntas vindas do ambiente:

1. Pergunta "*SIM / NÃO*": a pergunta tem a forma de uma sentença e a resposta a ser dada pelo sistema deve ser um valor verdade com a maior confiança possível.
2. Pergunta do tipo " *O que"*: é uma sentença em que um dos termos é substituido por um ponto de interrogação. O sistema deve responder que termo melhor substitui o ponto de interrogação para tornar a sentença verdadeira.

A gramática é constituida pelas seguintes regras de produção:

<julgamento> ::= <sentença> <valor verdade>
<questão> ::= <sentença> | <pergunta>
<pergunta> ::= ? <é um> <predicado> | <sujeito> <é um> ?
<sentença> ::= <sujeito> <é um> <predicado>
<termo> ::= <palavra> | {<termo>}
<palavra> ::= <letra> | <letra><palavra> | <palavra>-<palavra>
<letra> ::= a|b|...|y|z
<é um> ::= ⊂ |∈

1.6.5 Regras de Inferência

No NARS 2.2, existem dois tipos principais de inferência:

- inferência progressiva: parte de um julgamento e conclui um novo julgamento.
- inferência regressiva: parte de uma pergunta para um julgamento ou outra pergunta.

Considerando que a quantidade de conhecimento e recursos de memória são limitados, é comum que ocorram inconsistências. Uma inconsistência é encontrada quando existem dois julgamentos com sentenças idênticas e valores verdade distintos. Quando a inconsistência é encontrada, é feita a *verificação.*

Supondo que os julgamentos seguintes foram feitos com base em conhecimentos distintos:

$S \subset P \{ m_1 , n_1 \}$

$S \subset P \{ m_2 , n_2 \}$

A regra de inferência de verificação conclui $S \subset P \{ m_1 + m_2 , n_1 + n_2 \}$.

Supondo os julgamentos que seguem:

$S \subset M < f_2 , c_2 >$

$M \subset P < f_1 , c_1 >$

A regra de inferência *dedução* conclui $S \subset P < f_1 f_2 , (f_1 + f_2 - f_1 f_2) c_1 c_2 >$. A regra de dedução aproveita a característica da transitividade da herança. A regra da dedução pode ser descrita como: " *se existem características de S em M e se existem características de M em P, podem existir características de S em P*".

Para estudar a regra de inferência *indução,* aborda-se os seguintes julgamentos:

$M \subset P < f_1 , c_1 >$

$M \subset S < f_2 , c_2 >$

Com base nesses julgamentos, a regra de indução conclui:

$$S \subset P < f_1 , \frac{(f_2 c_1 c_2)}{(f_2 c_1 c_2 + k)} >$$

A regra da indução pode ser descrita como: " *se existem características de M em P e se existem características de M em S, podem existir características de S em P*".

A regra de inferência *Abdução* é muito semelhante a regra de indução. Para estudar essa regra, aborda-se os seguintes julgamentos:

$P \subset M < f_1 , c_1 >$

$S \subset M < f_2 , c_2 >$

Com base nesses julgamentos, a regra de abdução conclui:

$$S \subset P < f_2 , \frac{(f_1 c_1 c_2)}{(f_1 c_1 c_2 + k)} >$$

A regra da abdução pode ser descrita como: " *se existem características de P em M e se existem características de S em M, podem existir características de S em P*".

1.6.6 Exemplo

Um exemplo prático permite uma percepção melhor das regras de inferência. No presente exemplo, supõe-se que o NARS seja alimentado com os seguintes julgamentos:

pombas são aves

pombas normais são voadoras
pombas normais não são nadadoras
algumas pombas são brancas
algumas pombas não são brancas
cisnes são aves
cisnes normais são voadores
cisnes normais são nadadores
cisnes normais são brancos
pinguins são aves
pinguins normais não são voadores
pinguins normais são nadadores

Não existindo axiomas no NARS, a julgamentos positivos como "*pombas são aves*" ou " *pomba ⊂ ave* " é atribuído o valor verdade < 1, 0.9 > e não < 1, 1 >. De forma oposta, um julgamento negativo como "*pombas não são nadadoras*" é na verdade um julgamento como " *pomba ⊂ nadadores* " com valor verdade < 0, 0.9 >.

A palavra "*normal*" altera o valor verdade. Enquanto é atribuído o valor verdade < 1, 0.9 > à "*pombas são aves*", o julgamento "*pombas normais são voadoras*" é o julgamento "*pomba ⊂ voadores* " com valor verdade < 0.9, 0.9 >.

Perguntando ao sistema "*Aves são voadoras?*", encontra-se a resposta "< 0.6, 0.548 >". É dito ao sistema "*X é uma ave*". Questionando o sistema se X voa, o sistema responde "*X ∈ voador < 0.6, 0.493 >*" em que a crença vale 0.549 podendo ser entendida como "*eu acho que sim*". É interessante observar que não existe nenhuma regra explicitada ao sistema que afirme "*aves são voadoras*". Essa conclusão é feita com base nos julgamentos que pombas e cisnes voam e pinguins não voam e que pombas, cisnes e pinguins são aves. Esse é um exemplo da regra de raciocínio de abdução.

Continuando o exemplo, alimentando-se o sistema com "*X é um pinguim*" e perguntando-se ao sistema se X voa, encontra-se a resposta "*X ∈ voador < 0.063, 0.815> "*. Essa resposta pode ser entendida como "*não acredito que X voe*".

1.6.7 Comentários Finais

Por trabalhar com conhecimento incerto, o NARS é falível. É comum que o NARS abstraia conhecimento falsos. Esses conhecimentos falsos provocam revisões. O sistema tende a melhorar de desempenho com o tempo. Haverá melhores condições de criticar o NARS nos próximos capítulos depois de estudar outras idéias e outros sistemas.

Mesmo deixando para capítulos posteriores uma discussão mais profunda, pode-se ressaltar algumas características interessantes:

- O NARS aprende com a experiência e pode gerar conhecimento novo com base em conhecimentos prévios usando regras de inferência.
- Recursos limitados implica em memória e tempo de computação limitados. As regras de inferência não podem ser usadas ao máximo. Um número limitado de regras de inferência pode ser usado a cada unidade de tempo. Sendo assim, as inconsistências podem não ser resolvidas imediatamente.

- A modificação de um nodo de conhecimento não implica na alteração dos demais conhecimentos dele dependentes imediatamente tendo em vista que os recursos de computação são limitados.

Algumas idéias do NARS serão aproveitadas nos próximos capítulos na implementação da criatura popperiana; porém, deve-se observar que a criatura popperiana a ser aqui apresentada usará uma forma de representar o conhecimento muito diferente da forma que o NARS usa.

1.7 WebMind Artificial Intelligence Engine

1.7.1 Introdução

Chamado de Webmind AI Engine (WAE) [GOE2001], o motor de IA da empresa WebMind [WEB2001] teve inspiração no trabalho de Marwin Minsky que propõe a mente como uma sociedade de atores ou processos que formam grupos temporários de trabalho. O conceito de mente usado na construção do WAE é o que segue: "*a mente é resultado de uma rede de atores que interagem, trocam mensagens e modificam um ao outro*". Ben Goertzel [BEN2001], que é diretor da WebMind e é a pessoa mais influente no desenvolvimento teórico do WAE, acredita que a mente é um sistema auto-organizável de atores e relações.

Esse conceito pode ser aplicado à rede neural do cérebro humano onde cada neurônio é um ator. Existem características interessantes no projeto de redes neurais artificiais tais como paralelismo e homogeneidade. Por outro lado, redes neurais artificiais não representam o conhecimento de forma tão compacta e de fácil verificação como o uso de regras lógicas representariam. O WAE pretende ter atores que aproveitem características como paralelismo e auto-organização de uma rede neural artificial com a simplicidade de um conjunto de regras lógicas.

O conceito de inteligência usado na construção do WAE proposto por Ben Goertzel é: "*Inteligência é a habilidade de atingir objetivos complexos em ambientes complexos*". Nesse sentido, tanto maior será a inteligência de uma criatura quanto maior for o grupo de objetivos alcançados e quanto mais complexos eles forem. Sendo assim, calcular a inteligência de uma criatura é calcular quantos objetivos ela pode alcançar e sua complexidade.

Outro projetista do WAE, Pei Wang define inteligência como " *habilidade de trabalhar e se adaptar em um ambiente com recursos e informação insuficientes*". As definições de Wang e Goertzel não são conflitantes. Essas definições de inteligência poderiam ser fundidas em " *habilidade de trabalhar e se adaptar para atingir objetivos complexos em ambientes complexos com recursos e informação insuficientes"*.

1.7.2 Tipos de Agentes no WAE

O motor do WAE é um SMA que possui uma mistura de controle central e controle local em que os agentes apresentam as seguintes características:

- os agentes podem transformar-se, criar e destruir outros agentes.
- alguns agentes reconhecem padrões no mundo exterior, reconhecem padrões em outros agentes e agem diretamente no mundo exterior
- assim como em uma rede neural, os agentes passam sua força de ativação para os agentes vizinhos

- considerando as limitações de recursos, agentes menos usados ou menos importantes devem ser eliminados o que provoca esquecimento para a mente que os possui. A perda de agentes provoca perda de informação; porém, essa perda de informação é necessária para gerar e abstrair novos conhecimentos. A unidade de conhecimento é representada por um agente e suas conexões.
- a auto-percepção da mente deve ser feita por agentes que podem perceber padrões de todo o conjunto de agentes que formam a mente.

No WAE, os agentes são divididos em grupos: **agentes nodos, agentes de ligação, agentes de estímulo** e **agentes andarilhos** (do inglês *wanderer*) . Os **agentes nodos** representam textos, séries numéricas, conceitos e algoritmos entre outros. Os **agentes de ligação** representam relações entre os agentes nodos podendo relacionar similaridade, herança, ordem e implicação por exemplo. Os agentes de ligação podem relacionar agentes nodos ou agentes de ligação significando que agentes de ligação podem relacionar relações. Os **agentes de estímulo** estimulam outros agentes apresentando um padrão vagamente semelhante ao direcionamento da atenção. Os **agentes andarilhos** vagueiam pela rede de agentes criando agentes de ligação. Dentro desses quatro grupos de agentes, foram modelados mais de 100 agentes específicos.

Cada agente no sistema tem sua própria autonomia e seu próprio ciclo de vida; porém, o sistema como um todo deve manter-se operante eliminando agentes e permitindo a sobrevivência de outros agentes em uma abordagem evolucionária. Nesse contexto, a mente artificial é algo muito simples: uma rede de agentes em que os agentes continuamente reconhecem padrões, se estudam, se transformam com a tarefa de atingir os objetivos da criatura possuidora dessa mente.

Goertzel expressa seu otimismo em relação ao seu modelo de mente artificial afirmando: “ *eventualmente, eu suspeito que as idéias embarcadas no WAE serão consideradas demasiadamente óbvias ainda que hoje a idéia de construir uma máquina pensante usando idéias como essas permaneça herege*”.

Dentro da mente artificial, os agentes devem representar o conhecimento da mesma forma. Conforme será visto mais a frente no presente texto, a representação do conhecimento é feita usando lógica probabilística. A representação do conhecimento é feita através de agentes nodos e agentes de ligação permitindo a representação de relacionamentos e regras lógicas arbitrariamente complexas.

Um agente nodo representa o conhecimento de algo que existe no mundo exterior ou de algum conceito que existe no mundo interior. Os agentes de ligação representam relacionamentos entre entidades ou conceitos. Os agentes de ligação podem representar relações entre relações. O conhecimento declarativo é representado pelos agentes nodos. Os dois tipos mais importantes de agentes nodos são nodos de herança e semelhança. No WAE, o conhecimento declarativo versa sobre fatos enquanto que o conhecimento procedural versa sobre como proceder.

Inspirada no NARS, a relação no WAE apresenta uma força que varia no intervalo [0,1]. Força zero significa que a relação não existe enquanto que força 1 significa que a relação é sempre válida.

Devem ser feitas algumas notas sobre a tradução de termos encontrados na documentação sobre o WAE. Para adotar uma nomenclatura mais usual, os termos “*node actors*” e “*link actors*” foram traduzidos respectivamente para os termos “*agentes nodos*” e “*agentes de ligação*”. Na própria documentação original, reconhece-se que

atores e agentes são a mesma coisa para efeitos do WAE. Sendo assim, preferiu-se traduzir os termos na forma já citada.

1.7.3 Módulos

O WAE é dividido nos módulos apresentados na figura 1.2. Em cada módulo, residem agentes especializados para resolver uma classe de problemas. Por exemplo, o módulo de análise de séries numéricas possui agentes especializados em reconhecer padrões numéricos. Os agentes dentro do WAE podem se intercomunicar livremente. Agentes do módulo de categorização podem livremente se relacionar com agentes de séries numéricas. A divisão em módulos do WAE não restringe a comunicação entre os agentes. Ela apenas restringe o ponto de moradia dos agentes.

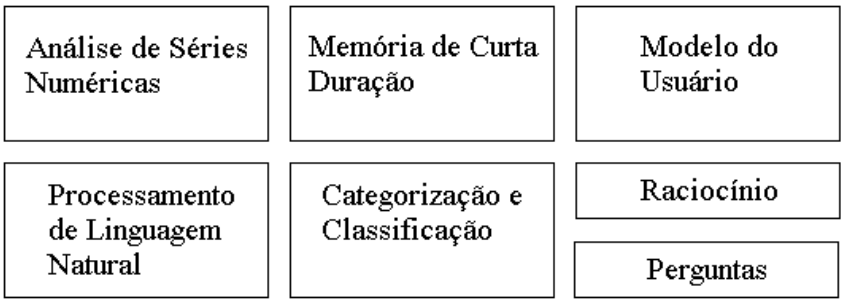

FIGURA 1.2 – Módulos do WAE

O módulo de processamento de linguagem natural apresenta agentes que representam textos, palavras, idéias, conceitos, estruturas lingüísticas e relações entre textos. Um ponto central e difícil de resolver em termos computacionais é a extração de conhecimento de texto e representação em forma de agentes. Esse trabalho é feito por agentes de leitura que realizam processamento sintático e semântico transformando a semântica do texto em agentes.

De acordo com o que o nome sugere, o módulo de categorização possui agentes que representam grupos de agentes. Os agentes que constituem o grupo podem pertencer a qualquer módulo.

No módulo de modelo de usuário, existem agentes que modelam o comportamento do usuário. Cada usuário deve ser representado por um agente. As propriedades e características de cada usuário são representadas também por agentes.

Agentes individuais podem ter alguma inteligência, porém a maior parte da inteligência está na maneira como os agentes criam e usam relações (conexões) com outros agentes podendo existir agentes que reconhecem e constroem relações de semelhança e herança entre outros agentes. Uma relação de herança significa que o agente que recebe a herança recebe propriedades de outro agente.

Esperar que certos agentes sejam gerados ao acaso pela dinâmica da mente pode ser muito custoso em termos de tempo. Se previamente sabe-se que a mente artificial a ser construída deve trabalhar com cálculos numéricos, pode-se implementar manualmente os agentes certos para cálculo. Da mesma forma, podem ser implementados agentes especializados em lógica, linguagem humana, banco de dados, internet, etc.

1.7.4 A Corporificação

Um aspecto importante na implementação e modelagem de inteligência artificial é que a inteligência sempre se apresenta corporificada em alguma criatura. A inteligência precisa de um corpo para existir que é o meio para perceber e agir sobre o ambiente. Sendo assim, para construir uma mente artificial equivalente a mente humana, seria necessário construir um corpo equivalente ao corpo humano. Uma máquina verdadeiramente pensante com uma mente digital manifestará a sua inteligência de forma diversa à forma humana por não ter um corpo humano. Isso não implica que uma máquina não possa ter uma mente verdadeira.

A máquina é o meio pelo qual a inteligência artificial observa e age no mundo. Uma máquina que possui percepção da internet perceberá textos, imagens e outras criaturas biológicas ou artificiais que se comunicam por meio da internet. Provavelmente, essa máquina falharia no teste de Turing por perceber um ambiente extremamente diferente do ambiente que a inteligência humana percebe e ainda assim ser um tipo genuino de inteligência.

Sendo assim, essa inteligência artificial pode ser mais inteligente em aspectos onde a seu ponto de observação é privilegiado e quase certamente apresentará desempenho inferior onde o ponto de observação da inteligência humana é favorecido. Tudo isso dependendo da forma como a inteligência artificial for corporificada.

Assim como outras criaturas têm objetivos de sobrevivência como alimentação e reprodução, o objetivo do WAE é responder a perguntas feitas por humanos da melhor forma possível. Para tanto, o sistema deve estimar quais são os desejos dos usuários humanos. Sendo assim, para aprender a responder da forma correta, o WAE deve receber punição ou recompensa para o resultado de cada pergunta a ele submetida. O esforço necessário para treinar o WAE é certamente enorme.

O WAE foi projetado para poder se comunicar em linguagem escrita humana com seres humanos, navegar na internet, ler arquivos de texto e dados numéricos. Por motivos já citados, o WAE não passa no teste de Turing ainda que deva ser fluente sobre os assuntos do seu próprio mundo. De todo jeito, a inteligência do WAE foi projetada para ser muito mais parecida com a inteligência humana do que a inteligência de um macaco ou jogo de xadrez.

1.7.5 O Esquecimento

A dinâmica de agentes gerando novos agentes que reconhecem determinados padrões acaba por provocar uma explosão combinatória no número de agentes. O esquecimento é muito importante para a mente artificial. O esquecimento ou perda dos agentes menos importantes provoca um evolucionismo ao nível da população de agentes. Os melhores agentes sobrevivem enquanto que os piorem são excluídos (esquecidos). Essa evolução no nível dos agentes faz com que a mente artificial seja um sistema dinâmico. Um aspecto importante é que o WAE usa uma mistura de evolução e inferência.

O desempenho da mente será muito melhor se existir uma estratégia para esquecimento. Apenas para efeito de exemplo, considerando 3 agentes que modelam as seguintes frases: (1) tudo o que voa pode cair, (2) passarinhos voam, (3) passarinhos podem cair. Pode-se concluir que o esquecimento do agente que modela a frase “passarinhos podem cair” não envolverá grande custo considerando que esta informação pode ser recuperada pela inferência.

Muitos problemas que uma criatura possuidora de mente resolve exigem a percepção de grandes quantidades de informação que necessitam ser lembradas em uma curta duração de tempo. Por exemplo, consultar um número na lista telefônica para então ligar, após discado o número, o número de telefone não é mais necessário na mente e pode ser esquecido. Esta é a utilidade da memória de curta duração onde a informação pode ser armazenada por um curto espaço de tempo e depois esquecida.

1.7.6 O raciocínio

O raciocínio lógico é indispensável para a mente estabelecendo relações sobre relações previamente conhecidas. No WAE, o raciocínio lógico é uma parte do sistema e não o centro conforme ocorre freqüentemente em sistemas especialistas. O raciocínio gera conhecimento de acordo com um conjunto de regras de inferência pré-determinadas que são independentes do domínio da aplicação. O raciocínio deriva conhecimento novo do conhecimento já existente usando informação não explicitamente apresentada no ambiente.

Para efeito de exemplo, aborda-se a forma com que o WAE trata a herança. Supondo que B é descendente de A e que C é descendente de B, a dedução lógica é que C é descendente de A.

O fato de que as relações possuem uma força significa que a herança não necessariamente deve ser total tendo força 1. Assim como qualquer outra relação, a herança pode ter qualquer valor no intervalo fechado [0,1]. Quando afirma-se que B é descendente de A, afirma-se que as características de A são encontradas em B. Mudando-se o ponto de vista, se B é descendente de A, então existem características de A que existem em B. Sendo assim, se B é descendente de A, então, com alguma força, A é descendente de B. Esse tipo de raciocinio é chamado de reversibilidade da herança. Quanto mais reversível for a herança, maior é a semelhança.

A notação a ser seguida é a que segue: **A → B** significa que B é descendente de A. Observe alguns tipos de raciocínio presentes no WAE na tabela 1.1.

Uma lógica determinista que parte de premissas absolutamente certas e chega em resultados absolutamente certos não é suficiente para o que é pretendido. A percepção do mundo real é cercada de regras lógicas que tem uma certa probabilidade de serem válidas. Nesse sentido, o WAE usa a **lógica de termos probabilista** (do inglês *Probabilistic Term Logic – PTL*) usando as regras de raciocínio apresentadas na tabela 1.1. A probabilidade de um conhecimento ou relação é definida pela força da relação.

TABELA 1.1 – Tipos de Raciocínio

Tipo de Raciocínio	***Descrição***
dedução lógica	**A → B e B → C implica A → C**
reversão da herança	**A → B implica B → A** em um certo grau
indução	**A → B e A → C implica B → C** em um certo grau
abdução	**A → B e C → B implica A → C** em um certo grau

A expressão “Maria acredita que Deus existe” com grau de confiança de 0.3 é representada por (acredita, Maria, Deus existe) [0.3]. Os relacionamentos de alta ordem são também chamados de relacionamentos de relacionamentos. Seguem alguns exemplos de relacionamentos de alta ordem:

(duvida, João, (acredita, Maria, Deus existe)[0.9]) [0.6] : *João duvida com força 0.6 que Maria acredita que Deus exista com força 0.9.*

(filho, B, A) => (pai, A,B) [0.5] : *Se B é filho de A, então A é pai de B com força 0.5.*

Os recursos computacionais são limitados. Sendo assim, deve existir uma estratégia de controle que determine quando aplicar uma regra de inferência, quando sair em busca de premissas e o que fazer com as conclusões. Essas decisões de quando cada ação deve ser tomada devem ser feitas para requisitar a menor quantidade de recurso computacional possível.

A auto-percepção do WAE está distribuída ao longo de diversos agentes. Um agente representa o sentido de “eu”. O agente “eu” é descendente do agente “computador”. Essa ligação faz com que as propriedades de “computador” sejam atribuídas com alguma força ao agente “eu”. Essa relação de herança entre “computador” e “eu” representa o sentido de “eu sou um computador”. A representação de “eu sou um computador” é feita pelos agentes nodos “eu”, “computador” e pelo agente de ligação de herança.

1.7.7 Representação do Conhecimento

Ainda que seja teoricamente possível representar todo o tipo de conhecimento em uma rede de herança, isso não implica que um sistema implementado contendo somente este tipo de relacionamento seja viável. Ainda que as regras matemáticas pudessem ser implementadas numa rede de herança, por simplicidade, as regras matemáticas são implementadas dentro de agentes específicos.

Para efeito de entendimento, aborda-se o seguinte exemplo:

ave ← animal [1.0,0.8]

Nesse exemplo, deve-se entender que ave é descendente de animal com força 1 e grau de confiança 0.8. O símbolo “← ” representa herança.

Abordando um exemplo mais geral:

B ← A [s , c]

B é descendente de A com força ***s*** (strength) e confiança ***c*** (confidence). Supondo que as propriedade de A e B sejam respectivamente a_1, a_2, ... a_n e b_1, b_2, ... b_n sendo a_x e b_x números reais que representam a força da propriedade *x*, a força da herança entre A e B é calculada como segue:

$$\text{força da herança} = \frac{\sum (a_i \,\text{AND}\, b_i)}{\sum a_i}$$

Pode-se interpretar a fórmula acima como a razão entre a semelhança das

propriedades de A e B pelas propriedades de A. Quanto mais semelhantes forem as propriedades entre A e B que A possui, maior será a descendência de B em relação à A. Observando que a_i e b_i são números reais, a operação lógica AND deve ser definida para o domínio dos números reais:

$$x \text{ and } y = xy$$

$$x \text{ or } y = x + y - xy$$

É interessante observar alguns exemplos numéricos das funções *and* e *or* definidas para os números reais:

TABELA 1.2 – Operações Lógicas

x	***y***	***x or y***	***x and y***
0	0	0	0
0	1	1	0
1	1	1	1
0.5	0.5	0.75	0.25
0.9	0.9	0.99	0.81
0.9	0.1	0.91	0.09
0.8	0.1	0.82	0.08

Da mesma forma, a semelhança entre A e B pode ser calculada como a semelhança de suas propriedades. A semelhança é numericamente calculada pela fórmula que segue:

$$\text{força da semelhança} = \frac{\sum (a_i \text{ AND } b_i)}{\sum (a_i \text{ OR } b_i)}$$

Abordando um exemplo numérico, calculando a semelhança entre A e B tendo cada um as mesmas duas propriedades sendo que A possui forças 0,5 e 0,9 para as suas propriedades e B possui forças 0,5 e 0,1 para as suas propriedades, calcula-se a semelhança entre A e B:

$$\frac{\sum (a_i \text{ and } b_i)}{\sum (a_i \text{ or } b_i)} = \frac{((0{,}5 \text{ and } 0{,}5) + (0{,}9 \text{ and } 0{,}1))}{((0{,}5 \text{ or } 0{,}5) + (0{,}9 \text{ or } 0{,}1))} = \frac{(0{,}25 + 0{,}09)}{(0{,}75 + 0{,}91)} = \frac{0{,}36}{1{,}66} = 0{,}21$$

1.7.8 Processamento de Linguagem Natural

O estudo do processamento de linguagem natural do WAE está muito além dos objetivos do presente trabalho. Ainda assim, é interessante ressaltar algumas idéias encontradas na documentação do WAE. No entendimento dos projetistas do WAE, o aprendizado da linguagem deve ser experimental e natural. O domínio da linguagem deve crescer aos poucos de forma integrada à mente.

O processamento de linguagem natural do WAE segue dois princípios filosóficos:

- *Linguagem é mente:* a linguagem não é uma interface para a mente e sim um aspecto da mente. Aprender linguagem é aprender o pensamento lingüístico.

Quanto mais o pensamento lingüístico for aprendido, melhor pode-se aprender linguagem.

- *Linguagem é interação social:* a linguagem é uma maneira de interagir com outras mentes. Para um sistema entender que a linguagem é um meio de interação com outras mentes, o sistema deve se comunicar com outras mentes.

Apesar do aprendizado da linguagem natural ser incremental, algumas regras podem estar previamente implementadas no sistema tornando-o predisposto ao aprendizado. Atualmente, algumas regras de aprendizado são importadas da gramática de Inglês Xtag [XTA 2002].

1.7.9 Baby WebMind

Mesmo quando completamente implementado, o WAE ainda não estará pronto para o uso. Antes do WAE poder entrar em operação, ele deve passar por uma fase de aprendizado. Está em desenvolvimento um protótipo do WAE para testar sua funcionalidade básica e seu aprendizado em um ambiente restrito.

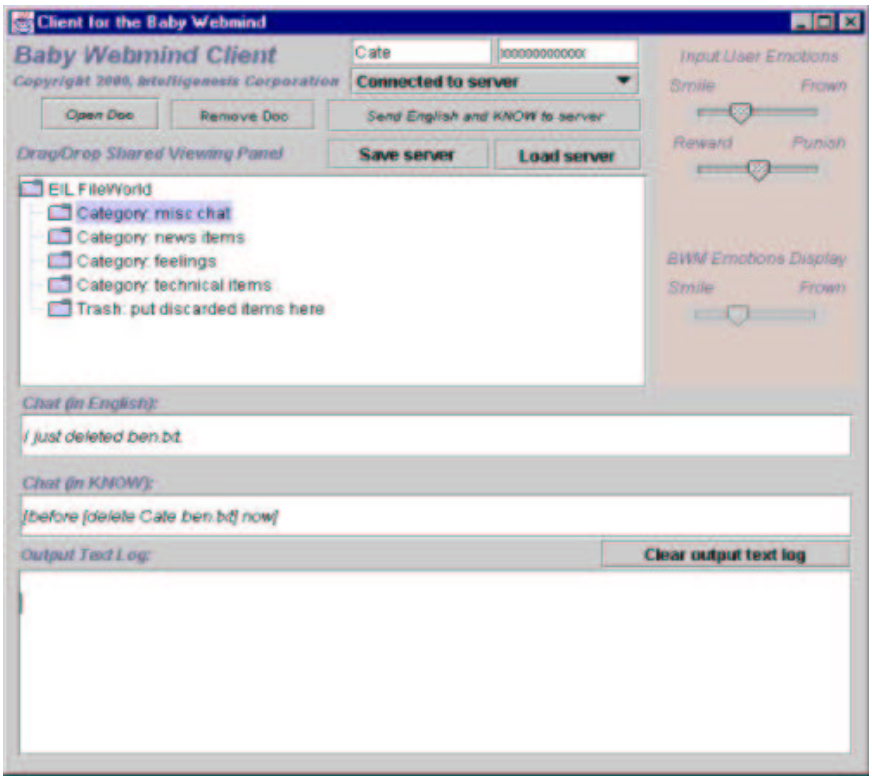

FIGURA 1.3 – Cliente do Baby WebMind em Java

O ambiente do Baby WebMind é um mundo de arquivos onde existem dúzias de operações possíveis tais como apagar, renomear e copiar arquivos. Espera-se que após algum tempo de aprendizagem, o Baby Webmind consiga conversar em Inglês sobre o assuntos relacionados ao seu mundo restrito. Conforme mostra a figura 1.3, a interface com o Baby WebMind apresenta:

- Janela para conversação com o WAE.
- Barra para rolar entre recompensa e punição.
- Barra para apresentar a emoção do usuário entre sorriso e “fazer cara

feia”.

- Barra para o WAE mostrar suas emoções.

1.7.10 Comentários Finais

Por certo, essa apresentação sobre o WAE é extremamente superficial. A apresentação de um projeto desta magnitude em poucas páginas é certamente superficial. Ainda assim, pode-se observar aspectos importantes no desenvolvimento do WAE tais como:

- Adquire conhecimento da interação com o seu ambiente, com usuários e consigo mesmo. Devem existir agentes que percebem relações e padrões no ambiente assim como relações no conhecimento ou comportamento de outros agentes.
- Inspirado no NARS, o conhecimento é não determinista apresentando um valor verdade.
- Raciocínio também inspirado no NARS para trabalhar com conhecimento não determinista.
- A mente artificial do WAE é uma sociedade de agentes que nascem, interagem entre si e morrem sendo este um sistema dinâmico. No cérebro, neurônios ativam ou suprimem a ativação de outros neurônios. Da mesma forma, no WAE, agentes ativam ou suprimem a ativação de outros agentes. O projeto do WAE pretende que seus agentes estejam no meio caminho entre uma rede de neurônios e um conjunto de regras lógicas.
- A inteligência do WAE emerge da interação de mais de uma centena de tipos de agentes diferentes.
- O aspecto da corporificação em máquinas da inteligência do WAE implica em um tipo de inteligência não humana. Isso não chega a ser um problema. É apenas uma característica da mente artificial.
- O esquecimento é uma forma de liberar recursos computacionais. A perda de agentes é necessária para o sucesso da mente.
- O aspecto mais importante do WAE não é o modelo ou a construção dos módulos em separado, mas sim a maneira como os agentes interagem para gerar inteligência.

Os desafios impostos à implementação do WAE são enormes. Existem diversos problemas teóricos e práticos envolvidos na construção de um sistema tão extenso. Cada módulo do WAE é motivo de estudo aprofundado. Os módulos envolvidos na coleta de dados, raciocínio e processamento de linguagem natural poderiam gerar estudos enormes em separado. Além de todos estes desafios, a interação entre uma miríade de agentes especializados nos diversos módulos gerando um comportamento global é sem dúvida um sistema dinâmico de dimensionalidade realmente alta.

O WAE é classificado como uma criatura gregoriana tendo em vista que ele dispõe de linguagem complexa, cultura e capacidade de construir ferramentas. No

entendimento do autor do presente texto, diversos problemas encontrados no desenvolvimento do WAE são decorrência da falta da perspectiva evolutiva da inteligência. Parece fazer mais sentido construir primeiramente um WAE popperiano e posteriormente um WAE gregoriano.

A tecnologia desenvolvida para a criatura popperiana é totalmente aproveitada para o desenvolvimento da criatura gregoriana. Diversas questões abordadas no WAE são comuns para as criaturas popperianas e gregorianas tais como: a corporificação, o aprendizado pela observação, o uso de conhecimento incerto, a predição, o planejamento e o esquecimento.

Além disso, por ser a criatura popperiana mais simples do que a criatura gregoriana, diversos erros de modelagem e implementação podem ser percebidos de forma muito mais fácil no desenvolvimento da criatura popperiana do que no desenvolvimento da criatura gregoriana sendo que a criatura popperiana pronta serve de ponto de partida para a construção da criatura gregoriana.

1.8 Proposta de um Modelo de Esquema Cognitivo Sensório-Motor Inspirado na Teoria de Jean Piaget

Inspirado no trabalho de Waslawick [WAS93], Muñoz [MUÑ99] propõe um modelo computacional de agente cognitivo inspirado na teoria de Jean Piaget [PIA47] sobre o desenvolvimento da inteligência sensória-motora. Muñoz acredita que a teoria piagetiana de esquemas permite a construção de agentes que tenham capacidade de adaptação e que não necessitam de conhecimento prévio a respeito do meio. Essas características permitem ao agente agir em um ambiente desconhecido ou imprevisto.

Nesse agente, tanto a memória, como as regras de comportamento e a previsão estão armazenadas em estruturas chamadas de esquemas. Os esquemas são uma trinca contendo um contexto, ações e o resultado esperado da aplicação dessas ações no contexto apropriado. Um ponto alto a ser considerado no trabalho de Muñoz é que o esquema pode ser usado para diferentes fins (memória, comportamento, previsão) proporcionando uma estrutura de dados bastante homogênea.

Quando o esquema é usado como uma regra de comportamento do tipo "*Se a situação pertence ao contexto, então aplica a ação com a esperança de encontrar o resultado*", é dito que ele é usado de forma procedural. Por outro lado, quando o esquema é usado como uma regra de previsão do tipo " *Se a situação pertence ao contexto e a ação for aplicada, então acredita-se que o resultado será encontrado*", é dito que o esquema é usado de forma preditiva.

Os esquemas podem ser encontrados em um dos seguintes estados:

- Não excitado: verificado quando a situação não pertence ao contexto do esquema.
- Excitado: verificado quando a situação pertence ao contexto do esquema e suas ações não são executadas.
- Ativado: verificado quando a situação pertence ao contexto do esquema e suas ações são executadas.

Os esquemas são construídos através da experiência e podem ser modificados por duas formas distintas:

- Por assimilação que significa a expansão do contexto ou ainda a generalização das situações para as quais a ação pode ser tomada.
- Por acomodação que significa encontrar ações que levem ao resultado.

Essa estrutura em esquemas permite ao agente descobrir novos meios para alcançar o resultado por experimentação mental. Esses novos meios são descobertos pela assimilação ou generalização. Sendo assim, o agente de Muñoz pode ser classificado como uma criatura popperiana tendo em vista que ele tem capacidade de aprender as regras do seu ambiente e imaginar situações futuras.

O estado de um esquema pode ser usado para definir o contexto de outro esquema. Sendo assim, a ativação de um esquema pode depender da ativação de outro esquema de tal forma que ações mentais podem ser modeladas de forma homogênea tendo em vista que tais ações são usadas somente para sinalizar outros esquemas.

O agente de Muñoz possui uma população de esquemas. Esses esquemas são avaliados segundo a sua utilidade e exatidão. Os melhores esquemas são preservados enquanto que os piores são descartados forçando um processo evolutivo ao nível de esquema.

Existem características semelhantes entre o agente de Muñoz e o WAE. A parte central do WAE e do agente de Muñoz é constituída de unidades funcionais que são o agente e o esquema respectivamente. Enquanto que a inteligência no WAE é fruto da coletividade de agentes, a inteligência no trabalho de Muñoz é fruto da coletividade de esquemas. Em ambos os casos, as unidades funcionais de cada sistema organizam-se em redes. A condição de ativação dos esquemas pode ser uma situação percebida no exterior assim como o estado de outro esquema. Da mesma forma, no WAE, os agentes agem baseados em situações percebidas no exterior ou em outros agentes que compõem o WAE. Em ambos os sistemas, as unidades funcionais competem por recursos e pode-se observar um processo evolutivo ao nível das unidades funcionais que resulta na melhora do comportamento global do sistema. Da mesma forma, o descarte de esquemas e o descarte de agentes significa o esquecimento para ambos os sistemas. Ambos os sistemas podem tirar proveito do paralelismo, aprendem com a experiência, predizem o futuro, imaginam situações não experimentadas e possuem ações mentais.

No final de sua dissertação de mestrado, Muñoz propõem duas pesquisas futuras entre outras:

- Utilização de esquemas em simulações mentais com o objetivo de testar mentalmente a aplicação de esquemas a uma dada situação. Essa utilização pode ser considerada como planejamento propriamente dito.
- Inclusão da noção de objeto através da combinação de esquemas.

A noção de objeto que mais tarde resulta na noção de ferramenta enquadra-se na criatura gregoriana e não é investigada no presente trabalho. Por outro lado, o planejamento explícito é uma característica da criatura popperiana e é objetivo da presente dissertação. Conforme pode-se verificar no próximo capítulo, o planejamento explícito proposto para a criatura popperiana é formado por unidades que se parecem bastante com esquemas.

2 Implementação da *Inteligência* das Criaturas

Na presente dissertação, pretende-se fazer um modelo genérico da *inteligência* das criaturas skinneriana e popperiana. Não é pretendido modelar o corpo, a forma de percepção ou as ações das criaturas artificiais. É pretendido apenas modelar a sua *inteligência*. Nessa forma, para efeito de implementação dessas criaturas, entende-se por *inteligência*[2] como a *"forma pela qual as decisões são tomadas com base nas percepções e no estado interno"*.

Sendo assim, pretende-se modelar apenas a forma pela qual as decisões são tomadas sem restrição ao ambiente, ao tipo de eventos percebidos ou ao corpo. Certamente, a forma como a *inteligência* se desenvolve ao longo da vida da criatura depende da forma como a criatura percebe seu mundo e seu corpo e age sobre eles. Aspectos como a forma de percepção e os tipos de ações sobre o ambiente são aspectos envolvidos na corporificação da *inteligência* discutidos na seção 1.7.4 "*A Corporificação*" do presente texto.

Para efeito de generalidade da implementação, é necessário definir a forma como a *inteligência* vai perceber o corpo e o ambiente e como a *inteligência* vai tomar decisões que possam resultar em ações no mundo exterior da forma mais genérica possível sem criar dependências entre a *inteligência* e um determinado tipo de percepção, ação ou corpo. De certa forma, essa definição apresenta a forma como a *inteligência* comunica-se com o mundo exterior.

É importante que a forma como a *inteligência* é implementada seja suficientemente genérica para qualquer tipo de corporificação. Considerando que todo tipo de informação pode ser representado em bits ainda que com ruído, o meio escolhido para fazer a comunicação entre a *inteligência* e o corpo do agente é um vetor de bytes, que também pode ser visto como um vetor de bits.

2.1 Modelo de Agente Skinneriano

Ainda que a definição do agente Skinneriano seja simples, a sua implementação não o é. O agente Skinneriano deve apresentar quatro características: (1) capacidade de perceber o ambiente e os estados internos, (2) capacidade de agir no ambiente, (3) capacidade de aprender a relacionar ações e estados do ambiente com satisfação e (4) saber escolher boas ações baseadas nas regras aprendidas. Dessas quatro características, as duas primeiras descendem da definição de agente enquanto que as outras duas são específicas do agente Skinneriano.

A figura 2.1 apresenta um modelo de agente Skinneriano. Esse diagrama apresenta os principais módulos, seus respectivos números e a informação que trafega entre eles. Esses números apresentados em cada um dos principais módulos representam a ordem de execução das tarefas do agente Skinneriano em uma máquina de processamento serial.

É provável que o resultado das ações do agente produza alguma modificação no que é percebido pelo agente tanto no ambiente como no estado interno. Essa dependência é representada pela seta pontilhada da figura 2.1. As demais setas representam fluxo de informação.

2 Grafa-se o termo inteligência em itálico quando ele apresenta a mesma semântica dada por Dennett.

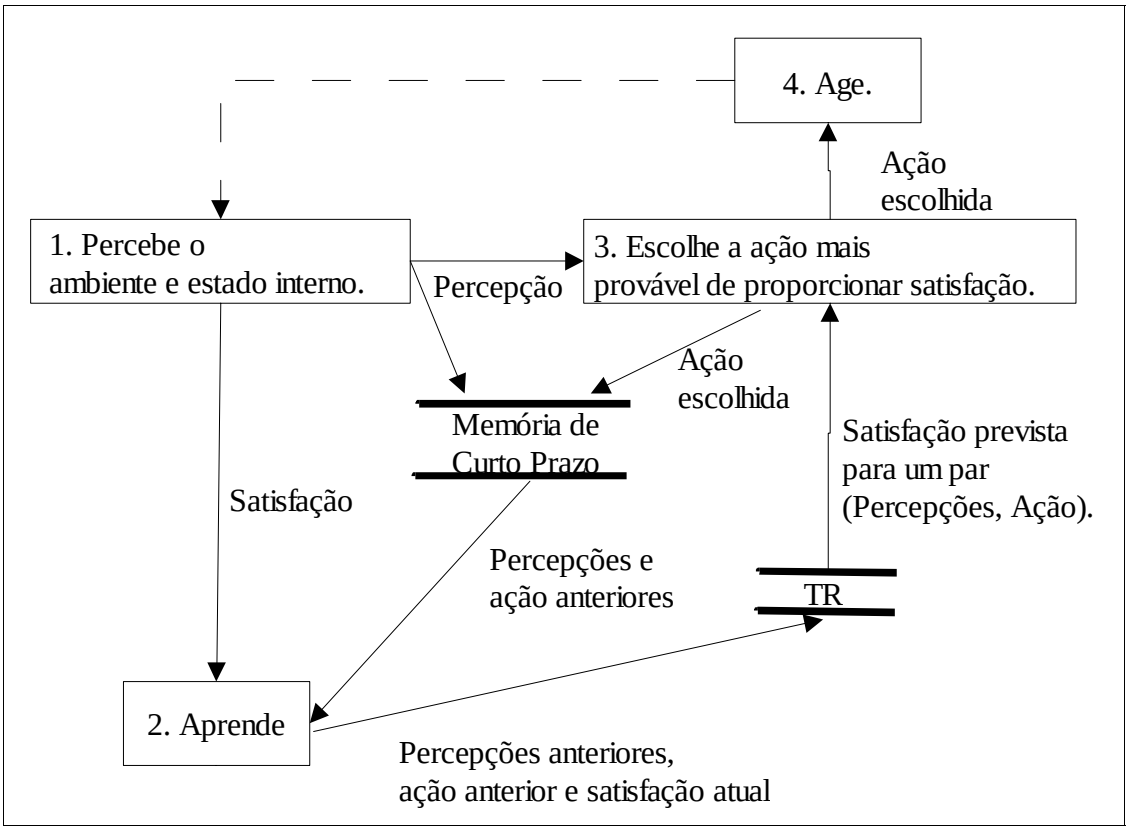

FIGURA 2.1 – Modelo de Agente Skinneriano

A percepção alimenta a memória de curto prazo e o módulo de aprendizado. Durante o aprendizado, o agente Skinneriano relaciona a situação do ambiente, a ação tomada e a satisfação obtida com as ações anteriores.

O estado do ambiente e a ação tomada são informações do passado recente a serem relacionadas (aprendidas) com a satisfação atual. Sendo assim, é necessária a existência da memória de curto prazo para relacionar causas (percepções e ações anteriores) e efeitos (satisfação) temporalmente próximos.

Na figura 2.1, TR representa a **tabela de relação** que é usada para relacionar causas e efeitos. Essa tabela será discutida em detalhes no presente capítulo. Para escolher uma ação, o agente Skinneriano usa a TR com o objetivo de escolher a ação que mais se relacione com satisfação no estado atual percebido.

Usando o relacionamento entre ambiente, ação e satisfação aprendido, o agente Skinneriano pode escolher as ações com maior probabilidade de proporcionar satisfação no futuro próximo.

2.2 Modelo de Agente Popperiano

A grande diferença entre a criatura Popperiana e a criatura Skinneriana é a capacidade que a criatura Popperiana tem de antever estados do ambiente e poder testar as ações nesse ambiente antevisto. Determinar as ações que proporcionem estados futuros desejáveis é planejar.

Além de aprender as regras do ambiente e prever estados futuros próprios e do ambiente, a criatura popperiana deve usar esse aprendizado para planejar seqüências de ações boas e descartar as ações que levem a estados indesejáveis. O sistema de

planejamento é discutido na seção 2.5 enquanto que o aprendizado é discutido na seção 2.3 da presente dissertação.

O modelo de agente popperiano proposto aqui é uma evolução do modelo de agente skinneriano apresentado. Comparando-se as figuras 2.1 e 2.2, as semelhanças entre os modelos ficam evidentes. Além das semelhanças, duas diferenças encontradas no agente Popperiano devem ser observadas: (1) o módulo de aprendizado relaciona percepções passadas com a percepção atual e (2) a escolha da melhor ação envolve planejamento e previsão de estados a serem percebidos no futuro.

Para que a criatura popperiana possa planejar, ela deve ter uma maneira de prevêr o estado a ser alcançado por uma ação tomada em uma determinada situação. Esse estado alcançado pode ser o ponto de partida para outra ação que a criatura planeja.

Além de prever estados futuros, a criatura popperiana deve ter uma maneira de avaliar quais estados são mais ou menos desejáveis ou perigosos. É o instinto que arbitra se um determinado estado é favorável ou perigoso. Para efeito de implementação do agente popperiano artificial, o instinto pode ser implementado como uma função de avaliação em código nativo da linguagem de programação.

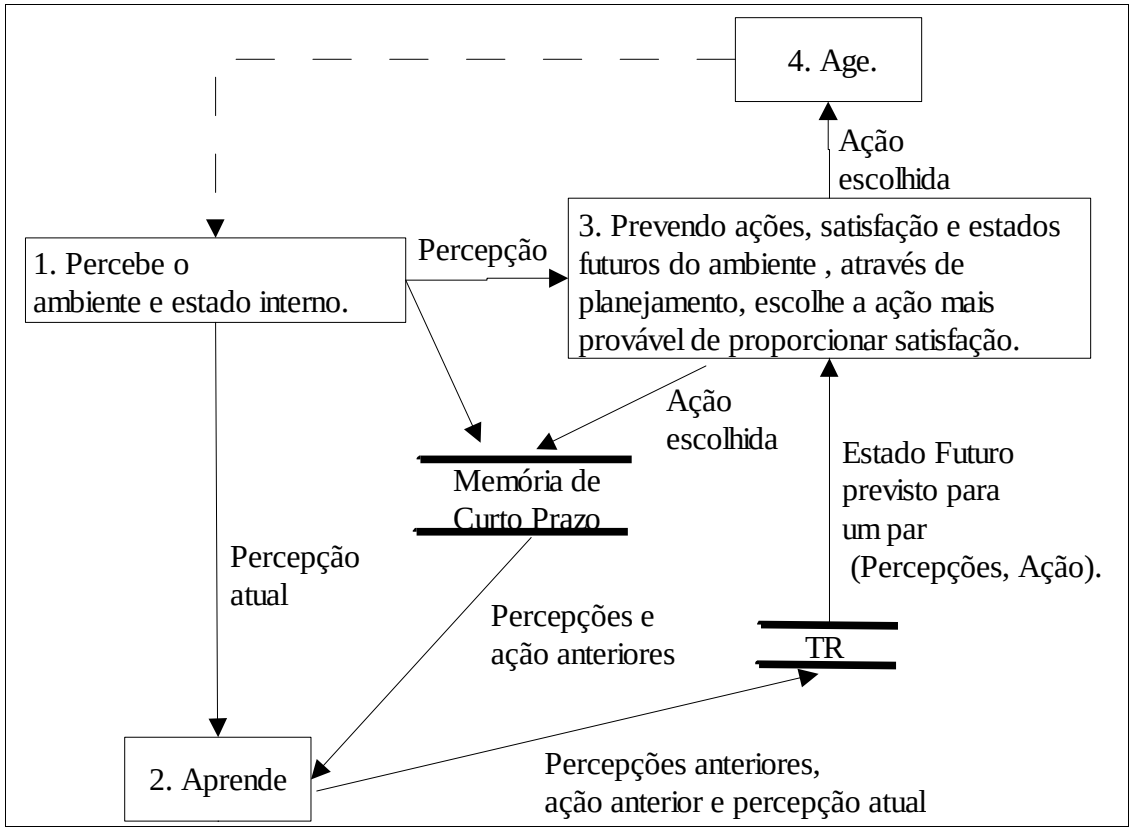

FIGURA 2.2 – Modelo de Agente Popperiano

Os agentes Skinnerianos e Popperianos devem ser capazes de relacionar causas e conseqüências sendo elas ações e satisfação ou estado atual e estado futuro. Esse relacionamento não é simples de implementar. Para relacionar causa e conseqüência foi definido e implementado um algoritmo inspirado no aprendizado de Hebb a ser descrito na próxima seção.

2.3 Implementação da Indução de Função Lógica não Determinista

O aprendizado de Hebb é encontrado tanto na bibliografia de redes neurais artificiais [SIM2001] como na bibliografia de neurologia médica [KAN97]. A maneira como Hebb define o aprendizado a nível celular é bastante genérica e permite modelagens distintas:

"Quando um axônio da célula A está perto o suficiente para excitar a célula B e repetidamente ou persistentemente toma parte no disparo dela, algum processo de crescimento ou mudança metabólica acontece em uma ou ambas as células de forma que a eficiência de A no disparo de B é incrementada" [SIM2001].

Uma implementação possível para o aprendizado de Hebb usada no presente trabalho é definida como segue: em dois neurônios A e B conectados por uma sinapse S que leva o sinal de A para B, a sinapse S é reforçada sempre que A participe excitatoriamente no disparo de B e suprimida quando A dispara e B não. Na seção 2.3.2, o modelo de aprendizado escolhido é discutido e comparado com outros trabalhos.

Além do relacionamento de implicação entre A e B, é interessante procurar por relacionamentos entre não A e B, A e não B e não A e não B. Esse exemplo simples pode ser modelado pela rede neural encontrada na figura 2.3. Na figura 2.3, foram representadas apenas as sinapses de interesse para esse exemplo. Existem sinapses que excitam A e B provenientes de fora do circuito apresentado não representadas na figura 2.3. Nesse modelo, os neurônios A e não A disparam sempre que respectivamente os eventos A e não A são percebidos. O mesmo pode ser afirmado para os neurônios B e não B.

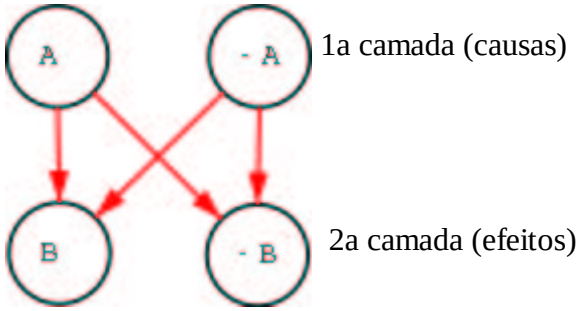

FIGURA 2.3 – Rede Neural de Causas e Efeitos

Todas as sinapses da figura 2.3 estão sujeitas ao aprendizado de Hebb. Sendo assim, sempre que dois neurônios não pertencentes a mesma linha dispararem, ocorrerá aprendizado de Hebb. A medida que as sinapses são reforçadas pelo aprendizado de Hebb, a ocorrência de A ou não A provocará a ocorrência de B ou não B dependendo do aprendizado.

A ocorrência ou não ocorrência de um evento C pode estar associada a ocorrência conjunta de outros dois eventos. Sendo assim, devemos modelar uma rede neural mais poderosa do que a rede neural apresentada na figura 2.3.

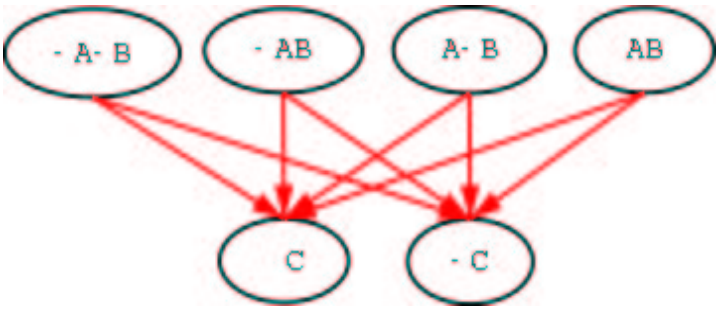

FIGURA 2.4 – Rede Neural de Causas e Efeitos

Para determinar a relação entre apenas dois eventos que ocorrem concomitantemente A e B e a conseqüência C, foram despendidos 4 neurônios na primeira camada. Para determinar a relação entre três eventos que ocorrem em conjunto e determinam a ocorrência de um quarto evento, serão necessários 8 neurônios na primeira camada. Sendo **n** o número de eventos que ocorrem em conjunto, para determinar a relação entre os **n** eventos e a consequência C são necessários 2^n neurônios na primeira camada.

Sendo assim, supondo um agente que possua capacidade para perceber 40 eventos distintos que podem ocorrer em conjunto ou em separado, para determinar se esses 40 eventos tem alguma implicação sobre a ocorrência de um evento C, serão necessários 2^{40} neurônios na primeira camada.

Por certo que é praticamente inviável construir uma rede neural com 2^{40} neurônios na primeira camada. Sendo assim, deve-se usar uma estratégia para tentar relacionar causas e efeitos mais prováveis ao contrário de testar todas as possibilidades. Outra restrição que pode ser feita é limitar o número de eventos que conjuntamente provocam um efeito. Esse número é 5 nos testes apresentados aqui permitindo que se possa relacionar de 1 a 5 eventos do tipo ***causa*** a um único evento do tipo ***efeito*** podendo-se induzir funções do tipo " *¬A e B => X*" ou "*A e B e C e D e E => X*". O problema dessa limitação é que funções do tipo *"A e B e C e D e E e F => X" não* podem ser induzidas por terem muitos termos na causa.

O problema de relacionar estados do ambiente, do corpo e ações com satisfação ou insatisfação é um problema de **indução de função lógica**. Entende-se aqui por indução a "*determinação da relação causal*" ou ainda a "*determinação da implicação*".

A criatura skinneriana pode induzir regras lógicas incertas que ocorrem em um ambiente incerto. Por exemplo, supondo que a regra de comportamento "*se calor, então sair do sol"* propicie satisfação em 70% dos casos em que é usada quando existe "*calor*", a criatura deveria induzir a regra "*percebe o evento calor e a própria ação sair do sol implica no estado satisfação em 70% dos casos"* ou simplesmente *"calor e sair do sol => satisfação [0.7] "*.

A dificuldade da implementação do comportamento do agente skinneriano quase resume-se a implementação de indução de função lógica. Nesse agente, a indução de função lógica é usada para estabelecer relações entre causas e efeitos sendo que a criatura deve tentar provocar as causas (ações) que geram o efeito desejado (satisfação).

Inicialmente, durante a modelagem do sistema de indução lógica, usou-se inspiração no aprendizado de Hebb e pretendia-se implementar um algoritmo baseado em

redes neurais artificiais. A medida que o projeto evoluiu, os aspectos de interesse da rede neural eram abstraídos para modelos cada vez mais simples e mais distantes de uma rede neural artificial. Na versão final, ainda que se possa observar que existe aprendizado por reforço, é imperceptível que em sua fase inicial usou-se redes neurais artificiais como inspiração. O presente texto relata desde a inspiração biológica, os primeiros testes até o modelo completo.

No decorrer do presente capítulo, ficarão evidentes algumas semelhanças entre o NARS e a forma como a indução de função lógica é proposta aqui. Para facilitar o entendimento e a comparação, a notação usada no presente capítulo é a mais próxima possível da notação usada no estudo do NARS e do WAE.

2.3.1 Exemplo Introdutório Simples

Nesse exemplo, aborda-se a figura 2.4. Na tabela 2.2, cada célula das colunas C e ¬C representa uma sinapse que liga um neurônio da primeira camada com um neurônio da segunda camada após a alimentação das percepções da tabela TX1 sendo que:

1) os pesos das sinapses da rede começam todos em zero.
2) os pesos das sinapses são incrementados sempre que os dois neurônios que a sinapse liga dispararem em tempos próximos
3) os eventos percebidos do ambiente são os eventos da tabela 1.1.

TABELA 2.1 – Percepção do Ambiente

Causas (primeira camada) AB	***Efeito (segunda camada) C***
FF	F
FV	V
VF	V
VV	F
FF	V
FV	V
VF	V
VV	V

TABELA 2.2 – Pesos da Rede após a percepção dos eventos da tabela 1.1

(causas)	*(efeitos)*		*f*	*d*
primeira camada	***¬ C***	***C***		
¬A ¬B	1	1	0.5	0.67
¬A B	0	2	1	0.75
A ¬B	0	2	1	0.75
A B	1	1	0.5	0.67

Para efeito de compatibilidade a nível de notação com o que foi estudado na seção 1.11 sobre o NARS, adota-se:

m: número de vezes que uma determinada causa provocou um determinado efeito. Na tabela 2.2, é o número de vezes que a causa provoca o efeito C.

n: número de vezes que uma determinada causa é observada.

f : freqüência ou força = ***m/n***.

d**:** crença em que $d=\frac{m+\frac{k}{2}}{n+k}$.

K: constante inteira e positiva que usualmente vale 2.

Pode-se observar que as colunas de efeitos da tabela 2.2 apresentam freqüências. Por suposição, após algum tempo, atualizando a tabela 2.2 com percepções de causas e efeitos hipotéticos, encontra-se a tabela 2.3. A primeira linha da tabela 2.3 indica que nas 200 vezes que se observou ¬A e ¬B, em 100 vezes observou-se ¬C e nas outras 100 vezes observou-se C. Da mesma forma, a segunda linha da tabela 2.3 indica que nas 200 vezes que se observou ¬A e B, observou-se C.

TABELA 2.3 – Pesos da Rede após a percepção dos eventos hipotéticos

(causas)	*(efeitos)*		*f*	*d*
primeira camada	***¬ C***	***C***		
¬A ¬B	100	100	0.5	0.5
¬A B	0	200	1	0.99
A ¬B	0	200	1	0.99
A B	100	100	0.5	0.5

Analisando a tabela 2.3, pode-se induzir com grande crença que ¬A e B ou A e ¬B implicam em C. De forma contrária, observa-se que ¬A e ¬B ou A e B não

determinam isoladamente o valor de C.

O valor de ***m*** para cada causa pode ser encontrado na coluna ***C***. Os valores de ***f*** na tabela 2.2 são os mesmos valores da tabela 2.3; porém, com valores de crença ***d*** distintos.

2.3.2 A Relação entre a freqüência *f*, a crença *d* e o Aprendizado de Hebb

Encontra-se na bibliografia diversas modelagens do aprendizado de Hebb. Uma forma bastante simples e comum de modelagem do aprendizado de Hebb [SIM2001] é a que segue:

$$\Delta w(n) = \eta\, y(n)\, x(n)$$

onde:

$\Delta w(n)$: variação do peso sináptico no instante n.

η : taxa de aprendizagem.

y(n) : impulso sináptico recebido no instante n.

x(n) : atividade neural (disparo) no instante n.

Muitos modelos artificiais do aprendizado de Hebb seguem o comportamento das tabelas 2.4 e 2.5 em sinapses excitatórias. Conforme mostra o trabalho de diplomação de Lúcio Moser [MOS2001], esses modelos não se prestam para identificar a relação de implicação.

TABELA 2.4 – Aprendizado de Hebb em sinapse excitatória em neurônio artificial do tipo [0,1]

Impulso Sináptico Recebido	***Atividade Neural (disparo)***	***Variação do Peso Sináptico***
0	0	nula
0	1	nula
1	0	nula
1	1	positiva

O aprendizado de Hebb apresentado na tabela 2.5 tanto mais fortalece a sinapse quanto maior for a identidade entre o impulso sináptico e a atividade neural também chamada de disparo. Na tabela 2.6, pode-se observar a modelagem do aprendizado de Hebb proposta por Moser [MOS2001] que fortalece o peso sináptico quando existe relação de implicação entre o impulso sináptico e a atividade neural. Nesse sentido, o impulso sináptico é a causa e a atividade neural é o efeito.

TABELA 2.5 – Aprendizado de Hebb em sinapse excitatória em neurônio artificial bipolar.

Impulso Sináptico Recebido	***Atividade Neural (disparo)***	***Variação do Peso Sináptico***
-1	-1	positiva
-1	1	negativa
1	-1	negativa
1	1	positiva

Entendendo-se que o impulso sináptico e a atividade do neurônio são respectivamente causa e efeito, pode-se traçar um paralelo entre as tabelas 2.6 e 2.7 tendo em vista que a variação da força da tabela 2.7 encaixa-se na variação do peso sináptico da tabela 2.6.

TABELA 2.6 – Aprendizado de Hebb proposto por Moser em sinapse excitatória.

Impulso Sináptico Recebido	***Atividade Neural (disparo)***	***Variação do Peso Sináptico***
0	0	nula
0	1	nula
1	0	negativa
1	1	positiva

Quanto maior for a relação de causa e efeito (implicação), maior será o valor de ***f***. Para o caso binário estudado no presente tópico, se o valor de ***f*** for 1, então em todos os casos percebidos houve a relação de causa e efeito. Se, por outro lado, o valor de ***f*** for 0, então existirá um outro tipo de causalidade: a causa inibe o efeito esperado. Se o valor de ***f*** for 0.5, então não existirá relação de causalidade.

Comparando-se as tabelas 2.6 e 2.7, pode-se observar que o comportamento de ***f*** e ***d*** está de acordo com o comportamento que deve ter o peso da sinapse no aprendizado de Hebb de Moser.

Muitos modelos de aprendizado de Hebb apresentam ponto de saturação[3]. Considerando que ***m*** e ***n*** não são negativos e que $m \leq n$, o valor de ***f*** sempre estará no intervalo fechado [0,1]. Da mesma forma, a crença $d = \frac{m + \frac{k}{2}}{n + k}$ apresenta-se no intervalo aberto (0,1).

3 Ponto de saturação é o limite máximo ou mínimo para o peso sináptico.

TABELA 2.7 – Comportamento de m,n,f e d para causas e efeitos

Causa	***Efeito***	***Variação de m***	***Variação de n***	***Variação da freqüência f***	***Variação da crença d***
0	0	0	0	0 (**nula**)	0 (**nula**)
0	1	0	0	0 (**nula**)	0 (**nula**)
1	0	0	1	$\frac{m}{(n+1)} - \frac{m}{n}$ **(variação ≤ 0)**	$\frac{-(m+\frac{k}{2})}{[(n+k+1)(n+k)]}$ **(variação < 0)**
1	1	1	1	$\frac{(m+1)}{(n+1)} - \frac{m}{n}$ **(variação ≥ 0)**	$\frac{[n+k-(m+\frac{k}{2})]}{[(n+k+1)(n+k)]}$ **(variação > 0)**

2.3.3 Exemplo Introdutório de Agente Popperiano

Para efeito de exemplo, aborda-se um agente com as seguintes características:

- Habita um mundo plano.
- Percebe a sua posição absoluta (x,y).
- Em 80% dos casos que ele executa ação de se mover para a direita, a sua posição no eixo X é incrementada em uma unidade.
- Depois de escolher a ação a ser tomada e antes de agir, considera-se "***causas***" o vetor ("ação a ser tomada", "posição no eixo X", "posição no eixo Y").
- Depois de agir, considera-se "***efeitos***" o vetor ("posição no eixo X", "posição no eixo Y").

O agente popperiano deve ter a capacidade de aprender regras que determinem a relação causal entre ***causas*** e ***efeitos***. De posse desse aprendizado, ele pode planejar seqüências de ações e prever os efeitos a serem encontrados.

Nesse exemplo, o índice dos elementos do vetor ***causas*** começa em zero enquanto que o índice do vetor ***efeitos*** começa em um. Sendo assim, a regra "*se a ação é mover para direita, então incrementa a posição no eixo X*" pode ser representada pela regra:

$$(\textbf{\textit{causas}}_{0} = \text{"mover para direita"}) \Rightarrow (\textbf{\textit{efeitos}}_{1} \leftarrow \textbf{\textit{causas}}_{1} + 1)$$

2.3.4 Exemplo de Indução

O sistema de indução proposto aqui a ser usado pelas criaturas é dividido em três módulos: (1) geração de regras, (2) verificação de regras e (3) previsão baseada em regras.

2.3.4.1Geração de Regras

A geração de regras é bastante simples. Cada regra gerada é válida para um determinado par (Causas, Efeitos). Não existe nenhuma garantia de que uma regra gerada a partir de um único par (Causas, Efeitos) possa ser usada para os demais pares (Causas, Efeitos) percebidos. Quem verifica a validade da regra é o módulo de verificação de regras.

TABELA 2.8 – Exemplo de Causas e Efeitos

Par	Causas			Efeitos	
	$Causa_0$	$Causa_1$	$Causa_2$	$Efeito_1$	$Efeito_2$
1	0	20	10	19	10
2	1	10	10	11	10
3	1	3	3	4	3
4	0	3	3	2	3

Por exemplo, supondo que o sistema de indução seja sucessivamente alimentado com os pares (Causas, Efeitos) da tabela 2.8, o módulo de Geração de Regras poderia gerar as seguintes regras a partir de cada par, entre outras:

Usando o par (Causas, Efeitos) 1:
($Causa_1 = 20$) => ($Efeito_1 \leftarrow 19$)
($Causa_0 < Causa_2$) => ($Efeito_2 \leftarrow Causa_0 + Causa_2$)

Usando o par (Causas, Efeitos) 2:
($Causa_2 = 10$) e ($Causa_2 <> Causa_0$) => ($Efeito_2 \leftarrow 10$)
($Causa_0 = 1$) => ($Efeito_1 \leftarrow Causa_1 + 1$)

Usando o par (Causas, Efeitos) 3:
($Causa_1 = 3$) => ($Efeito_1 \leftarrow Causa_0 + Causa_1$)

Usando o par (Causas, Efeitos) 4:
($Causa_2 = Causa_1$) => ($Efeito_2 \leftarrow Causa_2$)
(Verdadeiro) => ($Efeito_1 \leftarrow 2$)
(Verdadeiro) => ($Efeito_2 \leftarrow Causa_2$)

Existindo um conjunto não vazio de regras, pode-se usar o módulo de verificação de regras para calcular os respectivos valores de confiança e crença.

2.3.4.2Verificação das Regras

Usando-se a mesma tabela 2.8 para calcular os valores de freqüência e crença, obtém-se a tabela 2.9. No presente trabalho, a tabela que armazena as regras e seus respectivos valores de freqüência e crença é chamada de ***tabela de relação***. A tabela de relação poderia muito bem ser chamada de tabela de regras. O nome de tabela de relação persiste tendo em vista que ela é usada para determinar a relação entre causas e efeitos.

TABELA 2.9 – Exemplo de Tabela de Relação

Regra	m	n	f	d
1 ($Causa_1 = 20$) => ($Efeito_1 \leftarrow 19$)	1	1	1	0.66
2 ($Causa_0 < Causa_2$) => ($Efeito_2 \leftarrow Causa_0 + Causa_2$)	4	2	0.5	0.5
3 ($Causa_2 = 10$) e ($Causa_2 <> Causa_0$) => ($Efeito_2 \leftarrow 10$)	2	2	1	0.75
4 ($Causa_0 = 1$) => ($Efeito_1 \leftarrow Causa_1 + 1$)	2	2	1	0.75
5 ($Causa_1 = 3$) => ($Efeito_1 \leftarrow Causa_0 + Causa_1$)	2	1	0.5	0.5
6 ($Causa_2 = Causa_1$) => ($Efeito_2 \leftarrow Causa_2$)	3	3	1	0.8
7 (Verdadeiro) => ($Efeito_1 \leftarrow 2$)	4	1	0.25	0.33
8 (Verdadeiro) => ($Efeito_2 \leftarrow Causa_2$)	4	4	1	0,83

2.3.4.3Previsão Baseada nas Regras

TABELA 2.10 – Problema de Previsão

Par	Causas			Efeitos	
	$Causa_0$	$Causa_1$	$Causa_2$	$Efeito_1$	$Efeito_2$
	1	21	23	?	?

Baseando-se nas regras da tabela de relação, é possível prever efeitos nunca percebidos anteriormente. A previsão dos efeitos é feita com os seguintes passos:

(1)Para cada elemento do vetor Efeitos, seleciona-se as regras em que o lado independente (lado "SE") é verdadeiro e que o lado dependente (lado "ENTÃO") refere-se a esse elemento do vetor Efeitos.

(2)Das regras selecionadas no item anterior, seleciona-se a regra de maior força ou crença dependendo do problema. No presente exemplo, usa-se a regra de maior crença.

(3) Usa-se a regra selecionada no passo anterior para a previsão.

Por exemplo, usando-se a tabela 2.9 para prever o efeito da tabela 2.10, segue-se os passos:

a) Para o $Efeito_1$, seleciona-se as regras em que o lado independente é verdadeiro e que o lado dependente refere-se ao $Efeito_1$. Das regras selecionadas, escolhe-se a regra de maior crença.

TABELA 2.11 – Prevendo o $Efeito_1$

Regra	m	n	f	d
4 ($Causa_0 = 1$) => ($Efeito_1 \leftarrow Causa_1 + 1$)	2	2	1	0.75
7 (Verdadeiro) => ($Efeito_1 \leftarrow 2$)	4	1	0.25	0.33

b) Usa-se a regra de maior crença para determinar o estado de $Efeito_1$ que é $Causa_1 + 1 = 22$.

c) Por fim, para o $Efeito_2$, segue-se os mesmos passos.

TABELA 2.12 – Prevendo o $Efeito_2$

Regra	m	n	f	d
2 ($Causa_0 < Causa_2$) => ($Efeito_2 \leftarrow Causa_0 + Causa_2$)	4	2	0.5	0.5
8 (Verdadeiro) => ($Efeito_2 \leftarrow Causa_2$)	4	4	1	0,83

Para o $Efeito_2$ a regra de maior crença é a regra 8 concluindo-se que $Efeito_2$ vale o mesmo que $Causa_2$. Nesse exemplo, com base nos passos seguidos acima, o módulo de previsão prevê os efeitos (22,23).

O sistema de indução gera regras por uma heurística, calcula a força e a crença dessas regras e usa as regras de maior força ou crença para a previsão. O sistema de indução funciona tanto usando regras de maior força como usando regras de maior crença; porém, a experiência mostra qual das duas métricas é a mais indicada para cada problema. Especula-se que em problemas onde não existe determinismo o uso da crença apresenta melhores resultados.

2.3.4.4Descarte das Regras Inúteis

No exemplo de previsão, as regras usadas foram 4 e 8. Marcando-se as regras usadas, pode-se saber quais são as regras inúteis por não terem marca independentemente da sua crença ou força. Isso permite o uso desse sistema com pouca memória ao descartar as regras inúteis. A geração de regras novas e o descarte de regras não usadas ou pouco usadas criam um ambiente competitivo. As regras mais genéricas tendem a ter crença maior favorecendo a sua permanência em um ambiente competitivo ao nível de regra.

2.3.4.5Notas Finais

Essa seção apresenta a idéia básica do sistema de indução. O algoritmo do sistema de indução é apresentado no Anexo 3 em maiores detalhes. Uma heurística testada com sucesso foi incluída na geração de regras: apenas são geradas regras para efeitos que foram incorretamente previstos. Sendo assim, somente são geradas regras para tentar corrigir erros de previsão anteriormente encontrados. Se o sistema de indução acerta sempre, ele não despende computação para gerar novas regras. O resultado do uso dessa heurística é que o sistema de indução fica mais rápido.

2.3.5 Testes

Voltando ao exemplo anterior ($Causa_0 = 1$) => ($Efeito_1 \leftarrow Causa_1 + 1$), ***Efeito*** de índice ***1*** recebe $Causa_1 + 1$ sempre que o teste ($Causa_0 = 1$) é verdadeiro. O teste de igualdade não é o único teste que o sistema de indução pode gerar. Foram implementados 5 tipos de testes. A forma como o índice do operando é calculado é

chamada de **endereçamento.** Foram implementados os endereçamentos direto e relativo. No endereçamento direto, o índice é a posição absoluta do operando no vetor de causas. No endereçamento relativo, o índice é resultado da ***base*** mais o deslocamento.

Segue a lista dos testes implementados:

- Igualdade "=": testa se os dois operandos são iguais.
- Exemplo 1: ($Causa_0$ = 0). O primeiro operando $Causa_0$ usa endereçamento direto enquanto que o segundo operando é um byte imediato. Somente o teste de igualdade e a ação de atribuição aceitam operandos imediatos.

  Exemplo 2: ($Causa_{base-1}$ = $Causa_{base+1}$). Nesse exemplo, ambos os operadores apresentam endereçamento relativo.
- Diferença "≠": retorna verdadeiro quando os dois operandos são diferentes.
- Maior que ">": retorna verdadeiro quando o primeiro operando é maior que o segundo operando.
- Menor que "<": retorna verdadeiro quando o primeiro operando é menor que o segundo operando.
- Verdade "V" : sempre retorna verdadeiro e não tem operandos.

Os testes binários "≠", ">", "<" aceitam operandos com endereçamento direto ou relativo e não aceitam operandos imediatos.

2.3.6 Operações

Em todas as operações, o resultado é atribuído em $Efeito_{base}$.

Segue a lista de ações possíveis:

- Atribuição simples: $Efeito_{base}$ recebe o operando imediato.

  Exemplo: $Efeito_{base} \leftarrow 1$
- Incremento "inc": $Efeito_{base} \leftarrow Causa_{base} + 1$.
- Decremento "dec" : $Efeito_{base} \leftarrow Causa_{base} - 1$.
- Soma "+": resulta na soma dos dois operandos.

  Exemplo: $Efeito_{base} \leftarrow Causa_{base} + Causa_3$
- Subtração "-": subtrai o segundo operador do primeiro operador.
- Multiplicação "mul": resulta na multiplicação dos dois operandos.
- Divisão "div": se o segundo operando é diferente de zero, divide o primeiro operando pelo segundo.
- Resto da divisão "mod": se o segundo operando é diferente de zero, retorna o resto da divisão do primeiro operando pelo segundo.
- E lógico "and": retorna o resultado da operação ***e*** entre os operandos.
- Ou lógico "or": retorna o resultado da operação ***or*** entre os operandos.
- Ou exclusivo "xor": retorna o resultado da operação ***xor*** entre os operandos.
- Injeção "inj": $Efeito_{base} \leftarrow Causa_{base}$.

- Negação “not”: $\boldsymbol{Efeito_{base}} := \neg\, \boldsymbol{Causa_{base}}$.

As ações binárias ”-”, ”+”, **mul** ,**div**, **mod**, **and**, **or**, **xor** aceitam operandos com endereçamento direto ou relativo e não aceitam operandos imediatos.

2.3.7 Gramática

A formação das regras de implicação induzidas segue as seguintes regras de produção:

< implicacao >	→	<lsttestes> ⇒ <ação>
<lsttestes>	→	<lsttestes> <teste> | <teste>
<teste>	→	V | (<tst>)
<tst>	→	<Operando> <tstOp> <Operando> | <b> = <Operando>
<tstOp>	→	≠ | > | < | =
<ação>	→	$Efeito_{base}$:= <açãofnt>
<açãofnt>	→	(<Operando> <OperaçãoB> <Operando>) | <b> | <OperaçãoU> $Causa_{base}$
<Operando>	→	$Causa_{< i >}$ | $Causa_{base + < i >}$| $Causa_{base - < i >}$
<OperaçãoB>	→	+ | - | mul | div | mod | and | or | xor
<OperaçãoU>	→	inc | dec | inj | not
< i >	→	número natural
< b >	→	byte

2.4 Testes da Indução de Função Lógica não Determinista

Todos os testes foram realizados em uma máquina com processador AMD K6II-500Mhz, 128 MB de RAM, barramento frontal de 100Mhz e sistema operacional Windows 95. Nenhum dos experimentos usou memória de massa.

2.4.1 Contador de 2 Bits

Escolheu-se o contador de 2 bits como primeiro teste por sua simplicidade e ao mesmo tempo oferecer características interessantes. O bit menos significativo do contador simplesmente é sucessivamente negado durante a contagem. O próximo estado do bit menos significativo depende exclusivamente de seu estado atual ou simplesmente $Efeito_0 = \neg Causa_0$.

Por outro lado, o próximo estado do bit mais significativo depende de ambos os bits do contador. Sendo assim, existem duas funções lógicas distintas para induzir ao mesmo tempo: uma para cada bit. Uma entrada da tabela de relação que relaciona apenas um elemento do vetor ***Causas*** e um elemento do vetor ***Efeitos*** é suficiente para induzir a função lógica do bit menos significativo; porém, para formar a regra do bit mais significativo, é necessária uma regra que relacione os dois elementos do vetor ***Causas*** com o elemento $\mathbf{e_1}$ do vetor ***Efeitos***. Os estados do contador são listados na tabela 2.13.

O primeiro teste feito usou o algoritmo da figura 2.5. Nele, espera-se que o algoritmo aprenda a prever o próximo estado do contador de dois bits. Nesse algoritmo, o sistema de indução é alimentado com as causas pelo procedimento ***Pred*** retornando uma previsão para essas causas. Os efeitos são alimentados pela função ***Encontrou*** retornando o erro de previsão que é a diferença entre os efeitos anteriormente previstos e

os erros encontrados. ***Pred*** e ***Encontrou*** são executados até que nenhum erro de predição seja detectado.

TABELA 2.13 – Causas e Efeitos para Induzir Contador de 2 Bits

i	***Causas[i]***		***Efeitos[i]***	
	c_1	c_0	e_1	e_0
0	0	0	0	1
1	0	1	1	0
2	1	0	1	1
3	1	1	0	0

Conforme é apresentado subseqüentemente ao logo do texto, uma heurística para geração de regras é estudada: geração de regras apenas para causas não nulas. Foi introduzida uma "*flag*" chamada de ***TodaOperação*** que indica quando esta heurística é usada**.** ***TodaOperação*** está no estado "verdadeiro" quando apenas as causas não nulas podem ser usadas para gerar novas regras.

```
1. Repita
2.     ErrosPredicao ← 0
3.     Para I=0 até 3 faça
4.          Pred(Causas[I],EfeitosPreditos)
5.          ErrosPredicao ← ErrosPredicao +
                              Encontrou(Efeitos[I])
6.          fim-faça
7.    imprime(ErrosPredicao)
8. até (ErrosPredicao=0)
```

FIGURA 2.5 – Algoritmo para Testar Indução de Contador

Em um teste, usando o algoritmo da figura 2.5 com uma tabela de relação com 4 entradas, o sistema de indução gerou as seguintes regras:

```
1. B=0: (1 = C[0]) => E[B] := C[B] - 1 [ f=1 Vit=43 UltVit=92 ]
2. B=0: V => E[B] := 1 [ f=0,470588235294118 Vit=5 UltVit=91 ]
3. B=1: V => E[B] := (C[0] xor C[1]) [ f=1 Vit=4 UltVit=92 ]
4. B=1: V (1 = C[B-1]) => E[B] := (C[0] - C[B]) [ f=1 Vit=1
   UltVit=86 ]
```

Nas regras induzidas acima, observa-se a causa ***C***, o efeito ***E,*** a base ***b,*** a força ***f ,*** o número de vitórias ***Vit*** e o ciclo onde ocorreu a última vitória ***UltVit***. A figura 2.6 mostra uma tentativa de diagramar as regras induzidas de forma semelhante a uma rede neural.

O algoritmo da figura 2.5 foi executado 1000 vezes usando tabela de relação de 4 elementos e em média o algoritmo não encontrou erro de predição na décima quarta iteração do laço ***Repita***. O tempo total de computação despendido para executar 1000 vezes o algoritmo da figura 2.5 foi um pouco mais do que 4 segundos. Nesse tempo, os módulos ***Pred*** e ***Encontrou*** foram chamados 4000 vezes.

Em princípio, quanto maior for a tabela de relação, maior será o tempo de UCP gasto na execução dos módulos ***Pred*** e ***Encontrou***; porém, freqüentemente, quanto maior for a tabela de relação, menos chamadas aos módulos ***Pred*** e ***Encontrou*** são necessárias para induzir as regras de interesse. Por exemplo, executando o algoritmo da figura 2.5 1000 vezes com 8 entradas na TR, usou-se dois segundos de UCP e em média apenas 6,56 ciclos no laço ***Repita*** em cada execução, por outro lado, executando-se o mesmo algoritmo com 3200 entradas na TR, encontrou-se 6,51 ciclos em média e 39,4 segundos de tempo de execução. Esses números podem ser conferidos na tabela 2.14.

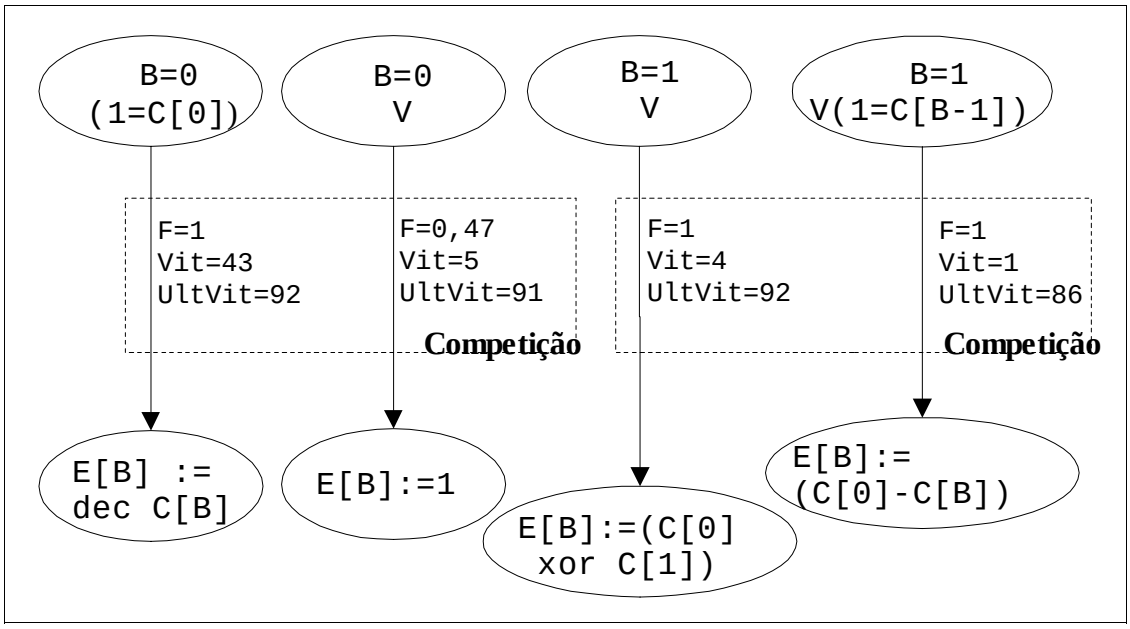

FIGURA 2.6 – Regras Induzidas

Comparando-se as tabelas 2.14 e 2.15, observa-se que o parâmetro ***TodaOperação*** desempenha papel crucial no desempenho do algoritmo. Lembrando que ***TodaOperação*** define se o valor zero deve ou não ser usado para gerar novas operações, o algoritmo apresenta melhor desempenho com ***TodaOperação*** em verdadeiro tendo em vista que o valor zero carrega informação útil no contador para causas ou efeitos.

TABELA 2.14 – Execuções do algoritmo da figura 2.5 usando os parâmetros:
número de execuções = 1000, TP = 3 e ***TodaOperação*** = Verdadeiro.

Tamanho da Tabela de Relação	***tempo de UCP (segundos)***	***n. total de ciclos no laço Repita***	***n. de ciclos médio do laço repita por execução***
4	4.74	14511	14.51
6	2.61	7027	7.03
8	2.52	6562	6.56
16	2.79	6378	6.38
32	3.01	6408	6.41
320	5.21	6643	6.64
3200	39.4	6513	6.51

Quando a tabela de relação está lotada de regras e uma nova regra deve ser gerada, uma regra é escolhida para ser descartada. A regra a ser descartada é a pior regra de ***TP*** regras escolhidas ao acaso. ***TP*** é um número natural que indica o ***tamanho da procura*** pela pior regra. É considerada a pior regra a regra que nunca foi usada utilmente ou foi usada utilmente pela última vez há mais tempo.

TABELA 2.15 – Execuções do algoritmo da figura 2.5 usando os parâmetros: ***número de execuções = 1000, TP*** = 3 e ***TodaOperação*** = Falso.

Tamanho da Tabela de Relação	***tempo de UCP (segundos)***	***n. total de ciclos no laço Repita***	***n. de ciclos médio do laço repita por execução***
4	5.16	16894	16.89
6	4.39	12984	12.98
8	5.32	14787	14.79
16	7.79	18809	18.81
32	12.96	24043	24.04
320	89.42	45321	45.32

Analizando-se a tabela 2.16, observa-se que a medida que o tamanho da procura ***TP*** cresce, o número de ciclos diminui. Na figura A02, observa-se que o função ***EscolhePiorRelação*** é chamada pelo procedimento ***SatFreq***. O tempo de UCP gasto pela função ***EscolhePiorRelação*** é diretamente proporcional ao tamanho da procura. Sendo assim, o tempo de UCP gasto pelo procedimento ***SatFreq*** e por conseguinte o tempo de todo o algoritmo cresce com o tamanho da procura. Essa relação pode ser verificada na tabela 2.16.

Conforme mostra a figura 2.7, a parte central do algoritmo da figura 2.5 foi facilmente implementada em Object Pascal. A tela do programa pode ser vista na figura 2.8. Ainda que o problema em questão seja bem simples, o resultado encontrado atendeu às expectativas.

TABELA 2.16 – Execuções do algoritmo da figura 2.5 usando os parâmetros: ***número de execuções = 1000, TR = 8*** e ***TodaOperação*** = Verdadeiro.

TP ***Tamanho da Procura***	***tempo de UCP (segundos)***	***n. total de ciclos no laço Repita***	***n. de ciclos médio do laço repita por execução***
1	2.84	7825	7.83
2	2.58	6886	6.89
3	2.52	6562	6.56
4	2.48	6531	6.53
100	2.86	6380	6.38
1000	42.41	6424	6.42

```
for I:=0 to 3 do
    begin
    MR.Predicao(csCausas[I],csCausas[I],P);
    ERRO:=ERRO+MR.Encontrou(csEfeitos[I]);
    end;
```

FIGURA 2.7 – Implementação em Object Pascal

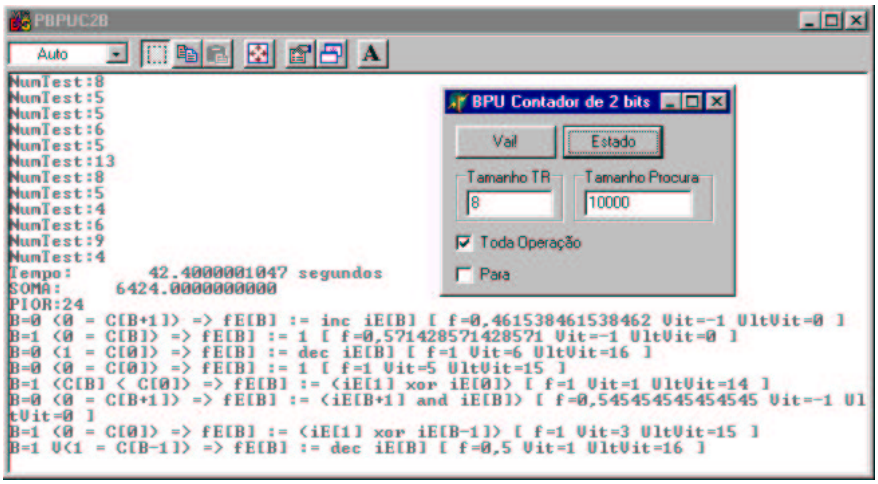

FIGURA 2.8 – Tela do programa Contador de 2 bits

2.4.2 Bit que Vai e Vem

O experimento apresentado no tópico anterior apresentava poucos bits enquanto que se deseja implementar criaturas popperianas com dezenas de bits de percepção. Sendo assim, o teste seguinte escolhido é o teste da previsão do *"bit que vai e vem"*. Na forma como foi implementado, esse teste usa 34 bits para a causa e mais 34 bits para o efeito totalizando 68 bits.

O presente teste pode ser descrito da seguinte forma: dos 34 bits da causa, apenas dois bits podem estar no estado 1 sendo que os outros devem estar todos no estado 0. O bit ***"seletor"*** obrigatoriamente deve ocupar a posição 1 ou 2 enquanto que o bit ***"transeunte"*** deve obrigatoriamente ocupar um dos últimos 30 bits. Quando o bit seletor está na posição 1, o bit transeunte deve *"caminhar"* para esquerda; quando o bit seletor estiver na posição 2, o bit transeunte deve *"caminhar"* para direita.

Tempo	Causas	Efeitos
	0 1 2 0123456789012345678901234567890	0 1 2 0123456789012345678901234567890
0	10000000000000000000000001000000000	00000000000000000000000010000000000
1	01000000000000000000000010000000000	00000000000000000000000001000000000
2	01000000000000000000000001000000000	00000000000000000000000000100000000
3	10000000000000000000000000100000000	00000000000000000000000000010000000

FIGURA 2.9 – Evolução do Bit que Vai e Vem

Na figura 2.9, observa-se o movimento do bit transeunte de acordo com o bit seletor. Não existe bit seletor no efeito.

O algoritmo usado para testar a indução do Bit que Vai e Vem é apresentado na figura 2.10 . Considerando que existe uma função lógica para cada bit do efeito a ser induzida a partir da causa, nesse experimento, foram induzidas 34 funções lógicas ao mesmo tempo. O desempenho do algoritmo pode ser verificado nas tabelas 2.17, 2.18 e 2.19.

Comparando-se as tabelas 2.17 e 2.18, observa-se que o parâmetro ***TodaOperação*** tem grande importância no desempenho do algoritmo. Isso ocorre tendo em vista que nesse experimento o valor zero não representa informação útil. Sendo assim, com ***TodaOperação*** em falso, somente os valores diferentes de zero são usados para formar novas implicações.

TABELA 2.17 – Execuções do algoritmo da figura 2.10 usando os parâmetros: ***número de execuções = 100, TP*** = 20 e ***TodaOperação*** = Falso.

Tamanho da Tabela de Relação	***tempo de UCP (segundos)***	***n. total de ciclos no laço Repita***	***n. de ciclos médio do laço repita por execução***
50	271.01	20769	207.69
60	117.27	8007	80.07
70	84.25	5235	52.35
80	61.31	3693	36.93
90	33.27	2008	20.08
100	12.97	815	8.15
200	10.42	545	5.45
1000	20.88	521	5.21
10000	169.38	530	5.3

Na tabela 2.19, observa-se a influência do tamanho da procura ***TP*** sobre o desempenho geral do algoritmo. Quando o tamanho da procura ***TP*** fica muito grande passando do tamanho da tabela de relação ***TR***, freqüentemente, a função ***EscolhePiorRelação*** escolhe sempre a mesma entrada da ***TR*** como pior. Sendo assim, o algoritmo entra em laço infinito. Quanto menor for o tamanho de ***TP*** em relação ao tamanho da ***TR,*** menos determinístico o algoritmo fica e menor é a probabilidade de entrar em um laço de execução infinito.

TABELA 2.18 – Execuções do algoritmo da figura 2.10 usando os parâmetros: ***número de execuções = 1000, TP*** = 20 e ***TodaOperação*** = Verdadeiro.

Tamanho da Tabela de Relação	***tempo de UCP (segundos)***	***n. total de ciclos no laço Repita***	***n. de ciclos médio do laço repita por execução***
90	560.12	21728	217.28
200	372.84	7926	79.26
1000	354.37	2773	27.73

Usando-se os parâmetros da tabela 2.19, não foi possível encontrar solução com ***TP*** = 1000 tendo em vista a ocorrência de laços infinitos de execução. Com o valor ***TP*** muito grande, o algoritmo freqüentemente escolhe a regra mais recentemente formada como pior regra tendo em vista que ela não tem vitórias. Sendo assim, as regras velhas continuam a existir e as novas não permanecem fazendo com que o algoritmo não progrida em direção da solução.

TABELA 2.19 – Execuções do algoritmo da figura 2.5 usando os parâmetros: ***número de execuções = 100, TR = 100*** e ***TodaOperação*** = Falso.

TP ***Tamanho da Procura***	***tempo de UCP (segundos)***	***n. total de ciclos no laço Repita***	***n. de ciclos médio do laço repita por execução***
1	19.98	1209	12.09
2	15.71	957	9.57
3	14.39	898	8.98
4	15.26	951	9.51
100	12.25	767	7.67

```
DefineNumCausasEfeitos(34,34)
Causas ← Percepção()
repita
    ErrosPredicao ←  0
    POS ← 15
    para I=1 até 60 faça
        // prepara vetor de causas
        Atribui 0 para cada elemento dos vetores causas e efeitos
        se I > 30
           então Causas[1] ← 1
           senão Causas[2] ← 1
        Causas[POS+4] ← 1
        // predição
        Pred(Causas,EfeitosPreditos)
        // prepara o vetor de efeitos
        se Causas[1]=1
           então faça
                 POS ← POS-1
                 se POS<0
                    então POS ← 29
                 fim-faça
           senão POS ← (POS + 1) mod 30
        Efeitos[POS+4] ← 1
        // testa predição
        ErrosPredicao ← ErrosPredicao + Encontrou(Efeitos)
    fim-para
   imprime(ErrosPredicao)
até (ErrosPredicao=0)
```

FIGURA 2.10 – Algoritmo para Testar Indução do Bit que Vai e Vem

Nesse experimento, o sistema de indução atingiu os objetivos esperados.

2.4.3 Comparação com o Modelo Neural Combinatório

O sistema de indução apresentado nesse capítulo apresenta algumas semelhanças com o modelo neural combinatório (do inglês, *Combinatorial Neural Model* – CNM) [MAC89]. Conforme mostra a figura 2.11, a rede CNM apresenta três camadas: (1) camada de entrada ou de evidências, (2) camada combinatorial e (3) camada de saída. A rede CNM é usada para classificar evidências apresentando uma fase de aprendizado supervisionado.

Na camada de entrada da rede CNM, cada neurônio representa uma evidência que pode ser verdadeira ou falsa. Na camada intermediária, existe um neurônio para cada combinação de no mínimo duas evidências verdadeiras sendo que a saída de cada neurônio dessa camada é o resultado da função ***and***. A camada de saída é uma camada competitiva onde existe um neurônio para cada classe. Existe uma sinapse que liga cada neurônio da primeira e segunda camadas com um neurônio da camada de saída.

A rede CNM apresenta uma fase de treinamento supervisionada. Cada sinapse possui um acumulador. Quando os neurônios de ambos os lados da sinapse representam informação verdadeira, o acumulador é incrementado em uma unidade. Ao contrário, quando um neurônio representa uma evidência ou combinação de evidências verdadeiras e outro neurônio representa a classe errada, o acumulador da sinapse que liga os neurônios é decrementado em uma unidade.

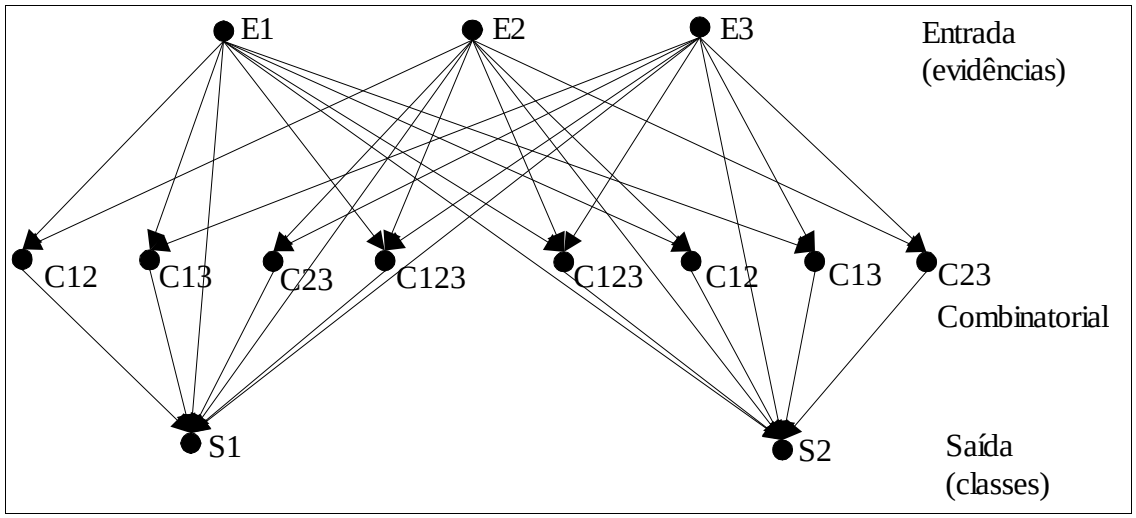

FIGURA 2.11 – Rede CNM

Após o aprendizado, quanto maior for o valor do acumulador, mais significa que ele é *verdadeiro.* Ao contrário, quanto mais negativo ele for, mais *falso* ele é. Após a fase de aprendizagem, é feita a poda das sinapses em que o acumulador não atingiu um determinado limiar.

Voltando ao sistema de indução apresentado aqui, observando-se a figura 2.6, verifica-se que cada entrada da TR representa: a relação entre uma combinação de causas (evidências) e sua conseqüência (classe). A rede CNM e o sistema de indução apresentado aqui possuem as seguintes características em comum: (1) aprendizado supervisionado por reforço que atende a tabela 2.6 permitindo a indução da implicação, (2) ambos combinam evidências verdadeiras e (3) ambos têm uma camada competitiva

na saída. Vale observar que o lado independente das regras da TR corresponde à camada combinatorial do CNM enquanto que o lado dependente corresponde à camada de saída. Essa conclusão pode ser encontrada ao comparar-se as figuras 2.6 e 2.11.

Em alguns casos, a rede CNM tem vantagens sobre o sistema de indução apresentado aqui: (1) é simples e fácil de implementar, (2) em muitos problemas, encontra a solução rapidamente.

Por outro lado, em problemas em que existam muitas evidências e classes (causas e efeitos), o sistema de indução apresentado aqui despende muito menos memória tendo em vista que os efeitos e combinações de causas são alocadas durante a execução.

Uma rede CNM que combine no máximo duas evidências (causas) em um espectro de trinta usaria C(30,2)*30 + 30 + 30 = 13110 neurônios para aprender o comportamento do *"Bit que Vai e Vem"*. A mesma rede CNM combinando até cinco evidências apresenta um número de neurônios igual a:

(n. de combinações)* n. de classes + n. de evidências + n. de classes =

(C(30,5) + C(30,4) + C(30,3) + C(30,2)) *30 + 30 + 30 =

(142506 + 27405 + 4060 + 435) *30 +30 + 30 =

5232180 + 30 + 30 = 5232240 neurônios

Por outro lado, conforme mostra a tabela 2.17, o sistema de indução apresentado conseguiu induzir o comportamento do *"Bit que Vai e Vem"* despendendo apenas 50 entradas na tabela de relação. Na TR, cada regra é armazenada juntamente com o número de vezes que ela foi usada com sucesso além da informação temporal que indica quando ela foi usada com sucesso pela última vez. Com essa informação, é possível estimar quais são as entradas da TR menos e mais úteis. Sendo assim, as entradas da TR menos úteis são descartadas enquanto que as mais úteis para a indução são preservadas. Além disso, somente combinações de causas percebidas são alocadas não sendo necessário alocar todas as combinações possíveis previamente.

Por fim, é importante ressaltar que nessa comparação não foram abordadas capacidades mais complexas do sistema de indução como a capacidade de induzir regras que envolvam soma, incremento, operações lógicas e testes.

2.4.4 Comparação com Redes Bayesianas

As redes bayesianas são grafos orientados acíclicos onde os arcos representam a relação de causalidade probabilística enquanto que os nodos representam variáveis ou estados. A figura 2.12 apresenta um exemplo de rede Bayesiana. Nesse exemplo, a probabilidade de "chuva" e "feriado" ocorrerem é de 0.15 e 0.01 respectivamente. Se ocorrerem "chuva e feriado", a probabilidade de ocorrer "JP na praia" é de 0.2.

A relação causal probabilística é uma característica que também é encontrada nas regras geradas pelo sistema de indução proposto na presente dissertação; porém, pode-se encontrar diferenças entre os modelos. Uma diferença é que enquanto uma rede bayesiana é acíclica, não existe nenhuma restrição para que o sistema de indução gere uma regra do tipo (a $\Rightarrow$ b) e (b $\Rightarrow$ a) . A regra (a $\Rightarrow$ b) gerada pelo sistema de indução deve ser lida como: " quando ***a*** ocorre, então ***b*** ocorrerá.

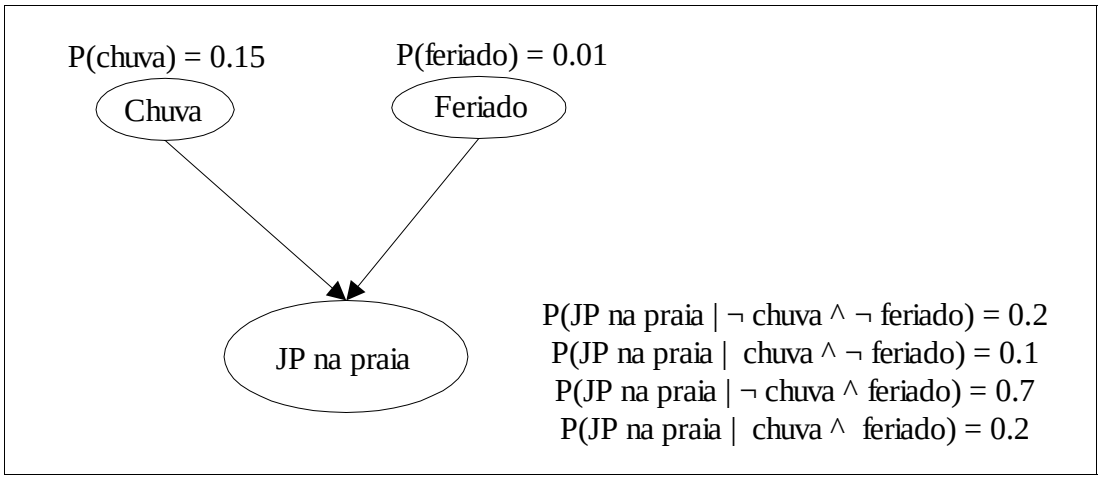

FIGURA 2.12 – Exemplo de Rede Bayesiana

Pode-se estabelecer um paralelo entre as probabilidades apresentadas nas redes bayesianas e as regras do sistema de indução. A probabilidade

$$P(JP\ na\ praia \mid chuva \wedge feriado) = 0.2$$

poderia ser entendida como uma regra do tipo

$$chuva \wedge feriado \Rightarrow JP\ na\ praia\ [\ f = 0.2\ ].$$

Observa-se que na rede bayesiana da figura 2.12 existe uma probabilidade de *"JP na praia"* para cada estado de *"chuva"* e *"feriado"*. Cada nodo da rede bayesiana deve ter a probabilidade de todas as combinações de causas possíveis; porém, o sistema de indução proposto aqui, gera apenas regras para as causas mais freqüentes.

Conforme mostra a figura 2.6, cada regra da tabela de relação assemelha-se a um neurônio que compete com outros sendo que o mais forte vence. Por tal motivo, o sistema de indução apresentado no presente texto assemelha-se muito mais a uma rede de neurônios competitiva onde "o vencedor leva tudo" do que uma rede bayesiana.

2.5 Sistema de Planejamento

Os tópicos 2.3 e 2.4 apresentaram a forma como as criaturas desenvolvidas no presente trabalho induzem ou aprendem as regras do ambiente; porém, além de aprender as regras do ambiente, a criatura popperiana planeja ações ou séries de ações e prevê a satisfação a ser alcançada pelas mesmas.

Ainda que por motivos de engenharia de software o sistema de planejamento tenha sido desenvolvido de forma independente do sistema de indução, a criatura popperiana usa o sistema de planejamento de forma integrada ao sistema de indução tendo em vista que o sistema de planejamento define seqüências de ações enquanto que o sistema de indução responde com o resultado a ser alcançado de cada ação proposta pelo sistema de planejamento. Certamente, falhas produzidas pelo sistema de indução poderão implicar em falhas no planejamento. Criaturas popperianas muito jovens com poucos ciclos de vida não dispõem de treinamento suficiente para induzir corretamente as regras percebidas. Sendo assim, criaturas popperianas jovens não conseguem planejar com eficiência.

O primeiro sistema de planejamento imaginado foi baseado em busca em árvore de estados alcançáveis através de ações. O problema desse tipo de abordagem reside no fato de que é praticamente impossível gerar e analisar uma árvore com grande profundidade. Por exemplo, para gerar uma seqüência de 30 ações seria necessária uma árvore de profundidade 30. Uma outra opção seria o uso de alguma heurística que indicasse a aproximação do estado desejado ou descartasse estados da árvore improváveis de levarem a satisfação. O problema em usar uma heurística é que o sistema de indução apenas induz regras de transição de estados do tipo "*causa implica em conseqüência*" e não regras do tipo "*causa aproxima da conseqüência*". Sendo assim, o sistema de indução não consegue por conta própria prever que determinada ação implica na aproximação de um estado desejável o qual trará satisfação.

É importante frisar que a criatura popperiana artificial deve aprender as regras do ambiente e fazer planejamentos nesse ambiente por conta própria. Sendo assim, os sistemas de indução e planejamento devem ser genéricos o suficiente para trabalhar em ambientes sob os quais eles não tem nenhuma informação prévia. Para efeito de modelagem da criatura popperiana, o sistema de planejamento deve obedecer a sérias restrições:

- As transições de estados usadas no planejamento são as previstas pelo sistema de indução.
- Sem planejamento prévio, não existe informação sobre o número de ações ou outra forma de medição que diga a distância entre um determinado estado de outro estado desejável. Sendo assim, não existe como o sistema de planejamento afirmar que determinada ação vai aproximar a criatura de um estado desejável impossibilitando a criação de heurísticas pela criatura.

Considerando as restrições apresentadas, a estratégia de planejamento escolhida foi a busca cega pela satisfação. A função de satisfação é distinta para espécies de agentes popperianos distintos. Por exemplo, robôs mineradores podem encontrar satisfação ao carregar e descarregar minérios enquanto que robôs predadores podem perceber satisfação ao encontrar a presa. Mesmo usando busca cega, os agentes popperianos implementados e testados apresentaram bom desempenho. Os resultados desses experimentos são apresentados na seção Experimentos com Criaturas Popperianas.

Para efeito de modelo, cabe afirmar que cada espécie de agente popperiano terá a sua própria função de satisfação. Para efeito de implementação, no presente trabalho, o sistema de planejamento deve ser o mesmo para qualquer espécie de agente popperiano sendo que a função de satisfação será ligada dinamicamente em tempo de execução para cada espécie de agente.

A figura 2.13 apresenta em detalhes a escolha de ação baseada em planejamento vista superficialmente na figura 2.2. A ação baseada em planejamento é tomada apenas se existe plano definido para o estado atual.

Os planos são criados usando-se o estado atual como ponto de partida. A construção de um plano ocorre com sucesso quando o sistema de indução prevê que o plano leva a um estado de satisfação ou a outro plano que leve para a satisfação.

Conforme é examinado nas seções seguintes, por ser o planejamento cego, é improvável que um plano recém gerado apresente a melhor seqüência de ações possíveis entre um estado inicial qualquer e a satisfação. Sendo assim, existe um módulo dedicado

apenas para otimização de planos já existentes. As próximas seções apresentam em detalhes o sistema de planejamento.

O sistema de planejamento foi implementado em 700 linhas de código fonte. Não serão apresentados aqui todos os detalhes de implementação, mas somente os aspectos mais importantes da estrutura de dados, algoritmo dos módulos mais importantes e alguns experimentos.

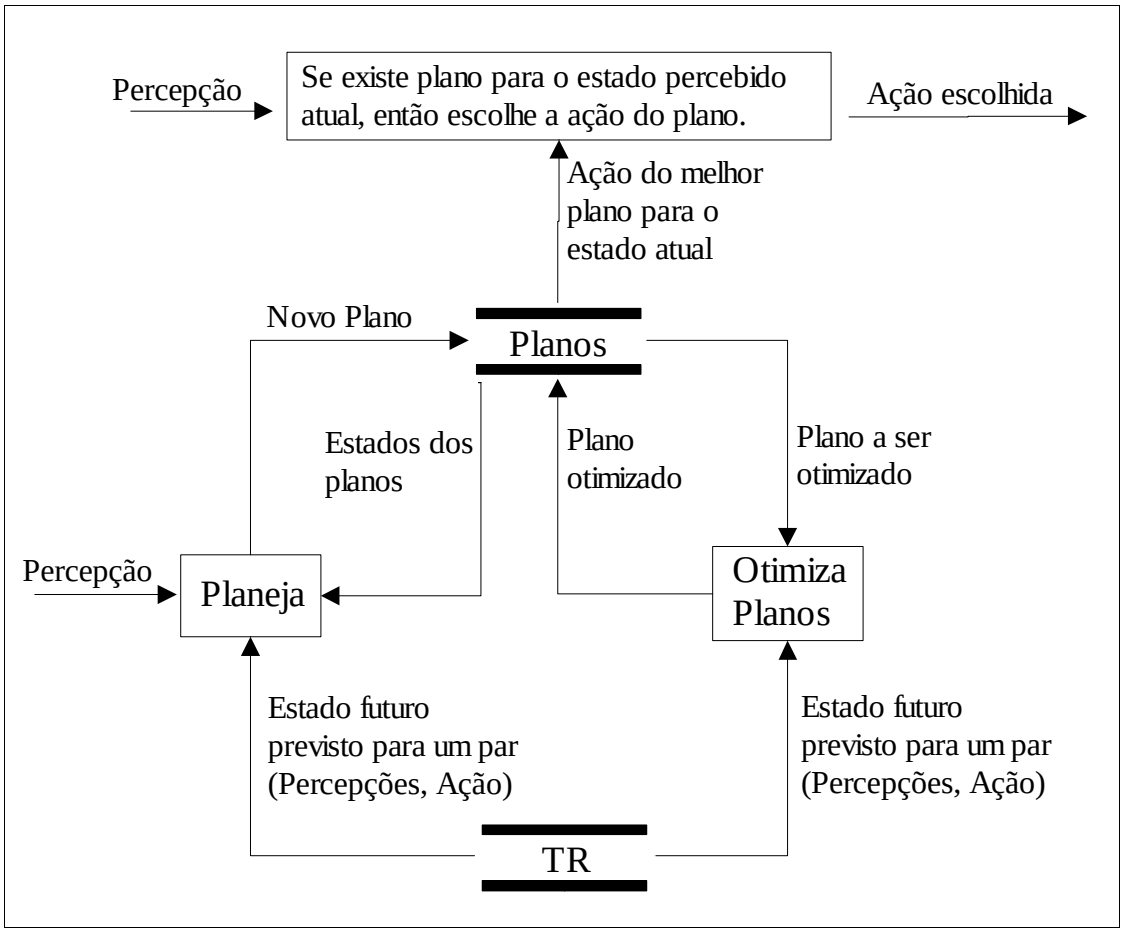

FIGURA 2.13 – Diagrama da Escolha da Ação

2.5.1 Estrutura de Dados

A estrutura principal do sistema de planejamento é o vetor ***Planos***. O vetor ***Planos*** é um vetor onde cada célula é outro vetor do tipo ***Plano***. A tabela 2.20 mostra um exemplo de vetor do tipo ***Plano***. Cada célula do vetor ***Plano*** é um par constituído por uma ação e o estado a ser atingido com essa ação. Por exemplo, a ação de índice *I* provoca a transição de estados do estado de índice *I*-1 para o estado de índice *I*. De acordo com a tabela 2.20, uma criatura que se encontre no estado (2,128,15,4) deverá tomar a ação 3 para atingir o estado (2,127,15,4) e posteriormente a ação 1 para atingir o estado de satisfação.

Pode ser traçado um paralelo entre o planejamento proposto aqui e o esquema usado por Muñoz [MUÑ99] estudado na seção 1.8. Esse esquema é composto por um contexto, uma ação e um resultado. No vetor ***Plano,*** o estado índice *I*-1 é o contexto enquanto que o estado de índice *I* é o resultado. Sendo assim, a tripla (contexto, ação,

resultado) é entendida como($Estado_{(I-1)}$, $Ação_{(I)}$, $Estado_{(I)}$).

TABELA 2.20 – Exemplo de Vetor do tipo Plano

Índice	*Ação*	*Estado*
0	-	(3,128,15,4)
1	2	(2,128,15,4)
2	3	(2,127,15,4)
3	1	(3,127,15,5) satisfação

2.5.2 Algoritmo

O sistema de planejamento foi implementado em uma classe Object Pascal em que sua funcionalidade está distribuída entre seus diversos métodos sendo os mais importantes ***PlanejaCegamente***, ***OtimizaCegamente*** e ***PodeAgir***. Tais métodos são descritos a seguir:

- ***PodeAgir(EstadoAtual,Ação)***: retorna verdadeiro se existe ***Ação*** planejada para o ***EstadoAtual***. Nesse caso, o parâmetro ***Ação*** retorna a ação a ser tomada. O parâmetro ***EstadoAtual*** é de entrada enquanto que ***Ação*** é parâmetro de saída. Internamente, ***PodeAgir*** varre todos os planos do vetor ***Planos*** a procura do ***EstadoAtual***. Dos planos que contêm o ***EstadoAtual,*** é selecionado o plano que mais rapidamente leva a satisfação. O algoritmo da função ***PodeAgir*** encontra-se na figura 2.14.

```
Função PodeAgir(EstadoAtual,Ação)
PlanosSelecionados ← seleciona os planos do vetor Planos que possuem
                     o EstadoAtual
Se existe pelo menos 1 plano em PlanosSelecionados
   então faça
         MelhorPlano ← seleciona o plano dos PlanosSelecionados que
                       leva mais rapidamente a satisfação.
         Ação ← seleciona a próxima ação de acordo com o MelhorPlano
         retorna verdadeiro
         fim-faça
   senão retorna falso
fim-função
```

FIGURA 2.14 – Algoritmo da Função PodeAgir

- ***PlanejaCegamente(Plano,EstadoInicial,Profundidade,FunçãoDePredição, NúmeroDeAções)***: essa função retorna verdadeiro se encontrou um plano que leve à satisfação retornando no parâmetro ***Plano*** o plano achado. Resumidamente, a busca cega foi implementada da seguinte forma partindo-se do ***EstadoInicial***: (1) escolhe-se uma ação ao acaso, (2) usa-se a ***FunçãoDePredição*** para prever o estado a ser alcançado com a ação escolhida e (3) se atingiu ***Profundidade*** ou não encontrou satisfação com o estado alcançado, volta ao passo 1. O parâmetro ***NúmeroDeAções*** indica o número de ações existentes. Os parâmetros ***EstadoInicial, Profundidade, FunçãoDePredição*** e ***NúmeroDeAções*** são parâmetros de entrada enquanto que ***Plano*** é parâmetro de saída. A ***FunçãoDePredição*** apresentada na figura 2.15 e passada como parâmetro na função ***PlanejaCegamente*** recebe o par

(***Ação***, ***Estado***) e retorna verdadeiro quando prediz satisfação além de retornar o próximo estado a ser alcançado em ***Estado***.

```
função FunçãoDePredição(Ação, Estado)
Estado[i] ← Ação
Pred(Estado,EfeitoEsperado)
Estado ← EfeitoEsperado
se AchouSatisfação(EfeitoEsperado)
   então retorna Verdadeiro
   senão retorna Falso
fim-função
```

FIGURA 2.15 – Algoritmo da função FunçãoDePredição

É interessante observar que a ***FunçãoDePredição*** chama o procedimento ***Pred*** do sistema de indução conforme pode ser visto na terceira linha do algoritmo apresentado na figura 2.15. O algoritmo da função ***PlanejaCegamente*** pode ser visto na figura 2.16.

```
função PlanejaCegamente(Plano,EstadoInicial,
                        Profundidade,FunçãoDePredição,
                        NúmeroDeAções)
Estado ← EstadoInicial
para Ciclos = 1 até Profundidade
     faça
     Se existe ação que leva a satisfação nesse passo
        então faça
              inclui essa ação e o estado alcançado no vetor Plano
e
              sai do procedimento
              fim-faça
     Ação ← escolhe probabilisticamente uma ação dando preferência
            a ações que não levem a um estado já existente no
            vetor Plano
     Achou ← FunçãoDePredição(Ação, Estado)
     inclui (Ação,Estado) no vetor Plano
     se Achou
       então faça
              marca o vetor Plano como válido
              sai do procedimento
              fim-faça
     fim-faça
fim-procedimento
```

FIGURA 2.16 – Algoritmo do função ***PlanejaCegamente***

- ***OtimizaCegamente (Plano, Profundidade, FunçãoDePredição, NúmeroDeAções)***: o procedimento ***OtimizaCegamente*** escolhe um trecho do vetor de entrada ***Plano*** e tenta substituir esse trecho por outro trecho menor de tamanho não superior à ***Profundidade.*** Esse trecho é criado cegamente usando o ***NúmeroDeAções*** permitido. Se não consegue otimizar, retorna o plano original. Os parâmetros ***Profundidade***, ***FunçãoDePredição*** e ***NúmeroDeAções*** são de entrada enquanto que ***Plano*** é de entrada e saída. O algoritmo desse procedimento pode ser visto na figura 2.20.

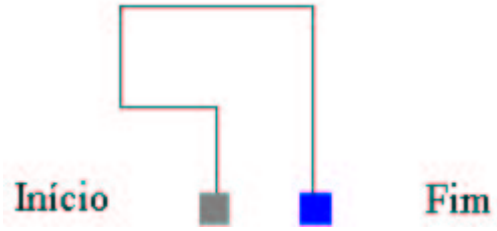

FIGURA 2.17 – Exemplo de plano a ser otimizado

Para efeito de exemplo, a figura 2.17 mostra um tracejado que representa um plano a ser otimizado que vai do “Início” ao “Fim”. O trecho que vai desde o quadrado verde até o quadrado azul representa o trecho do plano a ser otimizado.

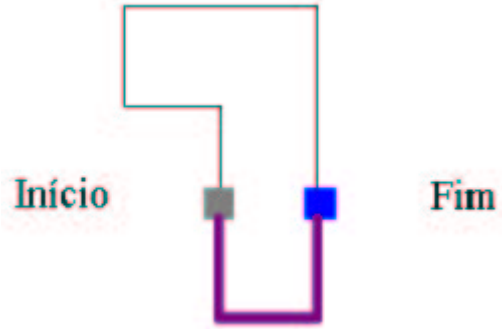

FIGURA 2.18 – Exemplo de plano e otimização

A figura 2.18 mostra o plano original e a otimização do trecho em linha grossa que poderia ter sido feita pelo procedimento ***OtimizaCegamente***. A figura 2.19 mostra o resultado da otimização.

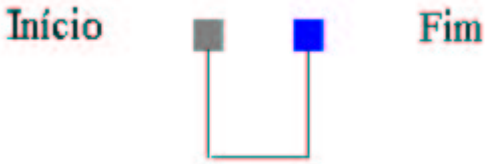

FIGURA 2.19 – Plano otimizado

Enquanto as figuras 2.17, 2.18 e 2.19 foram desenhadas manualmente para efeitos didáticos, as figuras 2.21, 2.22, 2.23 e 2.24 foram retiradas de um experimento implementado. A figura 2.21 mostra esquematicamente um plano que é fruto de planejamento cego que representa uma rota entre 2 pontos distantes de mais de 60 unidades em um mundo discreto e bidimensional que só permite 4 tipos de movimentos: para cima, para baixo e para os lados. Esse plano apresenta muitas inflexões justamente por ter sido gerado ao acaso podendo ser otimizado. As figuras 2.22, 2.23 e 2.24 mostram o resultado do processo de otimização descrito no presente texto sobre o plano da figura 2.21. Foram necessárias milhares de chamadas da função ***OtimizaCegamente*** para que o plano da figura 2.21 fosse otimizado até chegar ao estado da figura 2.24.

```
Procedimento OtimizaCegamente(Plano,
                              Profundidade,FunçãoDePredição,
                              NúmeroDeAções)
EstadoInicial,EstadoFinal, Tamanho ← Escolhe probabilisticamente trecho
                                     do Plano a ser otimizado. Devolve
os
                                     estados inicial, final e tamanho.
PlanoAuxiliar ← Tenta achar cegamente um caminho que sai do
EstadoInicial
                e vai ao EstadoFinal depositando o resultado em
                PlanoAuxiliar
Se achou um caminho que vai do EstadoInicial ao EstadoFinal
   então faça
         se Tamanho do PlanoAuxiliar < Tamanho do trecho do Plano
            então substitui Trecho do Plano por PlanoAuxiliar
         fim-faça
   senão faça
         Se PlanoAuxiliar possui estado não inicial que existe em Plano
            e é atingido por menor número de passos em PlanoAuxiliar a
            partir de EstadoInicial do que em Plano
            então substitui trecho do Plano pelo trecho de PlanoAuxiliar
                  que otimizaria o Plano.
         fim-faça
fim-procedimento
```

FIGURA 2.20 – Algoritmo do procedimento OtimizaCegamente

É interessante observar que não existe plano melhor do que o encontrado na figura 2.24 para a otimização do plano 2.21. Esse fato pode surpreender ao considerar que tanto o planejamento como a otimização do planejamento são cegos.

Um agente popperiano implementado não é unicamente composto por um sistema de planejamento e um sistema de indução das regras do universo percebido. Certamente, esses sistemas compõem uma parte essencial do agente; porém, falta descrever como esses sistemas são integrados para compor um agente popperiano completo.

A integração dos sistemas implementados em protótipos de agentes popperianos será descrita no próximo capítulo.

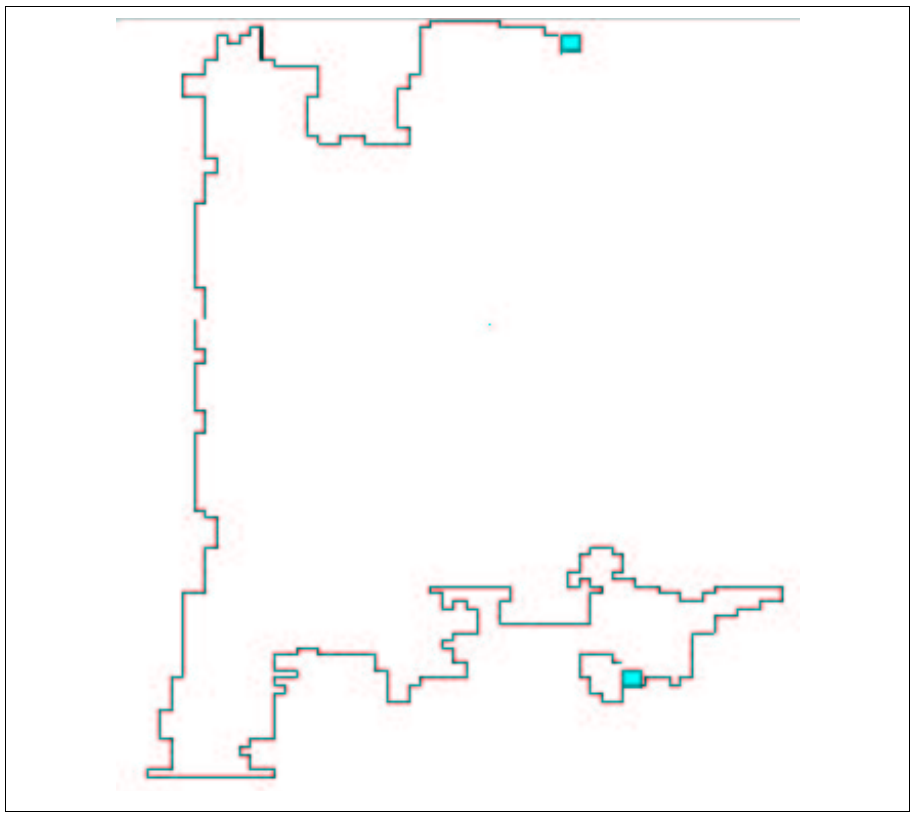

FIGURA 2.21– Plano gerado ao acaso

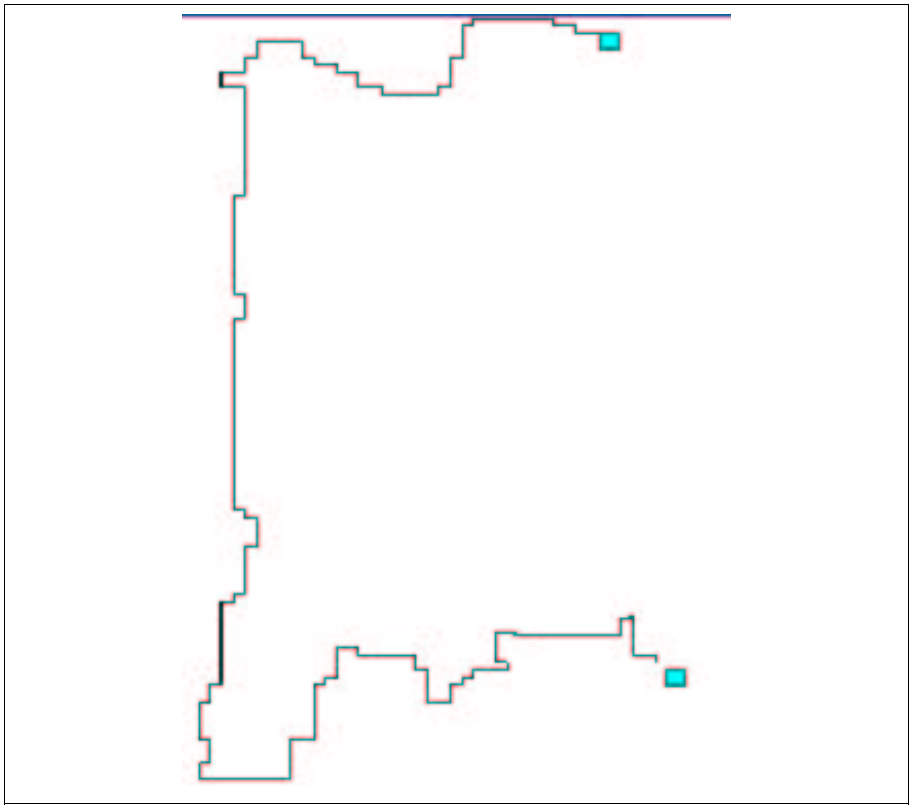

FIGURA 2.22- Plano da figura 2.21 um pouco otimizado

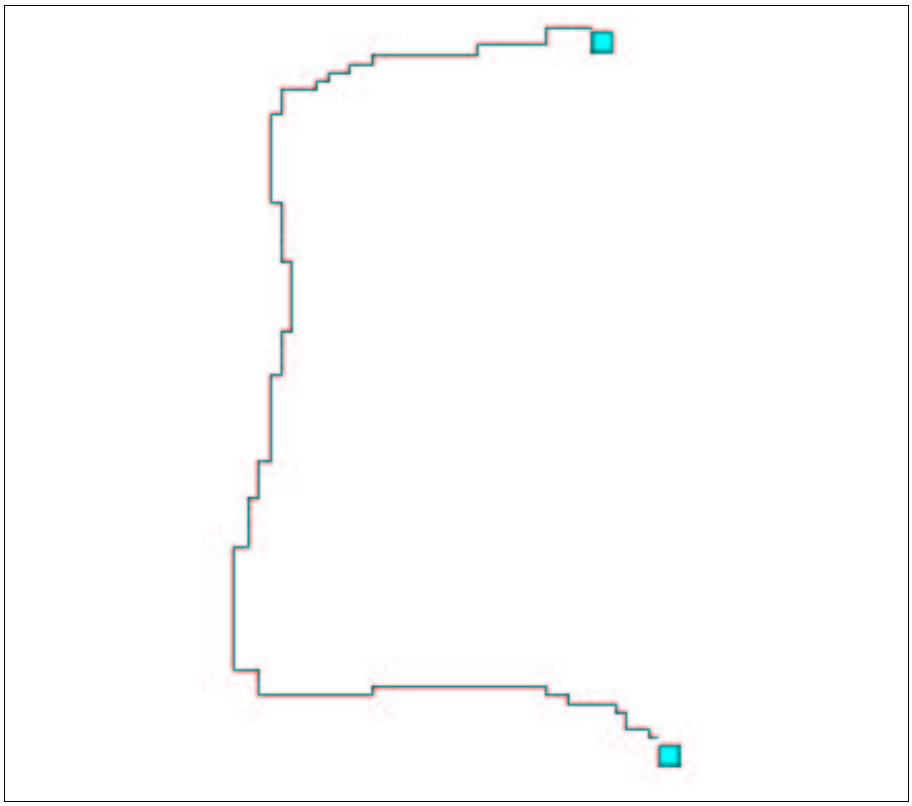

FIGURA 2.23 – Plano da figura 2.22 um pouco mais otimizado

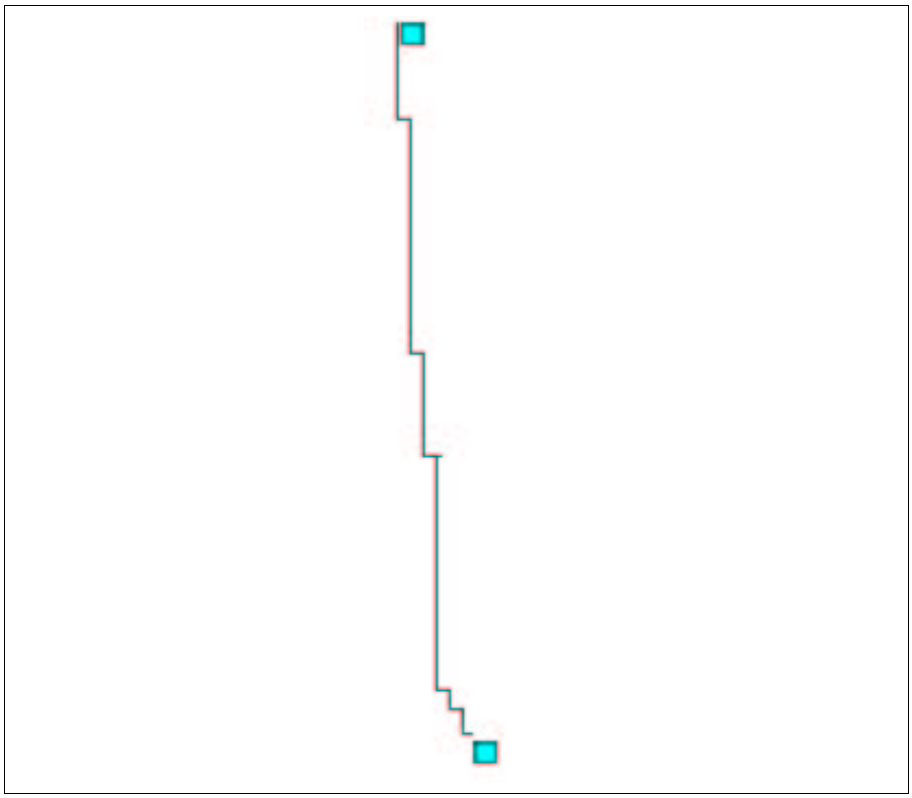

FIGURA 2.24 – Plano da figura 2.23 um pouco mais otimizado

3 Experimentos com Criaturas Popperianas

Nos capítulos anteriores, são estudados assuntos teóricos e práticos importantes para o desenvolvimento de uma criatura popperiana artificial. Aproveitando-se os módulos de indução e planejamento é implementada a criatura popperiana. As duas características mais importantes da criatura popperiana são o aprendizado das regras do universo percebido e o planejamento. A indução de função é responsável pelo aprendizado das regras que compõem o seu mundo enquanto que o planejamento planeja ações ou seqüências de ações que levem a satisfação.

Foram implementados dois protótipos de agentes popperianos:

- Agente minerador popperiano.
- Agente predador popperiano.

O uso de experimentos práticos que produzem resultados numéricos torna-se viável com a implementação de protótipos. Esses resultados permitem comparar o presente trabalho com trabalhos anteriores [COR2000] [DRO92] [HAY95a] [KOR92]. Sendo assim, para cada tipo de agente popperiano implementado, primeiramente é feita a revisão dos resultados existentes e posteriormente é apresentada a solução popperiana.

3.1 Agentes Mineradores Existentes

O primeiro trabalho sobre coleta de amostras foi feito por Steels [STE90] onde um conjunto de robôs coletam amostras de rocha em um ambiente desconhecido. Posteriormente, Drogoul e Ferber [DRO92] fizeram diversos experimentos com conjuntos de robôs reativos sendo que o universo foi definido como segue:

- Ambiente plano.
- Uma base no centro que emite sinal perceptível no raio de 40 metros.
- A velocidade dos robôs é de 1 metro por ciclo.
- Os robôs só podem carregar uma amostra por vez, percebem seu ambiente no raio de 2 metros e percebem o sinal da base no raio de 40 metros.
- Existem 3 pilhas a uma distância de 40 metros da base sendo que cada pilha contêm 100 amostras.

O problema consiste em levar as amostras de rocha (também conhecidas como minérios) para a base. Os dois melhores robôs gerados por Drogoul e Ferber são Petit Poucet 3 (também conhecido como Tomb Thumb Robots 3) e Dockers. As tabelas 3.1 e 3.2 mostram as regras de comportamento desses robôs em arquitetura de subsunção [BRO86][ALV97] apresentadas no trabalho de Cordenonsi [COR2000].

A única diferença de comportamento entre as regras das tabelas 3.1 e 3.2 é a regra 7. A regra 7 resulta no comportamento de fila dos robôs sendo que um robô passa o minério para o outro. 85 robôs do tipo Petit Poucet 3 transportam os 300 minérios para a base em 1075 ciclos enquanto que 93 robôs do tipo Dockers resolvem o problema em 697 ciclos [DRO92].

TABELA 3.1 – Comportamento dos Robôs Petit Poucet 3

Prioridade	***Condição***	***Ação***
.1	Descarregado E Não Percebe Mineral E Não Percebe Pista	Movimento Aleatório
.2	Descarregado E Atinge Mineral	Carrega Mineral
.3	Carregado E Não Atinge Base	Retorna à Base E Deixa Duas Pistas
.4	Carregado E Atinge Base	Descarrega Mineral
.5	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
.6	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral E Remove Pista

TABELA 3.2 – Comportamento dos Robôs Dockers

Prioridade	***Condição***	***Ação***
.1	Descarregado E Não Percebe Mineral E Não Percebe Pista	Movimento Aleatório
.2	Descarregado E Atinge Mineral	Carrega Mineral
.3	Carregado E Não Atinge Base	Retorna à Base E Deixa Duas Pistas
.4	Carregado E Atinge Base	Descarrega Mineral
.5	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
.6	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral E Remove Pista
.7	Descarregado E Atinge Robô Carregado	Captura Mineral do Robô

Posteriormente, Cordenonsi [COR2000] desenvolveu no ambiente de simulação de agentes SIMULA um algoritmo evolutivo para gerar regras de comportamento para robôs mineradores. O resultado encontrado foi o grupo S. As regras de comportamento do grupo S são apresentadas na tabela 3.3.

O grupo S consegue depositar na base os 300 minérios com 94 robôs em 540

ciclos apresentando desempenho superior aos demais robôs. As regras de comportamento do grupo S geradas por evolução artificial expressam o comportamento de um agente darwiniano.

TABELA 3.3 – Comportamento dos Robôs Dockers

Prioridade	***Condição***	***Ação***
.1	Descarregado E Não Percebe Mineral E Não Percebe Pista E Não Percebe Agente	Movimento Aleatório
.2	Descarregado E Não Percebe Mineral E Não Percebe Pista E Percebe Agente	Segue Agente
.3	Descarregado E Não Percebe Mineral E Não Percebe Pista E Atinge Agente	Foge de Agente
.4	Descarregado E Atinge Mineral	Carrega Mineral
.5	Carregado E Não Atinge Base	Retorna à Base E Deixa Pista
.6	Carregado E Atinge Base	Descarrega Mineral
.7	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
.8	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral

Uma medição interessante feita por Cordenonsi foi a medição da energia despendida pelos agentes do grupo S para coletar todos os minérios. A energia é o produto do número de agentes pelo número de cíclos:

energia = n. agentes * n. cíclos

A menor energia encontrada para coletar todos os minérios usando agentes do grupo S foi 34.020 com 27 agentes e 1260 ciclos. O gráfico da figura 3.1 mostra energia consumida em função do número de agentes.

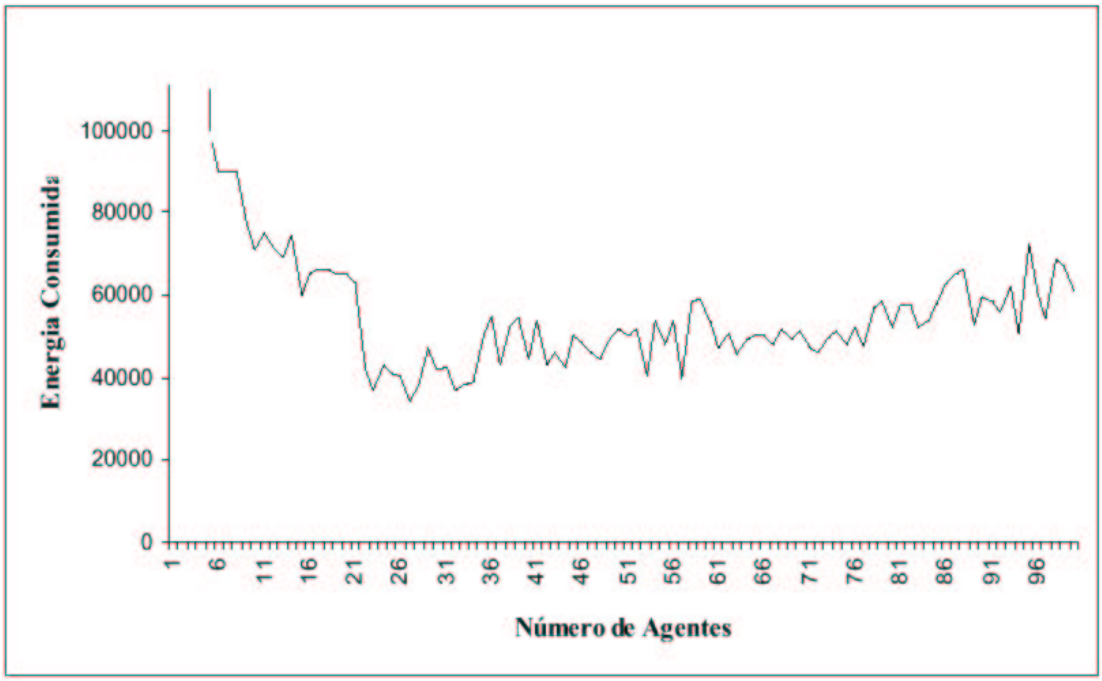

FIGURA 3.1 – Energia Consumida x Número de Robôs do Grupo S

3.2 Algoritmo do Agente Minerador Popperiano

O ambiente usado nos experimentos é o que segue:

- Ambiente plano e discreto de tamanho 100x100.
- Uma base no centro.
- Só existe um robô minerador chamado de agente popperiano minerador.
- Existem 3 pilhas a uma distância de 40 passos da base sendo que cada pilha contêm 100 amostras.

O agente minerador popperiano é definido como segue:

- Pode mover-se exclusivamente para frente, para trás e para os lados.
- Pode carregar um minério proveniente da mina e pode descarregar um minério na base.
- A velocidade máxima do robô é de uma unidade por ciclo.
- Pode transportar uma amostra de minério por vez.
- Percebe sua posição absoluta (x,y).
- Percebe a existência da mina ou da base no raio de uma unidade.
- Percebe o minério carregado.
- Percebe a própria ação tomada.
- Sente satisfação ao executar ou planejar cargas e descargas de amostras de minério.

Existem algumas diferenças entre a maneira como o problema é definido na bibliografia [DRO92][COR2000] e a maneira como ele é definido aqui. No presente trabalho, a base não emite sinal, existe somente um único robô minerador, esse robô percebe a base ou uma mina no raio de uma unidade e a sua posição absoluta.

Considerando que o sistema de indução de função lógica e o sistema de planejamento trabalham com vetores de bytes, a percepção do agente foi definida como um vetor de 5 bytes: última ação, posição X, posição Y, "*estou carregado (0,1)?*", "*base na vizinhança (0,1)?*" e "*mina na vizinhança (0,1)?*". Por exemplo, o vetor (1,30,31,0,0,1) significa que o agente percebe que a sua última ação foi a ação correspondente ao número 1, sua posição absoluta é 30,31 e percebe uma mina. Nesse exemplo, o agente não está carregado e não está próximo da base.

Para efeito do presente trabalho, a função que escolhe a ação que o agente popperiano vai tomar é a função que define o seu comportamento. Sendo assim, a função de escolha de ação chamada de ***EscolheAção*** é a parte central do agente popperiano. É a partir dessa função que os sistemas de indução e planejamento são chamados. O algoritmo da função ***EscolheAção*** do agente popperiano é apresentado no anexo 3.

Para facilitar o entendimento, aborda-se o seguinte exemplo: um robô minerador está explorando o seu mundo através de ações tomadas ao acaso. Esse robô guarda as suas últimas ações e estados passados em uma memória de curto prazo. Ao acaso, ele carrega um minério de uma mina próxima e percebe satisfação com essa ação. Para que no futuro ele possa encontrar a mina e se carregar com mais facilidade do que achar a satisfação ao acaso, ele guarda as suas ações passadas como um plano de como alcançar a satisfação de se carregar. É importante que o agente guarde as suas ações passadas que o levaram à satisfação para que ele repita essas mesmas ações e alcance satisfação posteriormente.

Abordando-se outro exemplo, um robô minerador está explorando o seu mundo através de ações tomadas ao acaso e guardando em memória as suas últimas ações e estados passados. Ao acaso, ele encontra um estado pelo qual já passou, armazenado como parte de um plano que o levou para a satisfação. Concatenando os estados e ações passados que o levaram ao estado atual com o plano que o leva do estado atual à satisfação, obtém-se um novo plano que leva a satisfação. Os planos bem formados sempre levam a satisfação.

No primeiro exemplo, o robô encontrou ao acaso a satisfação. No segundo exemplo, o robô encontrou ao acaso um plano que o leva à satisfação. Além dessas duas formas de planejar, foi implementada uma terceira: planejamento cego que procura a satisfação ou um plano que leva a satisfação. Sendo assim, o agente popperiano não apenas gera planos com experiências do passado como também gera planos usando seu sistema de indução e planejamento. Isso significa que o agente popperiano pode construir planos que passem por estados ou transições de estados nunca antes experimentados. Nesse sentido, o agente popperiano é criativo. De forma resumida, o agente popperiano gera novos planos em 3 casos:

- Escolhendo ações ao acaso, encontra a satisfação.
- Escolhendo ações ao acaso, encontra um plano que o leva à satisfação.
- Por planejamento cego, atinge a satisfação ou um plano que leva a satisfação.

Com o passar do tempo, o número de planos que o agente gera pode crescer consideravelmente. A figura 3.7 mostra o planejamento de um agente minerador popperiano. Os planos em vermelho são planos de descarga de minérios enquanto que os planos em preto são planos para carga de minérios. Pode ocorrer a existência de um plano que leve do estado A ao estado B e outro plano que leve do estado B ao estado A. Sendo assim, a criatura correria o perigo de entrar em um laço infinito de execução.

Um laço infinito de execução não é um problema em si. Um laço infinito de execução que propicie satisfação infinita é desejável. O problema é a existência de um laço infinito de execução que não leve à satisfação. Para impedir tal acontecimento, se o agente popperiano agindo exclusivamente de acordo com planos previamente feitos passa por um estado que já passou sem anteriormente atingir satisfação desde o início das ações planejadas, significa que o planejamento gerou um lanço infinito de execução. Sendo assim, para resolver esse problema, quando detectado o laço, apaga-se o último plano usado. É possível que existam políticas melhores de resolver esse problema; porém, ao considerar a extensão do presente trabalho, preferiu-se adotar a solução simples.

Além do erro de planejamento estudado no parágrafo anterior, o agente popperiano deve reconhecer outros tipos de erros de planejamento. Pode ocorrer que uma transição de estados prevista em um determinado plano não ocorra no momento que o agente popperiano põe esse plano em prática. O agente popperiano deve detectar o erro de transição de estados previsto no plano e descartar esse plano se ele não levar à satisfação.

Outro tipo de erro de planejamento é o que segue: o plano não entra em "*looping*", prediz transições de estados verdadeiras; porém, o agente completa o plano até o fim e não chega na satisfação. Partindo-se do princípio de que planos devem levar a satisfação, planos que não levam a satisfação estão incorretos e devem ser eliminados. De forma resumida, o agente popperiano elimina planos incorretos em 3 casos:

- Quando o planejamento provoca um "*looping*".
- Um plano prevê uma transição de estados e na prática ocorre outra transição que não leva à satisfação.
- A execução do plano não leva à satisfação.

A maior parte do algoritmo da função ***EscolheAção*** é destinado a criar, seguir e apagar planos; porém, a primeira coisa que o agente popperiano modelado faz em relação a escolha de uma ação propriamente dita é fazer uma busca em profundidade 1 para ver se existe alguma ação que leva à satisfação. Se existir essa ação, o agente escolhe essa ação.

Se o agente popperiano não está em condições de alcançar a satisfação com uma única ação e nem tem nenhuma ação planejada para o estado onde ele se encontra, então ele pode agir com uma ação escolhida ao acaso.

No algoritmo da função ***EscolheAção***, não se encontra nenhuma referência do tipo "*encontra mina*" ou "*descarrega amostra de minério*" ou qualquer outra indicação de que o algoritmo seja definido para resolver um problema em particular. Ao contrário, o algoritmo foi definido de forma genérica e pode ser aplicado a uma gama diferente de problemas. Cada agente popperiano pode ter as suas próprias características tais como formas de perceber o seu universo, agir e sentir satisfação com situações específicas.

Ainda assim, mesmo com agentes popperianos diferentes, o algoritmo da função ***EscolheAção*** pode ser seguido.

3.3 Implementação do Agente Minerador Popperiano

A implementação do agente foi feita em Delphi 5 e segue as definições apresentadas na secção anterior. Nessa implementação, o ambiente e o agente são objetos sendo que o agente possui internamente dois outros objetos: um de planejamento e outro de indução de função.

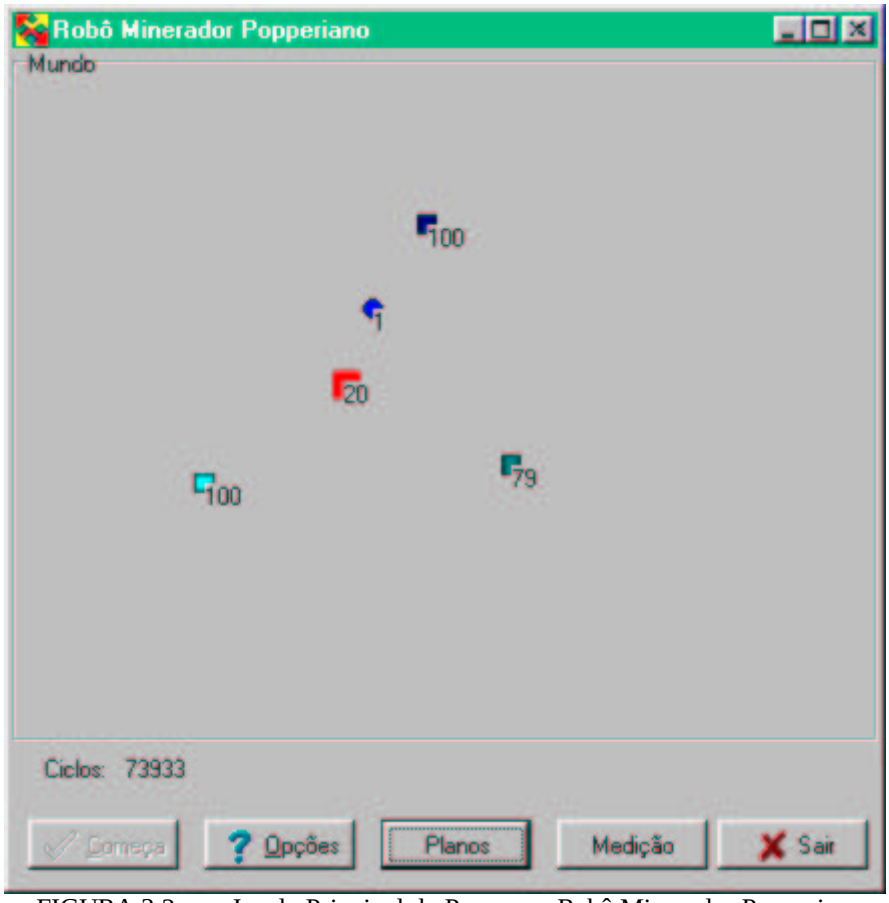

FIGURA 3.2 – Janela Principal do Programa Robô Minerador Popperiano

A janela principal do programa implementado é apresentada na figura 3.2. Nela, são visualizados quatro quadrados e um disco. O quadrado maior em vermelho no centro representa a base enquanto que os quadrados menores representam minas. Cada quadrado carrega um número que representa o número de minérios presentes. O agente minerador popperiano é o disco em azul e seu número é o número de minérios que ele transporta que necessariamente é zero ou um. Ainda na janela principal, são apresentados cinco botões:

- Botão "Começa": define um estado inicial para o mundo e começa a simulação. Uma vez que a simulação foi iniciada, não existe como parar. O agente popperiano desse protótipo não foi projetado para "morrer" ou ser reiniciado.

- Botão “Opções”: abre a janela de opções. Nessa janela, é possível ativar e desativar diversas características do comportamento do agente popperiano. Sua utilidade principal é facilitar os testes.
- Botão “Planos”: abre a janela de planos. A janela de planos apresenta uma representação dos planos para atingir carregamentos e descarregamentos.
- Botão “Medição”: esvazia a base, recarrega as minas e tira o minério do agente caso ele o tenha. A partir desse ponto, mede o número de ciclos que o agente despende para levar todos os 300 minérios para a base.
- Botão “Sair”: sai do programa.

A partir da janela principal do programa, é possível abrir a janela de opções pressionando-se o botão “opções” a qualquer momento. Essa janela pode ser vista na figura 3.3. As opções do agente popperiano podem ser modificadas antes ou durante a simulação. Essas opções são descritas a seguir:

- “Segue Planejamento”: quando desativada, o agente popperiano não age de acordo com seu planejamento e nem gera planos cegamente.
- “Planejamento Cego”: como o próprio nome sugere, quando essa opção está desativada, o agente não gera novos planos por planejamento cego.
- “Elimina Plano Incorreto”: quando desativada, o agente não elimina os planos que se mostrarem incorretos ao longo da simulação.
- “Procura Satisfação em 1 passo”: conforme mostra a listagem da função ***EscolheAção*** no anexo 3, o primeiro teste que pode resultar na escolha de uma ação é o teste da existência de uma ação que atinja a satisfação com uma única ação. Esse teste é modelado na linha 16 do referido algoritmo. Quando a opção “Procura Satisfação em 1 passo” está desativada, o agente não realiza esse comportamento.
- “Mostra Planejamento”: ao contrário das opções anteriores, essa opção regula a interface e não o comportamento do agente. Essa opção ativa ou desativa a atualização da janela “Planos” que mostra os planos que o agente internamente possui.
- “Debug”: essa opção não influencia o comportamento do agente. Quando usada, gera uma saída de texto bastante significativa. Essa opção é útil para verificar a correção dos sistemas de indução e planejamento.
- “Mostra Evolução”: essa opção também não interfere no comportamento do agente. O ambiente que o agente habita foi implementado de forma independente da interface. Sendo assim, a simulação pode ser feita sem a atualização da interface para possibilitar simulações mais econômicas em termos de tempo de UCP. Quando “Mostra Evolução” está desativada, nenhuma atualização é feita na janela principal ou na janela de planos.
- Barra de rolamento “Otimização”: define a freqüência com que o planejamento é otimizado. Por exemplo, 1% de otimização significa que é feita a otimização do planejamento em 1% das vezes que a função ***EscolheAção*** é chamada. As figuras 3.4 a 3.7 mostram o impacto da otimização sobre o planejamento.
- Barra de rolamento “Randomicidade”: quando o agente não tem plano para o

estado onde está e não pode atingir a satisfação com uma única ação, "Randomicidade" define a probabilidade das ações escolhidas ao acaso.

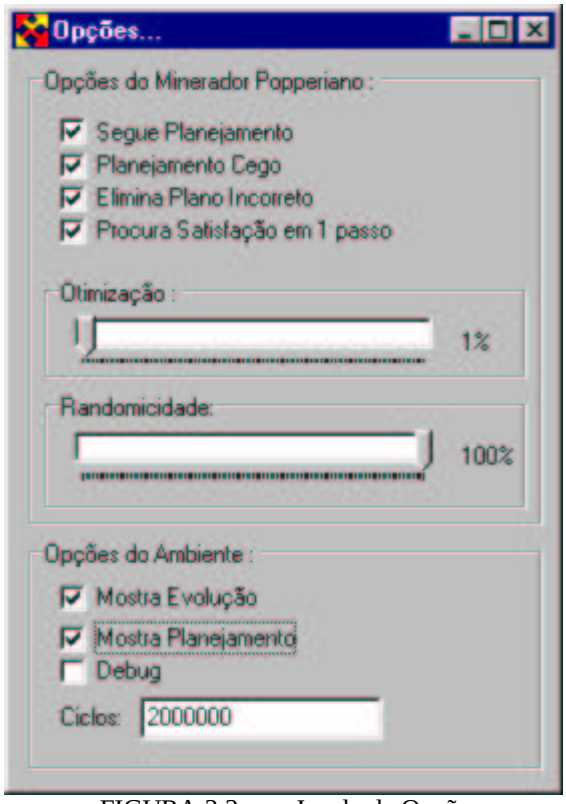

FIGURA 3.3 – Janela de Opções

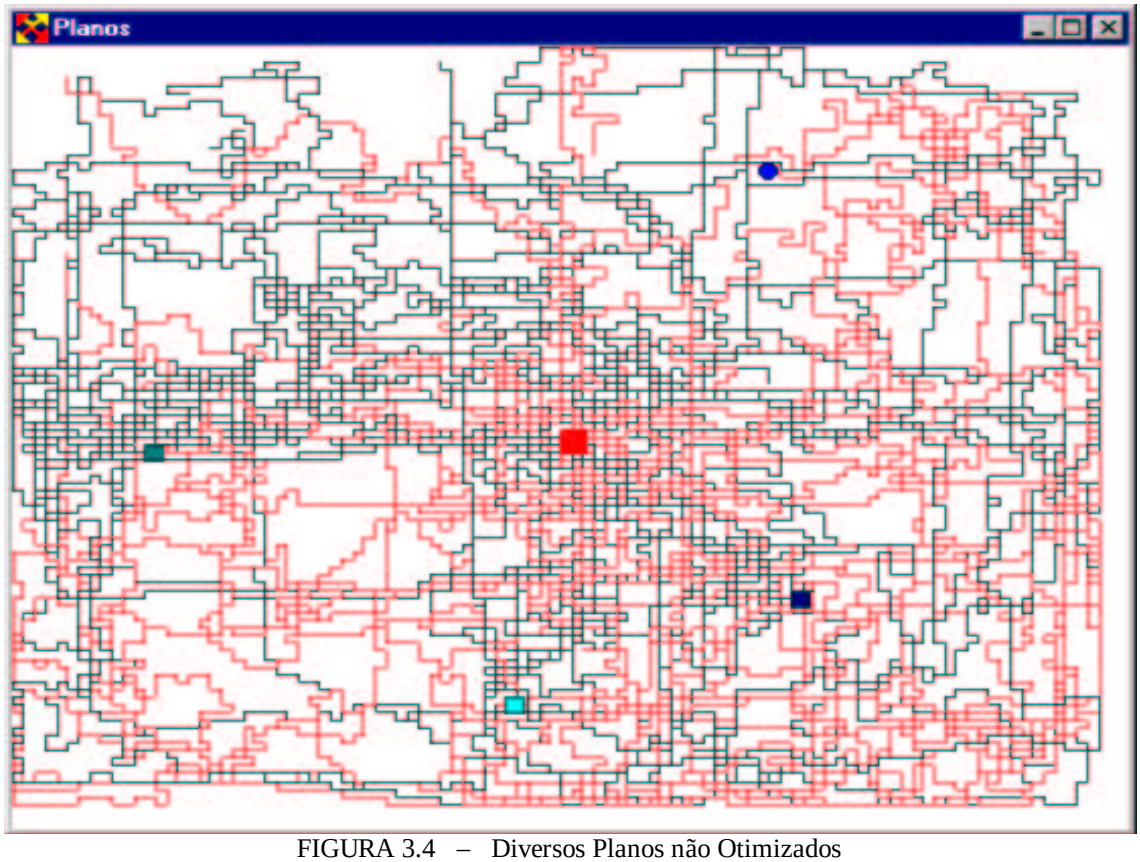

FIGURA 3.4 – Diversos Planos não Otimizados

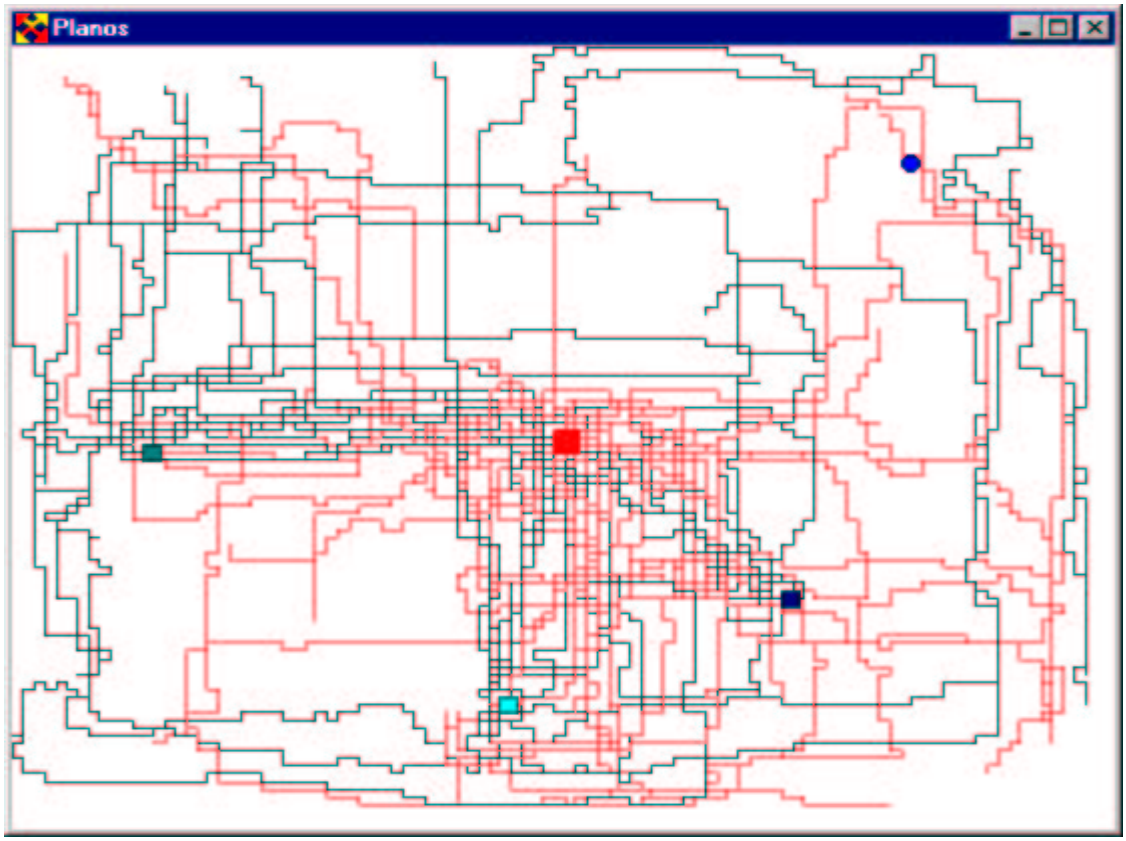

FIGURA 3.5 – Planos da Figura 3.4 um Pouco Otimizados

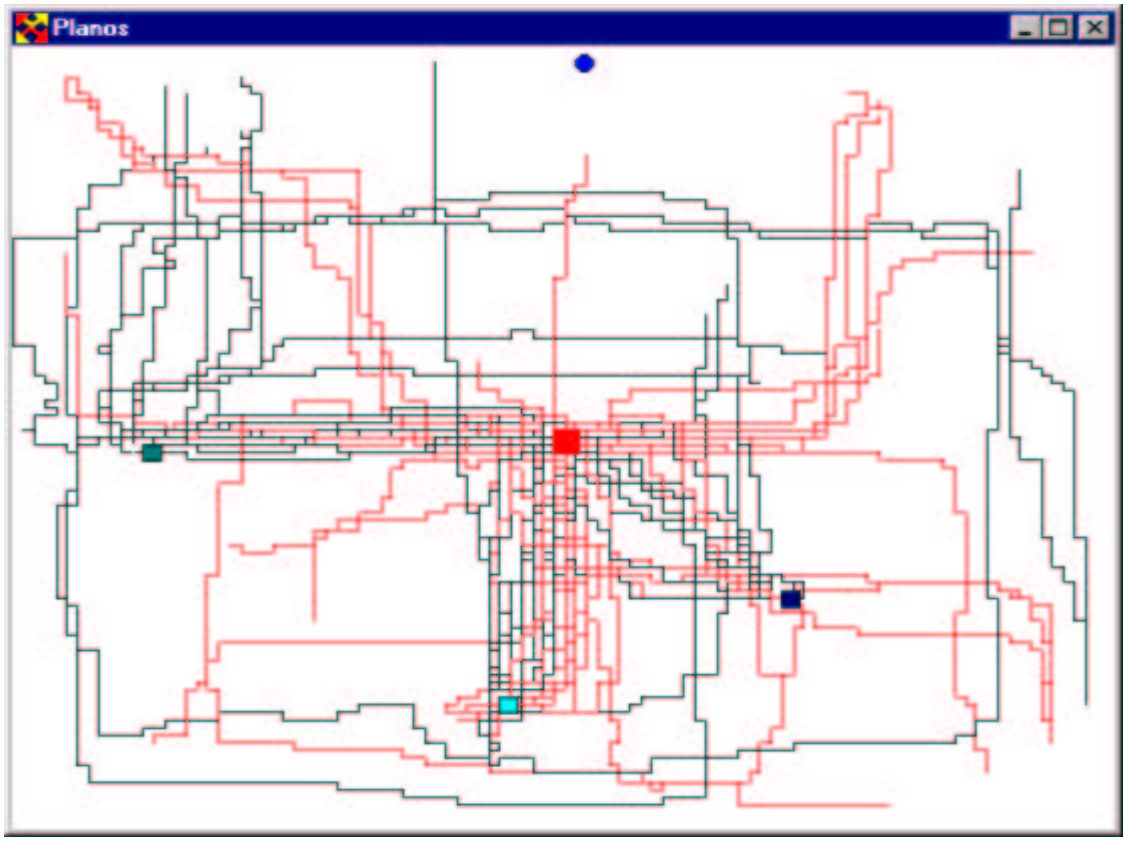

FIGURA 3.6 – Planos da Figura 3.5 um Pouco mais Otimizados

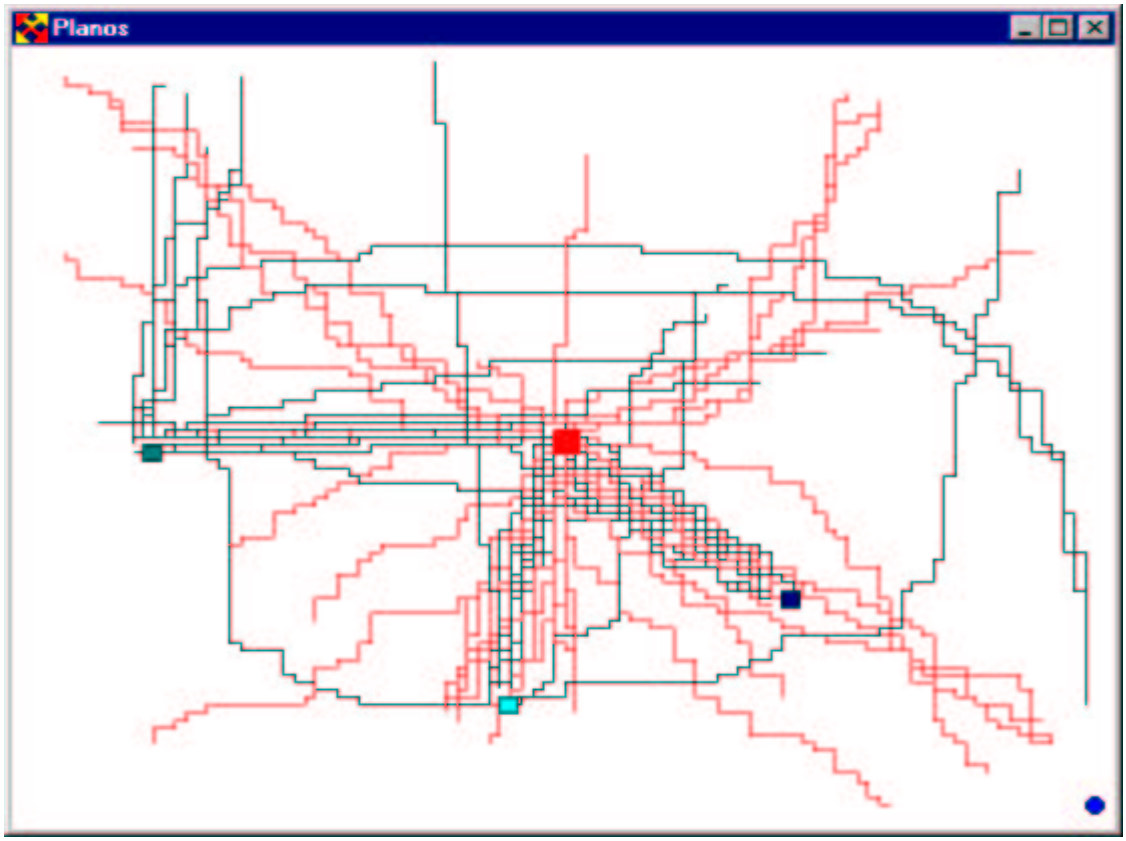

FIGURA 3.7 – Planos da Figura 3.6 um Pouco mais Otimizados

3.4 Resultados Obtidos com o Agente Minerador Popperiano

É uma questão central como comparar os resultados encontrados no presente trabalho com resultados encontrados em trabalhos anteriores. O número de ciclos não é um indicador justo tendo em vista que os testes feitos aqui usam somente um único agente para coletar os 300 minérios enquanto que experimentos anteriores usaram até uma centena de agentes mineradores.

No trabalho de Cordenonsi[COR2000], além da medição do número de ciclos, foi feita a medição da energia que é o produto do número de ciclos pelo número de agentes. Considerando que no presente trabalho só existe um único agente, a energia é igual ao número de ciclos.

O uso da energia despendida para transportar os 300 minérios até a base é um parâmetro de comparação mais justo do que o número de ciclos. Ainda assim, a energia não é uma medida justa. O problema dos agentes mineradores pode ser partido em dois problemas menores: (1) explorar o ambiente para achar as minas e (2) levar os minérios até a base.

No primeiro problema, o agente popperiano tem uma imensa desvantagem: ele tem que aprender as regras do seu ambiente, quando pode se mover, carregar e descarregar enquanto que o agente reativo "nasce" pronto. No segundo problema, o agente popperiano leva uma tremenda vantagem tendo em vista que a posição das minas faz parte do seu ambiente aprendido. A memória e o planejamento sobre o ambiente percebido permitem que o agente popperiano volte aos locais onde ele encontrou satisfação com menor energia.

Para efeitos comparativos, pode ser feita a seguinte pergunta: sabendo-se exatamente a posição das minas e da base, qual é a energia a ser gasta pelo robô que dispõe do melhor algoritmo possível de coleta de minérios? Tendo essa resposta, pode-se comparar os resultados com a menor energia teórica para o transporte dos 300 minérios.

Na forma como o ambiente foi modelado aqui, é impossível para o agente sair de uma posição vizinha à base, ir até a mina, carregar, voltar e descarregar em menos de 74 passos. Sendo assim, para levar os 300 minérios, são necessários 74*300 = 22.200 passos sendo essa a energia mínima.

Pode-se analizar o problema de forma oposta: medir a energia necessária para transportar os 300 minérios até a base por um agente que simplesmente escolhe ações ao acaso com a restrição que ele só pode se carregar na mina e descarregar na base. Esse teste não impede o agente de tentar descarregar longe da base. Ele pode tomar a ação descarregar longe da base; porém, nada vai acontecer. O resultado encontrado experimentalmente é que o agente descarrega na base 1 minério a cada 146 mil passos. Sendo assim, para descarregar os 300 minérios na base, seriam necessários cerca de 44 milhões passos.

Por hipótese, um agente que coleta todos os minerais em 200 milhões de passos apresenta desempenho pior do que o agente que age aleatoriamente significando que sua regra de comportamento atrapalha mais do que ajuda. Todos os resultados encontrados na bibliografia ou aqui são evidentemente bem melhores que o resultado encontrado com o agente minerador aleatório.

Outra característica importante a ser observada é a adaptabilidade do agente minerador popperiano. O agente minerador modelado aqui só pode mover-se

ortogonalmente; porém, para verificar a adaptabilidade do agente, permitiu-se que ele pudesse se mover na diagonal. Sendo assim, sem nenhuma alteração no código fonte do agente ou qualquer parte de seus componentes, espera-se que o agente aprenda as regras do ambiente e com essas regras planeje nesse novo ambiente.

Conforme a figura 3.8, o agente minerador popperiano consegue usar os movimentos na diagonal para o seu próprio benefício. No único teste feito nesse mundo, o agente transportou os 300 minérios em apenas 18.078 passos mostrando que certamente ele usou os movimentos em diagonal para a sua satisfação tendo em vista que usando somente os movimentos ortogonais são necessários no mínimo 22.200 passos.

Voltando ao mundo ortogonal, para calcular quantos passos um agente popperiano que já aprendeu as regras do ambiente e a posição das minas necessita para transportar os 300 minérios até a base, foram feitos testes seguindo-se os seguintes passos:

- Executa-se o protótipo.
- Na janela de opções, deixa-se somente a opção “Procura Satisfação em 1 passo” ativada. Fixa-se a otimização em 1% e a randomicidade em 100%.
- Pressiona-se o botão “Começa”.
- Deixa-se passar 1 milhão de ciclos para que o agente aprenda as regras do seu mundo, aprenda a posição das minas e construa seus primeiros planos. Essa fase é considerada a infância do agente popperiano minerador.
- Terminada a infância, ativa-se todas as opções do agente minerador, eleva-se a otimização para 100% e pressiona-se o botão “Medição”. Com isso, mede-se quantos ciclos o agente despende para carregar os 300 minérios até a base.

Seguindo-se os passos descritos acima 20 vezes, o agente minerador popperiano despendeu em média 23.787 passos para carregar os 300 minérios até a base sendo que o desvio padrão foi de 749 passos. Esse resultado é 7,1% maior do que o mínimo teórico para o problema que é de 22.200 passos. O desvio padrão vale 3,1% da média encontrada o que demostra que o agente minerador popperiano apresenta pouca variabilidade para resolver o problema.

Conforme foi comentado anteriormente, a energia de 23.787 em média não pode ser comparada diretamente com a energia de 34.020 do grupo S [COR2000] tendo em vista que os agentes do grupo S além de transportar os minérios devem explorar o ambiente para descobrir onde os minérios estão.

A análise importante a ser feita é que após um longo período de aprendizagem, o agente popperiano minerador resolveu o problema com apenas 7,1% mais energia do que a melhor forma possível de resolver o problema.

Conforme foi observado anteriormente, quando as regras do ambiente foram alteradas para permitir que o agente minerador pudesse se deslocar na diagonal, o agente minerador popperiano aproveitou essa possibilidade sem que nenhuma modificação fosse feita em seu código fonte mostrando sua capacidade de adaptação a novas condições.

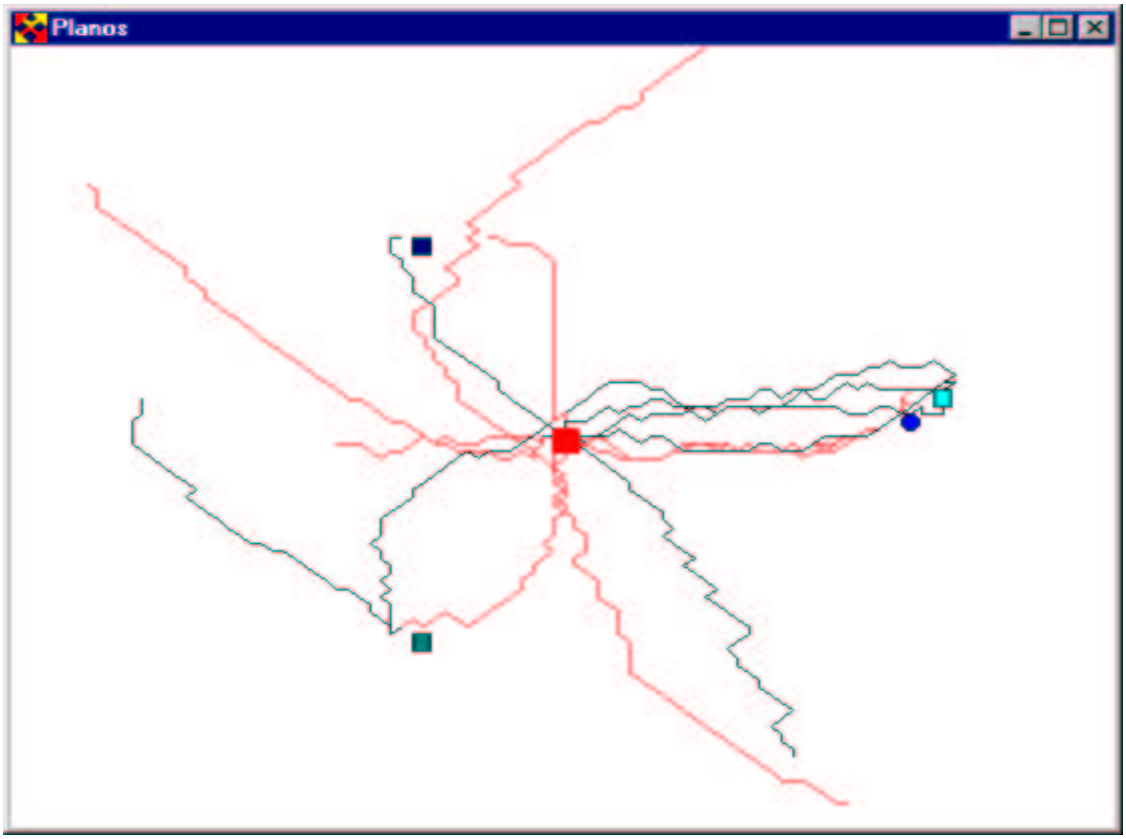

FIGURA 3.8 – Planos com movimentos diagonais

3.5 Predador e Presa

O termo “Predador e Presa” também referenciado como problema da perseguição (do inglês, *Pursuit Problem*) é usado para designar diferentes problemas que normalmente envolvem agentes do tipo predador que perseguem ou comem os agentes do tipo presa. O que freqüentemente muda de um modelo de “Predador e Presa” para o outro é a forma como os agentes agem e percebem o seu ambiente assim como o próprio ambiente.

Os problemas do tipo “Predador e Presa” tem sido usados com sucesso para o estudo de algoritmos evolutivos aplicados à sistemas multiagentes [HAY95b] [NIS97] [NOF98] [DIE95] [WAH 98]. Isso se deve ao fato de que é possível aplicar o algoritmo evolutivo tanto para o predador como para a presa fazendo com que ocorra uma “corrida armamentista” entre predadores e presas.

O modelo de “Predador e Presa” usado aqui foi inicialmente proposto por Miroslav Benda [BEN86] que é formado por um plano discreto e infinito, um agente vermelho chamado de presa e quatro agentes azuis chamados de predadores. Os agentes podem mover-se verticalmente e horizontalmente. Os predadores não podem ocupar a mesma posição da presa e a presa não pode ocupar a mesma posição de um dos predadores. A captura ocorre quando a presa é cercada pelos quatro predadores de forma que ela não possa se mover.

Posteriormente, baseando-se no modelo de Benda, Korf [KOR92] realizou experimentos com 8 predadores e uma presa que podem se movimentar na diagonal, horizontal e vertical. Além disso, Haynes e Sen [HAY95a] [HAY95b] usando

programação genética fortemente tipada (do inglês, *strongly typed genetic programming*) usaram um algoritmo evolutivo para selecionar os melhores algoritmos de predadores e presa gerando a "corrida armamentista". Nesse trabalho, os autores acreditam terem encontrado algoritmos para predadores onde os predadores cooperam entre si para caçar a presa enquanto que a presa aprende a evitar situações onde a cooperação dos predadores pode ter sucesso.

No modelo de Haynes, dois agentes não podem ocupar a mesma posição, o predador pode ver a presa a qualquer distância e a presa vê os predadores a qualquer distância. O ambiente desse modelo é uma matriz 30x30 onde as bordas são aderidas de forma que a posição (0,29) é vizinha da posição (0,0). Existem 5 ações possíveis: mover para os 4 lados e ficar parado. Nesse trabalho, concluiu-se que a melhor estratégia de fuga é simplesmente fugir em linha reta. Essa estratégia de fuga chamada de presa linear (do inglês, *linear prey*) é constituída de 2 passos: (1) escolher ao acaso uma direção e (2) seguir nessa direção para sempre.

3.6 Modelo do Predador Popperiano

O problema Predador e Presa foi modelado aqui de forma muito semelhante à forma encontrada no trabalho de Haynes. As definições dos agentes e do ambientem são as que seguem:

- O ambiente é plano e discreto de tamanho 30x30 com bordas opostas unidas.
- A presa é linear (*linear prey*).
- O predador percebe a sua própria posição relativa à presa e o estado de ocupação das 4 posições vizinhas da presa.
- O predador sente satisfação ao ocupar uma posição vizinha da presa.
- Existem 5 ações disponíveis para os agentes: mover para os lados ou ficar parado.
- Não é permitido que 2 agentes ocupem a mesma posição.
- Um agente não tem capacidade de empurrar outro agente.

O ideal seria que o modelo da função ***EscolheAção*** fosse idêntico tanto para o agente minerador popperiano como para o agente predador popperiano; porém, existe uma grande diferença nos dois problemas: no primeiro, o agente minerador age sozinho em um mundo totalmente determinista enquanto que, no segundo, os agentes predadores batem ou ficam presos por outros predadores que eles não percebem e não podem determinar qual será a ação futura. Sendo assim, do ponto de vista do agente popperiano predador, o mundo é cheio de incertezas.

Evidentemente, a predição do agente popperiano erra mais vezes em um ambiente incerto. Para entender a profundidade do problema, aborda-se o seguinte exemplo: o predador acredita que é impossível ir para a direita; porém, esse é o único caminho que o leva à presa e momentaneamente ele está aberto. Todas as vezes que o predador tentou ir para a direita no passado, ele não conseguiu e acabou desenvolvendo uma crença muito forte de que é impossível ir para a direita; porém, agora, isso é possível e ele não sabe. O predador não conseguiu ir para a direita anteriormente porque

existia outro predador nessa direção que o bloqueava; porém, um predador não percebe o outro e fica impossível saber quando pode-se ir.

Sendo assim, nesse exemplo, deve existir algo na regra de comportamento do predador que o faça ir para a direita mesmo quando ele acredita que é um "absurdo tentar ir para a direita porque a experiência mostra que não se sai do lugar fazendo isso". Para resolver esse problema, deu-se mais prioridade à ação aleatória para que o agente popperiano predador experimente, ainda que raramente, ações que em princípio são consideradas absurdas e que podem eventualmente acabar significando grandes descobertas.

Esse ganho de prioridade na ação aleatória foi conseguido transferindo-se o bloco da função ***EscolheAção*** do agente minerador que vai da linha 68 a 74 para a linha 21 sem nenhuma outra alteração. A função ***EscolheAção*** do predador popperiano é apresentada no Anexo 1.

3.7 Implementação do Predador Popperiano

A implementação do agente predador popperiano foi feita nos mesmos moldes da implementação do agente minerador popperiano. A figura 3.9 mostra a janela principal do protótipo implementado. Nessa figura, apresenta-se uma situação de captura onde os quatro predadores cercam a presa. A presa é o círculo vermelho e os predadores são os quadrados.

Ao comparar as figuras 3.2 e 3.9 ficam evidentes as semelhanças. A única diferença na tela principal é a ausência do botão "Medição". Isso ocorre porque no protótipo do predador e presa a medição é acionada automaticamente sempre que ocorre uma captura.

A janela de opções do agente predador pode ser vista na figura 3.10 apresentando diversas semelhanças com a janela de opções do agente minerador. Nessa janela, por motivo de enfoque do presente trabalho, as caixas de grupo "Tipo de Presa" e "Tipo de Predador" não são documentadas. As opções do predador popperiano e do ambiente já foram descritas no agente minerador popperiano.

A janela "Planos" também foi implementada de forma semelhante a encontrada no protótipo do agente minerador; porém, com duas diferenças principais: a presa fica sempre no centro dessa janela e cada plano desenhado apresenta a cor de seu respectivo agente. Essa janela pode ser vista na figura 3.11.

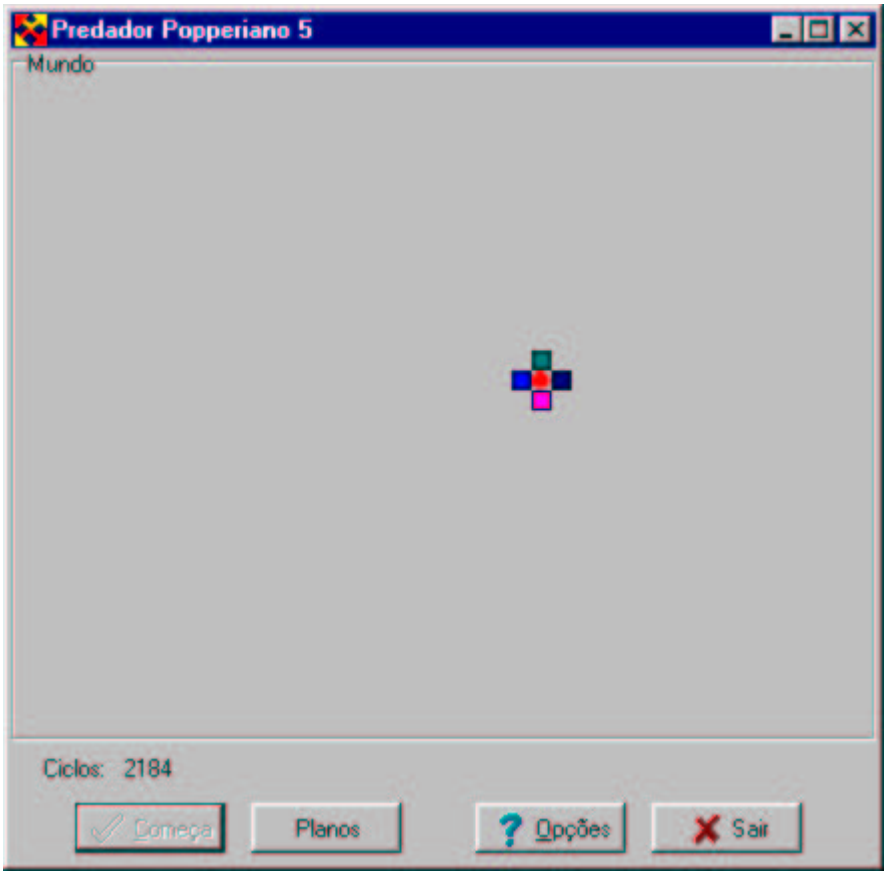

FIGURA 3.9 – Janela Principal do Protótipo

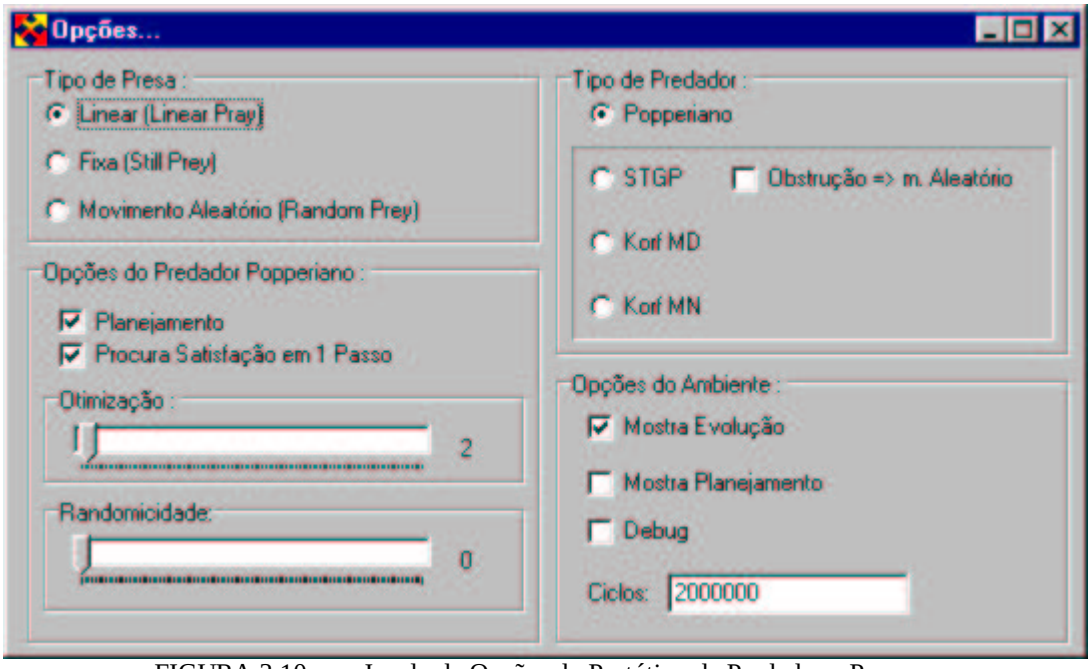

FIGURA 3.10 – Janela de Opções do Protótipo de Predador e Presa

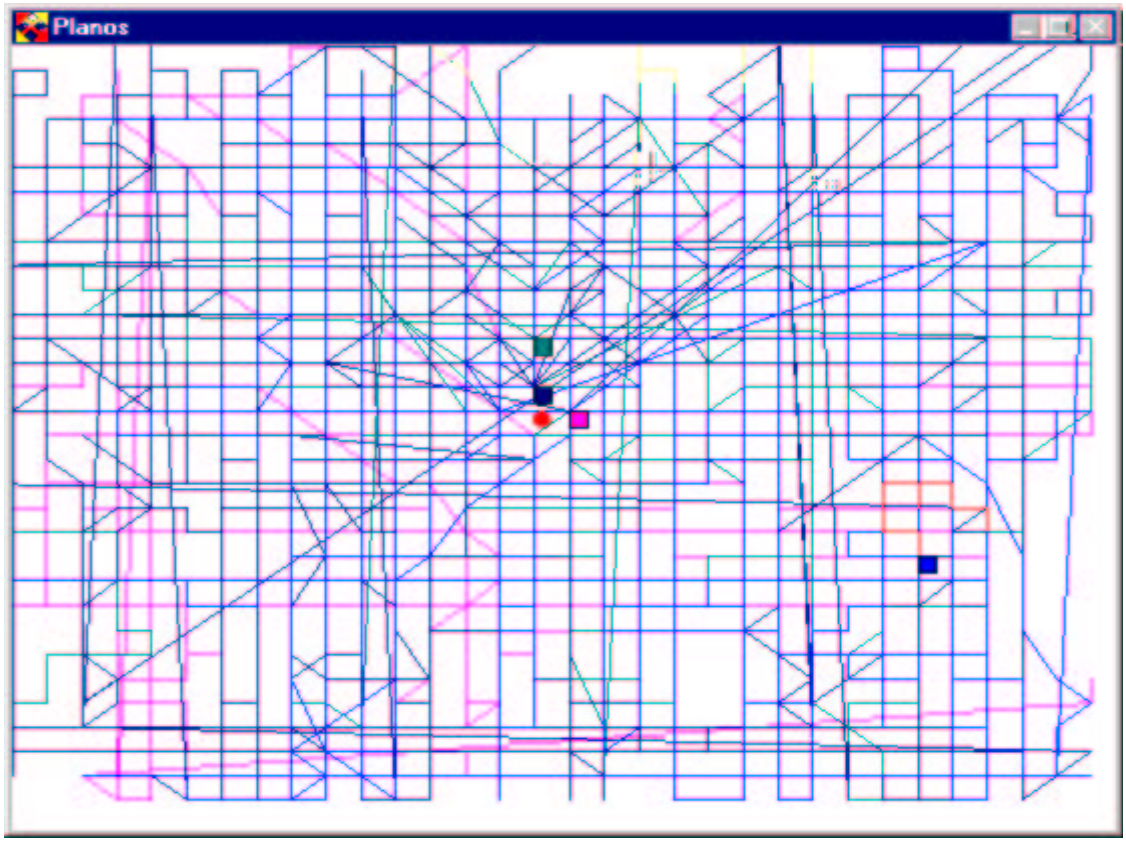

FIGURA 3.11 – Janela de Planos do Protótipo de Predador e Presa

3.8 Resultados Obtidos com o Predador Popperiano

Os mesmos problemas que foram encontrados para comparar o agente minerador popperiano com agentes mineradores existentes são encontrados aqui. O agente popperiano leva desvantagem ao considerar que enquanto os agentes reativos ou darwinianos “nascem prontos”, os agentes popperianos precisam de um longo processo de aprendizado para poder dar seus primeiros passos coerentes. Passada a fase onde o agente popperiano leva desvantagem, a situação inverte-se tendo em vista que o agente popperiano é adaptável, tem memória e planeja o futuro.

O resultado numérico a ser encontrado com o protótipo é o número médio de ciclos que os predadores popperianos despendem para capturar a presa linear. Outra característica interessante a ser investigada é a possibilidade de os agentes acabarem bloqueando-se mutuamente (do inglês, *Dead Lock*). Um exemplo de bloqueio ocorre quando dois agentes lado a lado querem ir em direções opostas e nenhum sai do lugar. Em princípio, agentes popperianos não deveriam bloquear-se mutuamente tendo em vista que eles devem aprender com a experiência e fazer planejamentos usando essas experiências. A experiência de bloqueios eventuais deve ser suficiente para que um agente popperiano tente evitar bloqueios. Para encontrar tais resultados, foram feitos testes seguindo-se os seguintes passos:

- Executa-se o protótipo.
- Na janela de opções, seleciona-se a presa linear, o predador popperiano, deixa-se somente a opção “Procura Satisfação em 1 passo” ativada. Fixa-se a

otimização em 0% e a randomicidade em 100%.

- Pressiona-se o botão “Começa”.
- Inicia-se a fase considerada como a infância do agente. Deixa-se passar 200 mil ciclos para que o agente aprenda as regras do seu mundo e construa seus primeiros planos.
- Terminada a infância, ativa-se todas as opções do agente popperiano, fixa-se a otimização em 1% e randomicidade em 5%. Com isso, automaticamente, a quantidade de ciclos usada em cada captura é apresentada na tela.

As opções de ambiente não alteram os resultados numéricos. Essas opções podem ser modificadas para alterar a velocidade do experimento e a forma de apresentação.

Na fase infantil do agente popperiano predador, a sua regra de comportamento é simplesmente: “se acredita que pode atingir satisfação em 1 passo, então toma essa ação, senão, movimento aleatório”. Mesmo com uma regra de comportamento tão simples, em uma amostra de 31 experimentos, os predadores capturaram a presa despendendo 2473 ciclos em média com um desvio padrão de 1608 ciclos.

Na fase adulta, em uma amostra de 28 experimentos, os predadores popperianos despenderam em média 313 ciclos para capturar a presa com um desvio padrão de 572 ciclos. Ainda que a primeira vista um desvio padrão tão alto sugira um erro de cálculo, ele está correto. Esse desvio padrão é atribuido ao fato de que basta um agente não conseguir atingir a presa para que a captura não ocorra. Ainda que a média tenha sido 313 ciclos, o pior caso registrado na amostra foi de 3.170 ciclos enquanto que o melhor caso foi de 57 ciclos. Em nenhum dos testes foi verificado o bloqueio mútuo.

4 Conclusão

No final da conclusão do livro A Origem das Espécies [DAR87], Darwin especula: *"Entrevejo, num futuro remoto, caminhos abertos a pesquisas muito mais importantes ainda ... isto é, sobre a aquisição necessariamente progressiva de todas as faculdades e de todas as aptidões mentais, o que lançará uma nova luz sobre a origem do homem e sua história"*. Estudando a evolução da mente, Dennett [DEN97] propõe quatro tipos de mentes que evoluíram ao longo do tempo: darwiniana, skinneriana, popperiana e gregoriana.

O presente trabalho propõe um modelo computacional para uma criatura popperiana. Os agentes popperianos implementados no capítulo anterior são rígidos; porém, a definição proposta deve servir como uma ferramenta para facilitar a classificação de agentes. Sendo assim, define-se o agente popperiano:

> *" o agente popperiano percebe estados do ambiente e de si próprio, aprende regras que regem a transição desses estados e usando as regras aprendidas planeja e executa planos com o objetivo de alcançar satisfação"*.

O agente popperiano não precisa perceber completamente os estados do ambiente e de si próprio nem aprender totalmente as regras de transição desses estados. Para um agente poder ser classificado como popperiano, é necessário apenas que ele tenha alguma capacidade de aprender as regras do ambiente e alguma capacidade de planejar nesse ambiente.

A capacidade de adaptação de um agente popperiano é certamente algo marcante; porém, toda essa adaptabilidade tem um custo: agentes popperianos muito jovens com pouca experiência de vida não dispõem de observações suficientes para induzir as regras do seu universo percebido de forma correta e quase certamente produzem planos impossíveis de serem realizados o que leva a situações indesejadas em sua juventude. Ao contrário, um agente reativo ou darwiniano está pronto para atuar em seu ambiente a partir de seu nascimento ou início de execução.

Comparando a definição de agente popperiano com a definição de inteligência de Pei Wang apresentada no tópico *Non-Axiomatic Reasoning System* que é *"trabalho e adaptação com recursos e informação insuficientes"* [WAN93], o aprendizado das regras que regem as transições de estados da criatura popperiana é obviamente um processo adaptativo assim como o planejamento. É interessante ressaltar que ao contrário do NARS, o agente popperiano não precisa ter raciocínio explícito. De fato, nenhum dos agentes popperianos implementados no presente trabalho possui raciocínio explícito.

O comportamento do WAE é resultado da interação de uma coletividade de agentes. Conforme alegam seus criadores, os agentes do WAE estão no meio do caminho entre um neurônio e uma regra lógica. Da mesma forma, o comportamento dos agentes popperianos implementados é regido em parte por uma coletividade de regras que assemelham-se a neurônios conforme mostra a figura 2.6.

No WAE, existe criação, competição e morte ao nível de agente fazendo com que ocorra evolução das unidades funcionais que compõem o WAE. Da mesma forma, regras são criadas, e eliminadas em um processo competitivo dentro do agente popperiano. Em ambos os sistemas, as unidades funcionais competem e evoluem provocando a evolução comportamental do sistema.

Na mesma linha de pensamento, o comportamento do agente de Muñoz é fruto da iteração de uma coletividade de esquemas. O esquema sendo formado por uma trinca (contexto, ação, resultado) assemelha-se com as regras lógicas do agente popperiano que apresentam uma causa (contexto + ação) e um efeito (resultado). Além disso, o sistema de planejamento proposto aqui apresenta unidades formadas por trincas (estado atual, ação, estado futuro) que se assemelham muito ao esquema do agente de Muñoz. Esses esquemas também competem entre si e apenas os melhores sobrevivem.

O agente de Muñoz não tem a capacidade de planejar longas seqüências de ações em um modelo de mundo sendo esta uma proposta de Muñoz para um trabalho futuro. A grande diferença entre o agente de Muñoz e o agente popperiano modelado aqui é que o agente popperiano pode realizar planejamento de longas seqüências de ações em seu modelo de mundo.

Os três sistemas (WAE, Muñoz e popperiano) apresentam a morte de unidades funcionais menos importantes impulsionando o processo evolutivo e economizando recursos computacionais. Por outro lado, esses mesmos sistemas apresentam o esquecimento tendo em vista a perda da unidade funcional contendo informação.

Nenhum dos três sistemas necessita informação prévia do meio tendo em vista a sua capacidade de adaptação. Evidentemente, essa adaptação tem um custo de tempo além de poder ser perigoso o aprendizado pela experiência em um ambiente que ofereça perigo.

O presente trabalho cumpre a proposta de trabalho futuro feita por Munhõz para um agente adaptativo que pudesse planejar baseando-se no conhecimento aprendido com a experiência. Além disso, o presente trabalho aproveita muitos temas abordados em trabalhos anteriores como a evolução da inteligência proposta por Dennett [DEN97], o cálculo da crença usado no NARS e no WAE, a noção de rede de nerônios competitiva [SIM2001], a seleção dos indivíduos mais aptos como um processo de otimização, uso de aprendizado de Hebb que atende o modelo de Moser [MOS2001] e, de forma mais distante, a idéia de Minsky em que a mente é uma sociedade de atores ou processos [MIN87].

Existe uma convergência teórica entre o WAE, o agente de Muñoz e o agente popperiano aqui proposto: o comportamento é ditado pela coletividade de unidades funcionais que nascem competem e morrem. Além disso, experimentos práticos nos problemas de predador e presa [BEN86] e mineração [DRO92] atestam a capacidade de adaptação da criatura popperiana aqui modelada.

O presente trabalho pode servir como base para a criação de novos agentes popperianos que possivelmente usem estratégias de aprendizado e planejamento distintas. A seção Predador Popperiano apresentou um protótipo de sistema multiagente onde cada agente é um agente popperiano. Sistemas multiagentes popperianos podem ser alvo de estudo.

Outra utilidade desse trabalho é servir como base para um audacioso projeto de modelo e implementação de um agente gregoriano. Nesse caso, o próximo passo a ser dado é modelar e implementar a capacidade de projetar, construir e usar ferramentas sejam elas reais ou abstratas. A projeto de ferramentas pode partir de um sistema de planejamento mais robusto que planeja ao mesmo tempo a ferramenta e o seu uso. Com isso, tem-se a base para começar a modelar uma criatura que domine linguagem complexa tendo em vista a linguagem como uma ferramenta abstrata.

Anexo 1 Algoritmo do Sistema de Indução

Estrutura de Memória

O sistema de indução é alimentado por dois vetores de bytes:

- ***Causas***: estado das causas. O elemento do vetor ***causas*** é representado por C_i.
- ***Efeitos***: estado dos efeitos, posterior à ação das causas. O elemento do vetor é representado por E_i.

A **tabela de relação *TR*** é uma tabela de tamanho variável onde cada linha corresponde a uma implicação entre um evento causal ou combinação de eventos causais e um efeito. Além de armazenar a implicação, a tabela de relação apresenta diversos valores numéricos que representam freqüência, confiança, utilidade e endereço de base. Cada entrada da tabela de relação pode ser entendida como uma implicação não determinista que apresenta a seguinte estrutura:

Lista de Testes $\Rightarrow$ Operação [base r_0 m NúmeroDeVitórias ÚltimaVitória]

onde:

- **Teste**: é um teste que testa valores do vetor **Causas** e retorna o valor verdadeiro ou falso. Por exemplo, $(C_1 = 1) \Rightarrow (E_{base} \leftarrow 1)$ significa que quando o valor do vetor ***Causas*** de índice 1 for igual a 1, o valor do vetor ***Efeitos*** de índice ***base*** receberá 1. O sinal "=" representa teste de igualdade enquanto que o sinal "←" representa atribuição. Nesse exemplo, o teste é apenas o termo $(C_1 = 1)$. Existem vários tipos de testes que são descritos na seção "***Testes".***
- **Lista de Testes**: retorna valor verdadeiro quando todos os testes da lista são verdadeiros; caso contrário, retorna falso.
- **Operação**: a operação sempre grava o seu resultado no elemento do vetor ***Efeitos*** de índice ***base***. A operação pode ser uma simples atribuição ou uma operação mais complexa. Existem vários tipos de operações descritas na seção "***Operações"***. A operação pode ser testada retornando verdadeiro se o resultado da ação é encontrado em ***Efeitos***.
- ***Base***: índice do elemento de ***Efeitos*** que recebe o resultado da operação. Pode ser usado para endereçamento relativo nos testes. Exemplo:

  $$(C_{base-1} = C_{base}) \Rightarrow (E_{base} \leftarrow 1)$$

- r_0: é o número de vezes em que a lista de testes é verdadeira e a operação não se observa verdadeira. $\mathbf{r_0 = n - m}$.
- ***m***: é o número de vezes em que a lista de testes é verdadeira e a operação se observa verdadeira. Aqui, o significado de ***m*** é semelhante ao encontrado no NARS e no WAE.
- ***Vitória***: quando a entrada da tabela de relação é usada com sucesso na predição do elemento do vetor ***Efeitos*** de índice ***base***, ocorre a vitória.
- ***Número de Vitórias***: é o número de vitórias observadas. Esse valor é usado para determinar se uma implicação é útil ou não.

- ***Última Vitória***: instante de tempo em que ocorreu a última vitória. Esse valor é usado para determinar se uma implicação ainda é útil ou não.

Os valores r_0 e ***m*** são usados para determinar a veracidade da implicação; porém, não basta definir se uma implicação é verdadeira ou falsa. É importante verificar se determinada implicação é útil ou não. Os valores ***NúmeroDeVitórias*** e ***ÚltimaVitória*** são usados para determinar a utilidade da implicação.

A implicação C_1 ***ou*** $C_2 \Rightarrow E_1$ pode ser quebrada em duas implicações: $C_1 \Rightarrow E_1$, $C_2 \Rightarrow E_1$. Implicações com o operador lógico ***ou*** podem ser quebradas em conjuntos de implicações que apresentam somente o conector ***e***. Para manter simplicidade no sistema de indução, são criadas somente implicações com o conector ***e*** já que o conector ***ou*** pode ser substituido por um conjunto de implicações.

Os vetores de bytes *causas* e *Efeitos* tem tamanhos arbitrários dependendo da aplicação para a qual estão sendo usados. Em um teste feito e descrito da seção Bit que Vai e Vem, chegou-se a usar vetores de *causas* e *Efeitos* com 34 bytes.

Algoritmo

A funcionalidade de principal importância tanto para as criaturas Skinnerianas como para as criaturas Popperianas é a capacidade de prever o futuro com base na percepção do passado. Quando uma criatura percebe qualquer tipo de evento, ela deve chamar o procedimento ***AtFreq*** que atualiza as freqüências da tabela de relação. Quando uma criatura pretende agir ou planejar o futuro ou uma situação, ela deve chamar o procedimento ***Pred*** que é responsável pela predição.

A implementação do sistema de indução apresenta os procedimentos que seguem, entre outros:

- ***AtFreq(causas, efeitos)***: atualiza os valores r_0 e ***m*** da tabela de relação com base nos vetores de causas e efeitos. Os vetores ***causas*** e ***efeitos*** são parâmetros de entrada. O tópico entitulado *"Exemplo Introdutório Simples"* proporciona uma boa idéia do que é feito por esse procedimento. Em tese, as atualizações das entradas da tabela de relação independem uma da outra podendo ser feitas em paralelo sendo essa uma propriedade herdada com a inspiração na rede neural artificial conforme pode ser visto na figura A01. Essa é uma propriedade não aproveitada em uma implementação feita em *hardware* seqüencial.

```
para cada entrada de índice I da TR
     faça
     se TR[I].ListaTestes é verdadeira
        então faça
              se TR[I].Acao é verdadeira
                 então  TR[I].M  ←  TR[I].M  + 1
                 senão  TR[I].R0 ←  TR[I].R0 + 1
              fim-se
     fim-para
```

FIGURA A01 – Algoritmo do procedimento AtFreq

- ***SAtFreq(causas, efeitos)***: esse procedimento executa os seguintes passos: (1) escolhe uma entrada da tabela de relação vazia ou com poucas vitórias, (2) escolhe testes verdadeiros no vetor ***causas*** e relaciona com uma operação verdadeira no vetor ***efeitos,*** aplica essa relação na entrada da tabela de relação

escolhida no passo 1. O procedimento ***SAtFreq*** é responsável pela geração de novas implicações e ao mesmo tempo pelo descarte de implicações menos interessantes (esquecimento). A escolha do passo 1 é feita usando o algoritmo da figura A08. Nesse algoritmo, são escolhidas ao acaso ***TP*** entradas da ***TR*** sendo que o índice da pior entrada é armazenado em ***Ipior***.

No passo 2, são gerados testes que retornam verdadeiro para o estado momentâneo de ***causas*** e operação que retornam verdadeiro sobre o estado momentâneo de ***efeitos***. A variável lógica ***TodaOperação*** define se todos os elementos dos vetores de causas e efeitos podem ser usados para formar novas operações ou apenas aqueles elementos que momentaneamente apresentem valores diferentes de zero.

A extratégia de gerar apenas testes e ações que sejam válidas no momento da geração elimina a hipótese de gerar relações que nunca se verifiquem verdadeiras.

```
IPior ← EscolhePiorRelação(TP)
TR[IPior].BASE ← índice do vetor Efeitos onde ocorreu erro
                 de predição
TR[IPior] ← CriaNovaRelação
```

FIGURA A02 – Algoritmo do Procedimento SAtFreq

- ***Pred(causas, efeitos)***: prediz o vetor ***efeitos*** com base no vetor ***causas*** usando a tabela de relação. ***Causas*** é o parâmetro de entrada enquanto que ***efeitos*** é o parâmetro de saída. O tópico entitulado "*Exemplo de Predição*" proporciona uma boa idéia do que é feito por esse procedimento. De forma geral, a predição constitui dos seguintes passos: (1) escolher as implicações da tabela de relação que possuam causas verdadeiras, (2) para cada efeito envolvido, escolher a implicação de maior força ***f*** e usá-la para predizer esse efeito. Dependendo da aplicação, pode-se escolher a implicação de maior crença ***d.***

```
Atribui 0 para cada elemento dos vetor OCORR
Atribui -1 para cada elemento dos vetor e VI
para cada entrada de índice I da TR
     faça
     se TR[I].ListaTestes é verdadeira
        então faça
              N ← TR[I].M + TR[I].R0
              se N > 0
                 então F ← M/N
                 senão F ← 0
              IndiceEfeito ← TR[I].BASE
              Efeito ← resultado da TR[I].Ação
              se F > OCORR[IndiceEfeito]
                 então faça
                       Efeitos[IndiceEfeito] ← Efeito
                       OCORR[IndiceEfeito] ← F
                       VI[IndiceEfeito] ← I
                       fim-se
              fim-se
     fim-para
```

FIGURA A03 – Algoritmo do procedimento Pred

Na figura A03, os vetores ***VI***, ***OCORR*** e ***efeitos*** tem o mesmo tamanho. Cada elemento do vetor ***OCORR*** armazena o valor da maior força encontrada para o efeito de índice ***IndiceEfeito.*** O vetor ***VI*** armazena os índices das relações usadas na predição. O elemento ***VI[i]*** armazena o índice da relação usada para prever ***efeitos[i]*** com força ***OCORR[i]***.

- ***DefineNumCausasEfeitos(PNC,PNE)***: define o número de causas e o número de efeitos com os quais o sistema deve trabalhar. Internamente, dimensiona vetores e aloca memória. É um procedimento normalmente chamado na inicialização da criatura.
- ***AtualizaVitórias***: conforme pode ser observado na figura A04, as entradas da TR que foram usadas com sucesso na predição recebem incremento no valor ***NúmeroDeVitórias*** e tem o valor ***ÚltimaVitória*** atualizado para o ciclo atual. Ao contrário, as entradas da TR que foram usadas provocando erros de predição recebem decremento no valor ***NúmeroDeVitórias***.

```
Ciclo ← Ciclo + 1
para cada elemento I do vetor EfeitosEncontrados
     faça
     IVict ← VI[I]
     se IVict é válido
        faça
        se EfeitoPredito[I] = EfeitoEncontrado[I]
           então faça
                TR[IVict].Vitorias ← TR[IVict].Vitorias+1
                 TR[IVict].UltimaVitoria ← Ciclo
                 fim-faça
           senão TR[IVict].Vitorias ←TR[IVict].Vitorias-1
        fim-se
     fim-faça
```

FIGURA A04 – Algoritmo do Procedimento ***AtualizaVitorias***

```
1. Causas ← Percepção()
2. ErrosPredicao ← 0
3. enquanto CriaturaVive faça
4.          Pred(Causas,EfeitosPreditos)
5.          EfeitosPercebidos ← Percepção()
6.          AtFreq(Causas,EfeitosPercebidos)
7.          se EfeitosPercebidos ≠ EfeitosPreditos
8.             ErrosPredicao ← ErrosPredicao +
                      Diferenca(EfeitosPercebidos,
                                EfeitosPreditos)
9.             SAtFreq(Causas,EfeitosPercebidos)
10.            AtFreq(Causas,EfeitosPercebidos)
11.            fim-se
12.         Causas ← EfeitosPercebidos
13.         fim-faça
14.imprime(ErrosPredicao)
```

FIGURA A05 – Algoritmo Genérico de Previsão

Conforme pode ser verificado nos parágrafos anteriores, o algoritmo de indução

lógica usado para predição está distribuído em seus procedimentos. Para efeito do presente texto, prever o futuro é simplesmente prever os eventos a serem percebidos. Para aprender a prever o futuro, as criaturas usam algoritmos baseados no algoritmo da figura A05.

No algoritmo da figura A05, as chamadas ao sistema de indução responsável pela previsão do futuro foram grafadas em negrito. Nesse algoritmo, o processo de aprendizado ocorre durante toda a vida da criatura. A variável ***ErrosPredicao*** acumula o número de erros feitos na predição sendo útil para medir quão bem a criatura consegue prever o futuro. Esse algoritmo prevê somente a próxima percepção; porém, ele poderia ser facilmente modificado para prever percepções posteriores.

Foi implementada a função ***Encontrou(efeitos)*** vista na figura A07 que engloba as linhas 6 à 11 do algoritmo da figura A05. Essa função recebe os efeitos percebidos e retorna o tamanho da diferença entre os efeitos preditos e os efeitos percebidos. Usando essa função, o algoritmo da figura A05 foi convertido no algoritmo da figura A06.

```
1. Causas ← Percepção()
2. ErrosPredicao ← 0
3. enquanto CriaturaVive faça
4.          Pred(Causas,EfeitosPreditos)
5.          Efeitos ← Percepção()
6.          ErrosPredicao ← ErrosPredicao +
                                Encontrou(Efeitos)
7.          Causas ← Efeitos
8.          fim-faça
9. imprime(ErrosPredicao)
```

FIGURA A06 – Algoritmo Genérico de Previsão

Conforme mostra a figura A06, durante o funcionamento, o algoritmo de previsão não precisa chamar outros módulos além de ***Pred*** e ***Encontrou***. O passo 2 do modelo de criatura skinneriana e popperiana apresentado nas figuras 2.1 e 2.2 deve chamar a função ***Encontrou*** enquanto que o passo 3 deve chamar o procedimento ***Pred***.

```
1. AtFreq(Causas,EfeitosPercebidos)
2. AtualizaVitorias
3. se EfeitosPercebidos ≠ EfeitosPreditos
4.    ErrosPredicao ← Diferenca(EfeitosPercebidos,
                                EfeitosPreditos)
5.    SAtFreq(Causas,EfeitosPercebidos)
6.    AtFreq(Causas,EfeitosPercebidos)
7.    fim-se
8. retorna( ErrosPredicao )
```

FIGURA A07 – Algoritmo da Função ***Encontrou(Efeitos)***

```
pior := número muito grande
para I=1 até TP
   faça
   IR := escolhe aleatoriamente um índice da TR.
   Se IR corresponde a uma entrada vazia da TR
      então Atual := -100
      senão Atual := Número de vitórias
   Se Atual<pior
      então faça
            pior := atual;
            Ipior:= IR;
            fim-se;
   fim-para;
```

FIGURA A08 – Algoritmo da Função EscolhePiorRelação(TP)

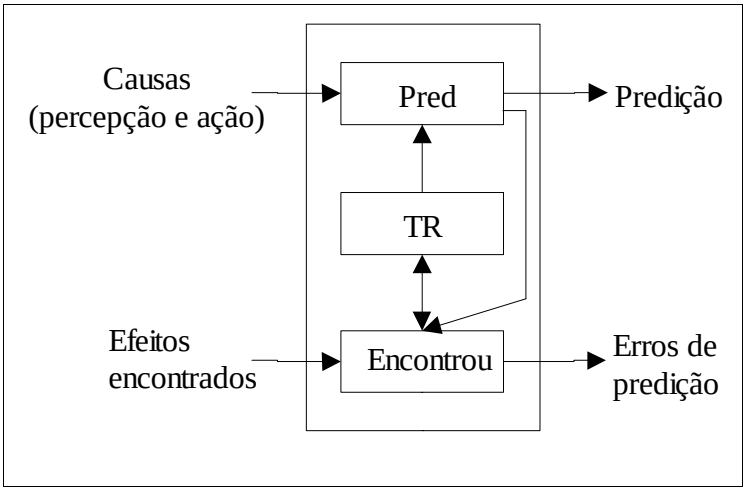

FIGURA A09 – Modelo dos Principais Módulos da Indução de Função

Anexo 2 Listagem do Algoritmo *EscolheAção* do Agente Predador Popperiano

```
função EscolheAção(PercepçãoAtual)
1. Ação ← "não faz nada" // preferencialmente uma ação neutra
2. PercepçãoAtual[0] ← "nenhuma ação"
3. Encontrou(PercepçãoAtual)
4. Sair ← falso
5.
6.  // transforma em plano as ações passadas?
7. se (encontra-se em satisfação) e
8.    (a ação passada foi escolhida por acaso) // por acaso = sem
   planejamento
9.    então faça
10.         gera plano idêntico à lista EstadosNãoPlanejados
11.         Limpa a lista EstadosNãoPlanejados
12.         fim-faça
13.
14.Otimiza todos os planos
15.
16.se (usando predição, prediz que alcança satisfação com uma única ação)
17.    então faça
18.          Ação ← "a ação que leva à satisfação"
19.          Sair ← verdadeiro
20.          fim-faça
21.
22. //Age ao acaso?
23.se não(Sair) e
24.   (número aleatório > constante)
25.   então faça
26.         Ação ← gera Ação ao acaso
27.         Sair ← verdadeiro
28.         fim-faça
29.
30.se não(Sair)
31.   então faça
32.
33.         // erro de planejamento?
34.         se (a ação passada foi planejada) e
35.            (o estado atual é diferente do estado previsto no ciclo
36.             anterior a ser alcançado pelo planejamento) e
37.            (não se encontra satisfação)
38.            então apaga o plano usado na ação passada
39.
40.         se PodeAgir(EstadoAtual,Ação) // existe plano para o estado
   atual?
41.            então faça
42.
43.                  // a variável Ação armazena a ação a ser tomada
44.                  // gera novo plano com as ações passadas?
45.                  se (a ação passada não foi planejada)
46.                     então faça
47.                           gera novo plano com (a concatenação da lista
48.                             EstadosNãoPlanejados com o plano encontrado)
49.                           Limpa a lista EstadosNãoPlanejados
50.                           fim-faça
51.
52.                  // entrou em loop?
53.                  se (Existe o EstadoAtual na lista de EstadosPlanejados)
54.                     então apaga o plano usado na ação passada.
55.
56.                  Sair ← verdadeiro
57.                  fim-faça
58.            senão faça // desenvolve planejamento
59.
```

```
60.                  // erro de planejamento?
61.                  se (não se encontra em satisfação) e
62.                     (a ação passada foi planejada)
63.                     então apaga o último plano usado
64.
65.                  Inclui (EstadoAtual e ÚltimaAção)
66.                         na lista EstadosNãoPlanejados
67.
68.                  se (consegue gerar novo plano) // Planeja Cegamente
69.                     então faça
70.                           Sair ← verdadeiro
71.                           Limpa a lista EstadosNãoPlanejados
72.                           fim-faça
73.                  fim-faça
74.         fim-faça
75.
76.se (a ação atual é planejada)
77.   então Inclui (EstadoAtual e AçãoAtual) na lista EstadosPlanejados
78.   senão faça
79.         Limpa a lista EstadosPlanejados
80.         esquece qual foi o último plano usado para escolha de ação
81.         fim-faça
82.
83.se (está em satisfação)
84.   então Limpa a lista EstadosPlanejados
85.
86.PercepçãoAtual[0] ← Ação
87.Pred(PercepçãoAtual,PercepçãoAtual,PercepçãoEsperada)
88.Retorna (Ação)
89.fim-função
```

Anexo 3 Listagem do Algoritmo *EscolheAção* do Agente Minerador Popperiano

```
função EscolheAção(PercepçãoAtual)
1. Ação ← "não faz nada" // preferencialmente uma ação neutra
2. PercepçãoAtual[0] ← "nenhuma ação"
3. Encontrou(PercepçãoAtual)
4. Sair ← falso
5.
6.  // transforma em plano as ações passadas?
7. se (encontra-se em satisfação) e
8.    (a ação passada foi escolhida por acaso) // por acaso = sem
   planejamento
9.    então faça
10.         gera plano idêntico à lista EstadosNãoPlanejados
11.         Limpa a lista EstadosNãoPlanejados
12.         fim-faça
13.
14.Otimiza todos os planos
15.
16.se (usando predição, prediz que alcança satisfação com uma única ação)
17.    então faça
18.          Ação ← "a ação que leva à satisfação"
19.          Sair ← verdadeiro
20.          fim-faça
21.
22.se não(Sair)
23.   então faça
24.
25.         // erro de planejamento?
26.         se (a ação passada foi planejada) e
27.            (o estado atual é diferente do estado previsto no ciclo
28.             anterior a ser alcançado pelo planejamento) e
29.            (não se encontra satisfação)
30.            então apaga o plano usado na ação passada
31.
32.         se PodeAgir(EstadoAtual,Ação) // existe plano para o estado
   atual?
33.            então faça
34.
35.                  // a variável Ação armazena a ação a ser tomada
36.                  // gera novo plano com as ações passadas?
37.                  se (a ação passada não foi planejada)
38.                     então faça
39.                           gera novo plano com (a concatenação da lista
40.                             EstadosNãoPlanejados com o plano encontrado)
41.                           Limpa a lista EstadosNãoPlanejados
42.                           fim-faça
43.
44.                  // entrou em loop?
45.                  se (Existe o EstadoAtual na lista de EstadosPlanejados)
46.                     então apaga o plano usado na ação passada.
47.
48.                  Sair ← verdadeiro
49.                  fim-faça
50.            senão faça // desenvolve planejamento
51.
52.                  // erro de planejamento?
53.                  se (não se encontra em satisfação) e
54.                     (a ação passada foi planejada)
55.                     então apaga o último plano usado
56.
57.                  Inclui (EstadoAtual e ÚltimaAção)
```

```
58.                          na lista EstadosNãoPlanejados
59.
60.                  se (consegue gerar novo plano) // Planeja Cegamente
61.                     então faça
62.                           Sair ← verdadeiro
63.                           Limpa a lista EstadosNãoPlanejados
64.                           fim-faça
65.                  fim-faça
66.          fim-faça
67.
68. //Age ao acaso?
69.se não(Sair) e
70.   (número aleatório > constante)
71.   então faça
72.          Ação ← gera Ação ao acaso
73.          Sair ← verdadeiro
74.          fim-faça
75.
76.se (a ação atual é planejada)
77.   então Inclui (EstadoAtual e AçãoAtual) na lista EstadosPlanejados
78.   senão faça
79.          Limpa a lista EstadosPlanejados
80.          esquece qual foi o último plano usado para escolha de ação
81.          fim-faça
82.
83.se (está em satisfação)
84.   então Limpa a lista EstadosPlanejados
85.
86.PercepçãoAtual[0] ← Ação
87.Pred(PercepçãoAtual,PercepçãoAtual,PercepçãoEsperada)
88.Retorna (Ação)
89.fim-função
```

Bibliografia

[ALV97] ALVARES, L.O.; SICHMAN, J. Introdução aos Sistemas Multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA; 16.; CONGRESSO DA SBC, 17., 1997, Brasília. **Anais...** Brasília: UnB, 1997. p.1-38.

[BEN86] BENDA, Miroslav; JAGANNATHAN, V.; DODHIAWALA, R. **On Optimal Cooperation of Knowledge Sources**: an empirical investigation. Seattle: Boeing Advanced Technology Center, Boeing Computing **Services**, 1986. (Technical Report BCS G2010-28).

[BEN2001] BEN Goertzel Home Page. 2001. Disponível em: <http://www.goertzel.org/> Acesso em: 8 nov. 2002.

[BRO86] BROOKS, R. A Robust Layered Control System for a Mobile Robot. **IEEE Journal of Robotics and Automation,** New York, v.2, n.1, p.14-23, Mar. 1986.

[COR2000] CORDENONSI, Andre Zanki. **Um Ambiente de Evolução de Comportamentos para Sistemas Multiagentes Reativos**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[DAR87] DARWIN, C. **A Origem das Espécies**. Rio de Janeiro: Tecnoprint, 1987.

[DEN97] DENNETT, Daniel C. **Tipos de Mentes.** Rio de Janeiro: Rocco, 1997.

[DIE95] DIECKMANN, U.; MARROW, P.; LAW, R. Evolutionary Cycling in Predator-Prey Interactions: population dynamics and the red queen. **Journal Theoretical Biology**, [S.l.], n**.** 176, p. 91-102, 1995.

[DOW98] DOWLING, John E. **Creating Mind**. New York: W. W. Norton & Company, 1998. p.9-40, 43-47, 62-80.

[DRO92] DROGOUL, Alexis; FERBER, Jacques. From Tom Thumb to the Dockers: some experiments with foraging robots. In: INTERNATIONAL CONFERENCE ON SIMULATION OF ADAPTATIVE BEHAVIOR ,2. , 1992. **Proceedings...** [S.l.: s.n.], 1992. p. 451-459

[EIG97] EIGEN, Manfred. O que restará da biologia do século XX. In: MURPHY, Michael P.; O'NEILL, Luke A. J. **O que é Vida? 50 Anos Depois: especulações sobre o futuro da biologia**. São Paulo: UNESP, 1997. p. 13-33.

[FEL88] FELTRE, Ricardo. **Química.** São Paulo: Moderna, 1988. v. 3, p. 347-356.

[FEY82] FEYNMAN, Richard P. Simulating Physics with Computers**. International Journal of Theoretical Physics**, New York, v. 21, n. 6-7, p. 467-488, 1982.

[GAR94] GARDNER, Howard. **Estruturas da Mente:** a Teoria das Inteligências Múltiplas**.** Porto Alegre: Médicas Sul, 1994.

[GOE2001] GOERTZEL, Ben. **Digital Intuition**: fragment of a secret book. 2001. Disponível em <http://www.goertzel.org/books/DIExcerpts.htm>. Acesso em: 8 nov. 2002.

[GUY91] GUYTON, Arthur C. **Neurociência Básica**. Rio de Janeiro: Guanabara Koogan, 1991. p.5, 84-88

[HAM87] HAMEROFF, Stuart R. **Ultimate Computing.** Amsterdam: Elsevier Science Publishers, 1987.

[HAY95a] HAYNES, Thomas; SEN, Sandip. Evolving Behavioral Strategies in Predator and Prey. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE WORKSHOP ON ADAPTATION AND LEARNING IN MULTIAGENT SYSTEMS. **Proceedings...** Pittsburgh: Morgan Kaufmann, 1995. p.32-37.

[HAY95b] HAYNES, Thomas et al. Strongly Typed Genetic Programming in Evolving Cooperation Strategies. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS, 6., 1995. **Proceedings...** Pittsburgh: Morgan Kaufmann, 1995.

[HAY98] HAYES, Brian. **The Invention of the Genetic Code.** Disponível em: <http://www.amsci.org/amsci/issues/ Comsci98/compsci9801.html>. Acesso em: 8 nov. 2002.

[JAM84] JAMES, Mike. **Inteligência Artificial em BASIC.** Rio de Janeiro: Campus, 1984. p.10-58.

[KAE96] KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement Learning: a introduction. **Journal of Artificial Intelligence Research,** San Francisco, v.4, p. 237-285,1996.

[KAN97] KANDEL, Eric R.; SCHWARTZ, James H.; JESSEL, Thomas M. **Fundamentos da Neurociência e do Comportamento**. Rio de Janeiro: Guanabara Koogan, 1997. p.5-33, 110-160.

[KOH26] KÖLER, Wolfgang. **Psicologia:** a inteligência dos antropóides. São Paulo: Ática. 1926. p.39-56.

[KOR92] KORF, Richard E. A simple solution to pursuit games. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED ARTIFICIAL INTELLIGENCE, 11. ,1992. **Proceedings...** [S. l.: s. n.], 1992. p.183-194.

[MAC89] MACHADO, R. J. **Handling Knowledge in High Order Neural Networks**: the combinatorial neural model. Rio de Janeiro: IBM Rio Scientific Center, 1989. (Technical Report CCR076).

[MIN87] MINSKY, Marvin. **The Society of Mind**. New York: Simon and Schuster, 1987.

[MOR90] MORAIS, Manuel. **Logos - Enciclopédia Luso-Brasileira de Filosofia.** Lisboa: Verbo, 1990. p.1447-1463.

[MOS2001] MOSER, Lúcio D. **Um Modelo Conexionista de Aprendizagem Baseado na Inteligência Natural.** 2001. 56f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[MUÑ99] MUÑOZ, Mauro E. S. **Proposta de um Modelo de Esquema Cognitivo Sensório-Motor Inspirado na Teoria de Jean Piaget**. 1999. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[NET92] NETTER, Frank H. **The CIBA Collection of Medical Illustrations**. [S. l.]: CIBA-GEIGY Corporation, 1996. v.1, p.154-161.

[NIS97] NISHIMURA, S.; IKEGAMI, T. Emergence of Collective Strategies in Prey-Predator Game Model. **Artificial Life,** Cambridge, MA, v.3, n.4, 1997.

[NOF98] NOLFI, S.; FLOREANO, D. Co-evolving predator and prey robots: Do 'arm races' arise in artificial evolution? **Artificial Life,** Cambridge, MA, v.4, n.4,1998.

[NOR81] NORA, James J.; FRASER, Clarke. **Genética Médica.** Rio de Janeiro: Guanabara Koogan, 1991.

[PEN89] PENROSE, Roger. **A Mente Nova do Rei.** Rio de Janeiro: Campus, 1991.

[PEN 94] PENROSE, Roger. **Shadows of the Mind.** New York: Oxford University Press, 1994. p. 101

[PIA47] PIAGET, J. **O Nascimento da inteligência na criança.** Lisboa: Publicações Dom Quixote, 1986.

[PIN97] PINKER, Steven. **Como a Mente Funciona.** São Paulo: Companhia das Letras, 1997. p.32

[SAG77] SAGAN, Carl. **The Dragons of Eden:** speculations on the evolution of human intelligence. Toronto: The Ballantine Publishing Group, 1977.

[SAG83] SAGAN, Carl. **COSMOS.** Rio de Janeiro: Francisco Alves, 1983. p.23-42.

[SIM2001] SIMON, Haykin. **Redes Neurais:** princípios e prática. Porto Alegre: Bookman, 2001.

[STE95] STERNBERG, Robert. Interview With Robert Sternberg. Disponível em: <http://www.skeptic.com/03.3.fm-sternberg-interview.html>. Acesso em: 8 nov. 2002.

[STE96] STEELS, Luc. **The Origins of Intelligence**. Disponível em: <http://arti.vub.ac.be/~steels/origin/origin.html>. Acesso em: 8 nov. 2002.

[STE90] STEELS Luc, Cooperation Between Distributed Agents through Self-Organisation. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MUITI-AGENT WORLD. **Decentralized AI**. Amsterdam: Elsevier, 1990. p. 175-196.

[SUT98] SUTTON, Richard S. **Reinforcement Learning:** an introduction. Cambridge, MA: Mit Press, 1998. Cap. 1.

[XTA2002] The XTag Project. Disponível em: <http://www.cis.upenn.edu/~xtag/> Acesso em: 8 nov. 2002.

[WAH98] WAHDE, Mattias; NORDAHL, Mats G. Coevolving Pursuit-Evasion Strategies in Open and Confined Regions. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL LIFE, 6.,1998. **Proceedings**... Cambridge, MA: MIT Press, 1998.

[WAN2001] WANG, Pei. **HyPlan file for Wang Pei.** 2002**.** Disponível em <http://www.cogsci.indiana.edu/farg/peiwang/> Acesso em: 8 nov. 2002.

[WAN93] WANG, Pei. Non-axiomatic Reasoning System (version 2.2). In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 8., 1993. **Proceedings...** Menlo Park, CA: AAAI Press, 1993. p. 867-874.

[WAS93] WASLAWICK, R. **Um modelo operatório para construção de conhecimento.** 1993. Tese (Doutorado em Ciência da Computação) – Departamento de Informática e Estatística, UFSC, Florianópolis.