Maurício Archanjo Nunes Coelho

Predição de Dados Estruturados Utilizando a Formulação Perceptron com Aplicação em Planejamento de Caminhos

Juiz de Fora

2010

Maurício Archanjo Nunes Coelho

Predição de Dados Estruturados Utilizando a Formulação Perceptron com Aplicação em Planejamento de Caminhos

Orientador:
Carlos Cristiano H. Borges

Co-orientador:
Raul Fonseca Neto

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Mestrado em Modelagem Computacional

Juiz de Fora

2010

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

O problema de planejamento de caminhos apresenta diversas subáreas, muitas das quais já extensamente abordadas na literatura. Uma dessas áreas em especial é a de determinação de caminhos, os algoritmos empregados para a solução deste problema dependem que os custos estipulados para os ambientes ou mapas sejam confiáveis. A dificuldade está justamente na definição dos custos referentes a cada tipo de área ou terreno nos mapas a serem examinados. Como se pode observar, o problema mencionado inclui a dificuldade em se determinar qual o custo de cada característica relevante presente no mapa, bem como os custos de suas possíveis combinações. A proposta deste trabalho é mostrar como é feita a predição desses custos em novos ambientes tendo como base a predição de dados estruturados definindo um aprendizado funcional entre domínios de entrada e saída, estruturados e arbitrários. O problema de aprendizado em questão é normalmente formulado como um problema de otimização convexa de máxima margem bastante similar a formulação de máquinas de vetores suporte multi-classe. Como técnica de solução realizou-se a implementação do algoritmo MMP (Maximum Margin Planning) (RATLIFF; BAGNELL; ZINKEVICH, 2006). Como contribuição, desenvolveu-se e implementou-se dois algoritmos alternativos, o primeiro denominado *Perceptron Estruturado* e o segundo *Perceptron Estruturado com Margem*, ambos os métodos de relaxação baseados na formulação do *Perceptron*. Os mesmos foram analisados e comparados. Posteriormente temos a exploração dos ambientes por um agente inteligente utilizando técnicas de aprendizado por reforço. Tornando todo o processo, desde a análise do ambiente e descoberta de custos, até sua exploração e planejamento do caminho, um completo processo de aprendizado.

Palavras-chave: Aprendizado de Máquina, Planejamento com Máxima Margem, Perceptron Multi-classe, Planejamento de Caminhos e Predição de Dados Estruturados.

Abstract

The problem of path planning has several sub-areas, many of which are widely discussed in the literature. One of these areas in particular is the determination of paths, the algorithms used to solve this problem depend on the reliability of the estimated costs in the environments and maps. The difficulty is precisely the definition of costs for each type of area or land on the maps to be examined. As you can see, the problem mentioned includes the difficulty in determining what the cost of each relevant characteristic on the map, and the costs of their possible combinations. The purpose of this study is to show how the prediction of these costs is made into new environments based on the prediction of structured data by defining functional learning areas between input and output, structured and arbitrary. The problem of learning in question is usually formulated as a convex optimization problem of maximum margin very similar to the formulation of multi-class support vector machines. A solution technic was performed through implementation of the algorithm MMP (Maximum Margin Planning) (RATLIFF; BAGNELL; ZINKEVICH, 2006). As a contribution, two alternative algorithms were developed and implemented, the first named *Structured Perceptron*, and the second *Structured Perceptron with Margin* both methods of relaxation based formulation of the *Perceptron*. They were analyzed and compared. Posteriorly we have the exploitation of the environment by an intelligent agent using reinforcement learning techniques. This makes the whole process, from the environment analysis and discovery of cost to the exploitation and path planning, a complete learning process.

Keywords: Machine Learning, Maximum Margin Planning, Perceptron Multi-class, Path Planning and Prediction of Structured Data.

1 Introdução

Um problema que abrange diversos conceitos é o de encontrar o caminho mínimo entre dois nós de um grafo ou de uma rede, sendo considerado um grande clássico da Ciência da Computação. Esse problema consiste, normalmente, em encontrar o caminho de menor custo entre dois nós da rede, considerando a soma dos custos associados aos arcos percorridos. Porém, geralmente, a dificuldade maior está em definir os custos de transição entre esses arcos e em se determinar qual é o custo de cada característica relevante presente na transição. Também foi abordado a exploração do ambiente com base no aprendizado através de punições e recompensas.

1.1 Proposta de Trabalho

O objetivo do presente trabalho é possibilitar a predição de custos em novos ambientes ou mapas tendo como base a predição de dados estruturados definindo um aprendizado funcional entre domínios de entrada e saída, estruturados e arbitrários. Aplicado ao problema de planejamento de caminhos este aprendizado torna possível a obtenção de planos ou políticas a partir da percepção das características dos mapas, sendo de grande importância a sua utilização em sistemas de navegação de robôs móveis. Frequentemente, nestes sistemas de navegação, ocorre uma clara distinção entre os níveis de percepção e de planejamento, sendo o planejamento de caminhos obtido somente a partir do prévio conhecimento da matriz de custos relacionada ao espaço de estados-ações do problema. Em resumo, este trabalho se propõe há pesquisar, analisar e formalizar como é feita a predição desses custos e como o agente os explora e aprende o caminho mínimo.

Neste problema, relacionado à predição de custos, tem-se como dados de entrada um conjunto de caminhos escolhidos por um especialista em um ou mais mapas. Estes caminhos são escolhidos de forma a beneficiarem algum tipo de estratégia relacionada à presença ou não de determinadas características, servindo de base ou de exemplo na definição do mapeamento de custos de novos mapas. Desta forma, o mapeamento obtido

possibilitará o planejamento de novos caminhos em novos ambientes de forma similar ao tipo de estratégia escolhida pelo especialista.

O problema de aprendizado em questão é formulado como um problema de otimização convexa de máxima margem estruturado bastante similar a formulação de máquinas de vetores suporte multiclasse (WESTON; WATKINS, 2003).

Como técnica de solução para as predições de custos realizou-se a implementação do algoritmo MMP proposto por (RATLIFF; BAGNELL; ZINKEVICH, 2006) que utiliza uma técnica de sub-gradiente, aplicada à minimização de uma função objetiva irrestrita, relacionada à formulação de máxima margem. Propôs-se e realizou-se também a implementação de dois métodos de relaxação, o primeiro denominado *Perceptron Estruturado* e o segundo *Perceptron Estruturado com Margem*, ambos tem semelhanças com o esquema de correção proposto para o algoritmo *Perceptron Multiclasse* (BAKIR et al., 2006). Posteriormente elaborou-se uma formulação contendo uma margem pré-fixada que pode ser incrementada ou não e seu respectivo algoritmo, obtendo desta maneira o *Perceptron Estruturado com Margem*. Com o intuito de comprovar a eficiência desta abordagem bem como a corretude dos algoritmos implementados, foram realizados diversos testes com diferentes dados de entrada refletidos na escolha de diferentes mapas, caminhos e características.

Depois da descoberta dos custos tem-se o processo de determinação de caminhos. Para isso são analisados e formalizados os fundamentos e métodos necessários para que um agente inteligente possa aprender e determinar caminhos em ambientes variados eficientemente, baseado em suas próprias percepções. Deste modo são apresentados estudos sobre agentes inteligentes, métodos de buscas variados e técnicas de aprendizado. Possibilitando que uma entidade agente se locomova num ambiente complexo. Dessa forma é possível que todo o processo, desde a descoberta dos custos até a exploração do ambiente, seja baseado na descoberta de conhecimento e aprendizado.

Na primeira forma de resolução do problema do caminho mínimo foram utilizadas técnicas de determinação de caminhos, através de um processo de exploração denominado busca *forward*, que toma como ponto de partida o estado inicial ou raiz do problema e vai até o estado final. Complementando a utilização de técnicas de planejamento, é abordada uma outra forma de resolução do problema do caminho mínimo, no sentido *backward*, que seria a forma invertida da solução anterior, ou seja, parte-se do estado final (objetivo) até atingir o estado inicial (raiz) do problema, utilizando um processo de programação recursiva. E também apresentando os principais conceitos do aprendizado por reforço e de todos os elementos necessários para a utilização dessa forma de aprendizado de máquina.

Para o agente explorar esses ambientes tem-se como principal implementação o algoritmo *Q-Learning Incremental*, que é uma variante do *Q-Learning Tabular*, sendo esse último um dos algoritmos mais utilizados na implementação do aprendizado por reforço.

Essa abordagem, desde a descoberta de custos de caminhos até a determinação das caminhos, foi a forma escolhida para ser estudada e explorada neste trabalho, porém as formulações apresentadas bem como suas técnicas de solução podem ser enquadradas com poucas adaptações nos mais variados problemas. Tais como o experimento apresentado em (TASKAR et al., 2005) sobre a predição da conectividade do dissulfeto nas proteínas contendo resíduos de cisteína. Ou então o experimento em (TASKAR; GUESTRIN; KOLLER, 2003) sobre a identificação de letras em documentos manuscritos. Ou o processamento da linguagem natural abordado em (TSOCHANTARIDIS et al., 2005). E para finalizar, a extração de certas imagens e sua análise, tal como o reconhecimento de face, visto em (YANN et al., 2006).

1.2 Organização do Trabalho

O Capítulo 2 apresenta alguns conceitos necessários para o entendimento do trabalho. Tais como otimização combinatória, problema do caminho mínimo e agentes.

No Capítulo 3 tem-se a explicação de diversos tipos de buscas, *Forward* e *Backward*, e aborda-se os conceitos do aprendizado por reforço e o algoritmo *Q-learning*.

Segue-se o aprendizado supervisionado no Capítulo 4, o sub-campo da Inteligência Artificial dedicado ao desenvolvimento de algoritmos e técnicas de aprendizado.

No Capítulo 5, os modelos estruturados podem ser observados e entendidos através de exemplos. São vistos também a modelagem do problema de maximização sob o ponto de vista estruturado.

Segue que no Capítulo 6 são apresentadas as técnicas de solução do problema apresentado no Capítulo 5.

O Capítulo 7 trata especificamente sobre o problema da predição de custos ilustrando o conhecimento aplicado.

Tem-se que no Capítulo 8 os resultados experimentais para o problema específico de predição de custos abordado no Capítulo 7 são apresentados. Tabelas e Imagens são usadas para explicar e ilustrar o experimento.

Finalmente, no Capítulo 9, algumas conclusões sobre o trabalho são apresentadas juntamente com as possibilidades de trabalhos futuros.

2 Noções Introdutórias

O objetivo desta seção é contextualizar esse trabalho dentro da grande área da I.A. abordando seus elementos. Para isso são vistos 5 importantes elementos: grafos, formulação nó-arco, planejamentos de caminhos, otimização combinatória com o problema do caminho mínimo e agentes inteligentes.

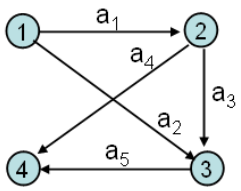
2.1 Grafos

O grafo é uma representação gráfica das relações existentes entre elementos de um conjunto. Ele pode ser descrito num espaço euclidiano de n dimensões como sendo um conjunto V de vértices, também conhecidos como nós, ligados por um conjunto E de curvas contínuas (arestas ou arcos). Dependendo da aplicação, as arestas podem ser direcionadas, e são representadas por setas. Então o grafo é um par ordenado representado por $G = (V, E)$ (CORMEN; RIVEST, 2000). O grafo dentro do contexto de otimização combinatória é comumente chamado de rede.

2.2 Formulação Nó-Arco

Existem diferentes modos de se tratar estruturas e suas relações existentes em um grafo. Agora é apresentada a formulação nó-arco e o porquê da sua escolha para a representação dos caminhos.

Em um grafo $G = (V, E)$ cada um dos seus arcos $a \in E$ e suas direções podem ser representados através de uma matriz $A_{n,m}$ chamada de **matriz de incidência nó-arco**, com uma linha para cada nó e uma coluna para cada aresta. Onde $|V| = n$ e $|E| = m$. Se o arco sai do nó, seu respectivo lugar na matriz de incidência é preenchido com 1, se chega no nó, com -1, e se não há ligação entre o nó e o arco seu valor é 0 (Fig 1).



$$A_{m\times n} = \begin{bmatrix} +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & +1 & +1 & 0 \\ 0 & -1 & -1 & 0 & +1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

Figura 1: Exemplo de um grafo e sua matriz de incidência nó-arco

2.3 Planejamento de Caminhos

"A tarefa de apresentar uma sequência de ações que alcançarão um objetivo é chamada planejamento" (RUSSEL; NORVING, 2004).

O planejamento de caminhos começa com a percepção dos dados relevantes à resolução do problema, em seguida tem-se a análise desses dados e a determinação da função de avaliação com seus custos ou recompensas como base para a escolha de um caminho, e por último a determinação de quais ações devem ser empregadas para que tal caminho possa ser percorrido.

2.4 Otimização Combinatória e Problema do Caminho Mínimo

Problemas de otimização objetivam maximizar ou minimizar uma função definida sobre um domínio. A teoria clássica de otimização trata do caso em que o domínio é infinito. No caso dos problemas de otimização combinatória, o domínio é tipicamente finito. Em geral é fácil listar os seus elementos e testar se um dado elemento pertence a esse domínio. Porém, testar todos os elementos deste domínio na busca pelo melhor mostra-se inviável na prática para a maioria dos problemas (GOLDBARG; LUNA, 2005).

Como exemplos tem-se o problema da mochila, o problema do caixeiro viajante e o problema da satisfabilidade máxima. Eles possuem várias aplicações práticas: projeto de redes de telecomunicação, o empacotamento de objetos em *containers*, a localização de centros distribuidores, análise de dados, na economia (matrizes de entrada/saída), na física (estados de energia mínima), entre outras.

O caminho de custo mínimo é a sequência de ligações que se deve seguir do nó inicial até o nó final num grafo, cujo custo total é mínimo. É um dos problemas mais estudados em Otimização Combinatória. Existem quatro tipos de situações no problema do caminho

mínimo, porém é abordado com ênfase só a primeira:

- caminho mínimo entre origem e destino;
- caminho entre a origem e os demais vértices;
- caminho mínimo entre todos os pares de vértices;
- k-ésimos caminhos mínimos entre um par de vértices: contingenciamento.

Na Figura 2 o caminho mínimo (ótimo) entre a e e é o caminho que passa pelos vértices a, b, c, d, e.

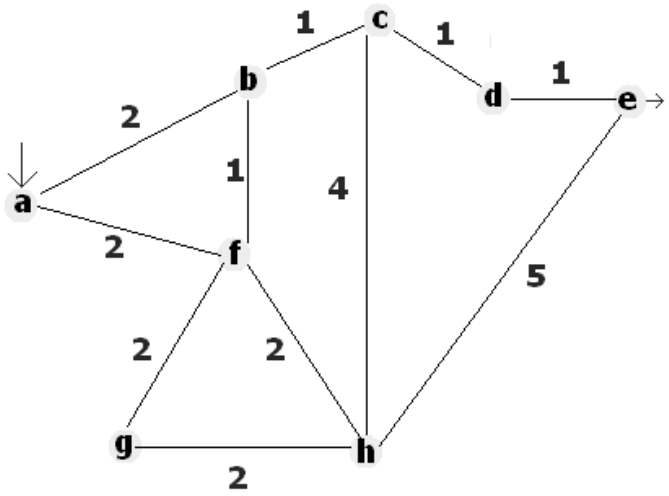


Figura 2: Exemplo de um grafo com custos nas arestas

O problema do caminho mínimo entre dois vértices fixos pode ser modelado como um problema de fluxo compatível a custo mínimo. A quantidade de fluxo que passa no arco (i, j) é representado como $x_{i,j}$. Neste trabalho considera-se a passagem de um fluxo unitário na rede, deste modo o valor da função objetivo retrata o custo do caminho, ou seja, $x_{i,j} \in \{0, 1\}$ então se $x_{i,j} = 0$ o fluxo não passa pelo arco (i, j) e se $x_{i,j} = 1$ o fluxo passa pelo arco.

Não há restrições de capacidade no arco quando o fluxo é unitário, simplesmente passa ou não passa, ou seja, o fluxo é sempre compatível.

A modelagem matemática da forma primal do problema do caminho mínimo se apresenta da seguinte forma. Tomando $X = [x_{i,j}]$, com $i, j = 1, 2, ..., n$, como o vetor de variáveis, representando a quantidade de fluxo que transita no arco (i, j) e $C = [c_{i,j}]$ como o vetor de custos, representando o custo unitário desta transição, pode-se definir o

problema na sua forma primal como:

$$Minimizar \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j}.x_{i,j} \tag{2.1}$$

Sujeito às restrições de balanceamento:

$$\sum_{j=1}^{n} x_{i,j} - \sum_{k=1}^{n} x_{k,i} = \begin{cases} 1 & \text{se } x_i \text{ for origem} \\ -1 & \text{se } x_i \text{ for destino} \\ 0 & \text{caso contrário} \end{cases} \tag{2.2}$$

e acrescido com as restrições de não-negatividade: $x_{i,j} \geq 0$.

Algumas considerações sobre a modelagem matemática: A variável $x_{i,j}$ representa a quantidade de fluxo que deve passar no arco (i, j). Observe que, pelo fato de ser imposta uma restrição de não-negatividade no valor dessas variáveis, os seus valores serão iguais a 0 ou 1. Como a quantidade de fluxo é unitária ($f = 1$) não há necessidade de representar as restrições de capacidade dos arcos no modelo. Se existir um arco de retorno, ligando o objetivo à origem, tem-se um problema de fluxo circulatório que simplifica as restrições de balanceamento, que fica na forma:

$$\sum_{j=1}^{n} x_{i,j} - \sum_{k=1}^{n} x_{k,i} = 0 \tag{2.3}$$

Nesse caso, a matriz de incidência recebe um arco adicional de capacidade infinita, ligando o destino à origem, e a variável $x_{n,1}$ recebe o valor 1. As restrições de balanceamento, também chamadas de equilíbrio, asseguram a conservação dos fluxos em cada vértice. As restrições de integridade (valores inteiros) são garantidas, pela propriedade de uni modularidade da matriz de incidência, ou seja, qualquer submatriz possuirá determinante com valores 0, 1 ou -1, garantindo, dessa forma, a obtenção de soluções inteiras nos sistemas algébricos que fazem a sua utilização.

Na forma compacta, ou seja, na representação matricial, pode-se definir o problema primal através da seguinte formulação:

Min $C.X$

Sujeito à:

$A.X = 0$,

$X \geq 0$

A utilização de uma técnica de programação linear, como exemplo, o método simplex, para solucionar esse problema, não se mostra eficiente na prática. Ao longo do trabalho serão vistas outras estratégias para a determinação do caminho de custo mínimo entre dois vértices. Entretanto, a formulação primal do problema é importante na obtenção de heurísticas, na formulação dual e, consequentemente, na derivação de diversos algoritmos.

Este problema se adapta a diversas situações práticas. Em roteamento, por exemplo, pode-se modelar os vértices do grafo como cruzamentos, os arcos como vias, e os custos associados aos arcos como o tempo de trajeto ou à distância percorrida, e a solução seria o caminho mais curto, ou o caminho mais rápido, entre os vértices. Em redes de computadores, os vértices podem representar equipamentos diversos, os arcos correspondem a trechos de cabeamento, e os custos poderão estar associados às taxas máximas de transmissão de dados. Neste caso, a solução seria a rota de transmissão mais rápida. Além disso, pode-se considerar o problema do caminho mínimo como um subproblema presente na solução de diversos problemas de fluxo em redes, como exemplo, no processo de geração de colunas para a solução de problemas de distribuição de fluxos de diferentes mercadorias, modelado segundo a concepção de formulação de caminhos.

Outras possibilidades de aplicação do problema do caminho mínimo incluem quaisquer problemas envolvendo redes ou grafos em que se tenham grandezas (distâncias, tempos, perdas, ganhos, despesas, etc.) que se acumulam linearmente ao longo do percurso da rede e que alguém deseje minimizar. Neste trabalho o problema do caminho mínimo é aplicado ao problema de determinação de caminhos para um agente móvel, ou robô, em um ambiente caracterizado por um grid de células (ou estados) associados a um conjunto de características, para determinado estado objetivo ou vértice de destino.

2.5 Resolução de Problemas por meio da Busca

A formulação de problemas é um processo para decidir quais ações e estados devem ser considerados. Depois segue-se a busca: um processo de procurar uma sequência de ações e estados que o leve a seu objetivo.

Um algoritmo de busca recebe um problema como entrada, o ambiente do problema é representado por um espaço de estados. O algoritmo retorna uma solução sob uma forma de sequência de ações, com isso tem-se um projeto que consiste em formular, buscar e executar. Um problema pode ser definido formalmente em quatro componentes (RUSSEL; NORVING, 2004):

- O estado inicial em que o agente começa;
- Uma descrição das ações possíveis que estão disponíveis para o agente;
- O teste de objetivo, que determina se um estado é o objetivo;
- E uma função de custo de caminho, que atribui um valor numérico a cada caminho refletindo sua própria medida de desempenho.

Conceitos básicos para se entender um problema de busca: ao invés de se usar a notação $C_{x,y}$ para representar o custo do caminho, pode-se usar também *c(x,a,y)*, e levar em conta qual ação a foi executada para ir do estado x ao y. Uma solução para um problema de busca é um caminho desde o nó inicial até o final, e uma solução ótima tem o menor custo de caminho entre todas a outras. O espaço de estados é dividido em três conjuntos: os já visitados: **conjunto Fechados**, os candidatos: **conjunto Abertos**) e os não-visitados: **conjunto Desconhecidos**. E o custo estimado do caminho mais econômico de um estado x até um estado objetivo y é conhecido como **componente heurística**. São apresentadas agora as definições de alguns termos que medem o desempenho da busca.

2.6 Medição de Desempenho da Busca

Para medir o desempenho durante uma busca te-se quatro fatores a considerar:

- Completeza: O algoritmo encontra uma solução se a mesma existir;
- Otimalidade: A solução retornada é ótima;
- Complexidade de tempo: quanto tempo leva para retornar uma solução;
- Complexidade de espaço: quanta memória é necessária pra efetuar a busca.

A Complexidade depende do fator de ramificação no espaço de estados, representado por b, e pela profundidade da solução, representado por d.

A determinação de um caminho ótimo entre dois nós em uma rede é um problema fundamental que recebeu atenção considerável de várias comunidades de pesquisa nos últimos quarenta anos, pois suas complexidades de tempo e espaço tornam a resolução de certos problemas impraticável. (WINK; NIESSEN; VIERGEVER, 2000).

O capítulo seguinte tem as explicações de dois tipos diferentes de formulações para se determinar um caminho mínimo: as abordagens *forward* e *backward*.

2.7 Agentes Inteligentes

"Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores" (Fig. 3). (RUSSEL; NORVING, 2004)

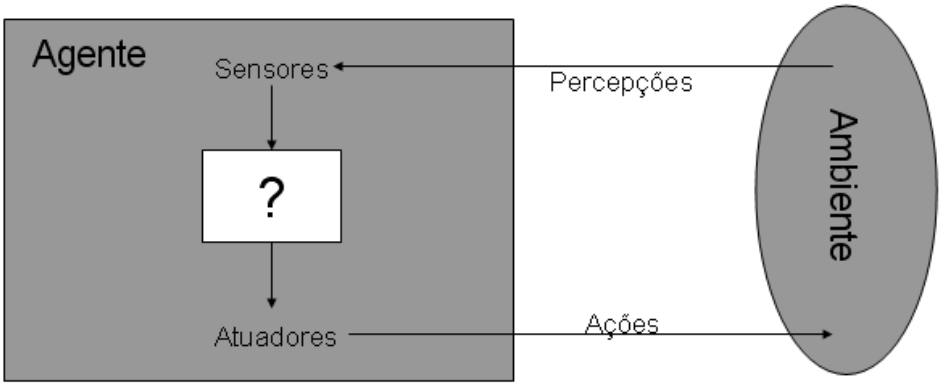


Figura 3: Agentes interagem com ambientes por meio de sensores e atuadores

Como exemplo tem-se um agente simulando um ser humano. Seus sensores seriam os cinco sentidos: visão, audição, paladar, olfato e tato. E seus atuadores suas articulações, principalmente as pernas e os braços atuando no ambiente.

Quando um agente visa maximizar suas chances de sucesso é dito que tal agente possui algum grau de racionalidade. A definição de racionalidade depende de quatro fatores: a medida de desempenho que define o critério de sucesso, o conhecimento anterior que o agente tem do ambiente, as ações que o agente pode executar e a sequência de percepções do agente até o momento.

Tem-se a seguinte definição de um agente racional segundo (RUSSEL; NORVING, 2004): "Para cada sequência de percepções possível, um agente racional deve selecionar uma ação que se espera venha a maximizar sua medida de desempenho, dada a evidência fornecida pela sequência de percepções e por qualquer conhecimento interno do agente."Nos modelos de agentes inteligentes parte-se do princípio de que a inteligência, de alguma forma, já deve estar presente nos elementos autônomos.

"Agência representa o grau de autonomia e autoridade incorporadas ao agente. À medida que o ambiente no qual o agente atua torna-se cada vez

mais imprevisível, inovador, torna-se necessário incorporar mais agência ao agente. Esta incorporação de agência é feita através da construção e atualização de modelos mentais que eventualmente são manipulados internamente pelo agente. Ao receber estímulos do ambiente onde está inserido, ou através de uma decisão interna, o agente atualiza o estado de seus modelos mentais, que é o processo de cognição." (FERNANDES, 2000)

O grau de agência dado ao agente deve ser comparada ao nível de dificuldade da tarefa pretendida. Se for preciso modelar a mente de um agente com capacidades próximas a de uma mente humana, por exemplo, necessita-se criar um modelo muito complexo. Por outro lado, se o agente tem apenas que consultar algumas bases de dados na **Web**, com intenção de pesquisar preços, então o seu modelo mental e suas capacidades cognitivas podem ser bastante rudimentares.

É preciso também definir um meio para medir seu desempenho. O critério para se medir o sucesso do comportamento do agente é conhecido como medida de desempenho. Quando um agente é inserido em um ambiente, ele gera uma sequência de ações, de acordo com as percepções que recebe. Essa sequência de ações faz o ambiente passar por uma sequência de estados. Se ela é desejável, o agente funcionou bem. Porém, não existe uma medida fixa de desempenho que abrange todos os agentes. O projetista é o responsável por criar essas medidas de acordo com o resultado desejado no ambiente.

Existem quatro tipos básicos de programas de agentes: reativos simples, reativos baseados em modelo, baseados em objetivos e baseados na utilidade, são abordados aqui somente os utilizados neste trabalho. Um caso especial são os agentes com capacidade de aprendizagem.

2.7.1 Agentes Baseados em Objetivos

O agente combina informações que descrevem situações desejáveis com informações sobre os resultados de ações possíveis, a fim de escolher ações que alcancem os objetivos. para isso o agente deve considerar longas sequências de ações até encontrar um meio de atingir o objetivo, sendo necessário algum tipo de busca. Um típico exemplo é o do agente utilizando o algoritmo A* para achar o melhor caminho em determinado ambiente.

2.7.2 Agentes Baseados na Utilidade

Os objetivos permitem apenas um distinção entre estado esperado e estado qualquer. No Agente Baseado em utilidades existe uma função de utilidade que mapeia um estado em um número real e descreve o grau de recompensa associado . Essa abordagem é muito usada em problemas de aprendizado por reforço.

2.7.3 Agentes com Aprendizagem

O aprendizado permite ao agente operar em ambientes inicialmente desconhecidos e se tornar mais competente do que se tivesse apenas seu conhecimento inicial. Qualquer agente, inclusive os já citados, pode tirar vantagem da aprendizagem. Existe uma grande variedade de métodos de aprendizado.

O aprendizado em agentes inteligentes pode ser resumido como um processo de modificação de cada componente do agente, a fim de tornar mais preciso as informações de realimentação disponíveis, melhorando assim o desempenho global do agente (Fig. 4).

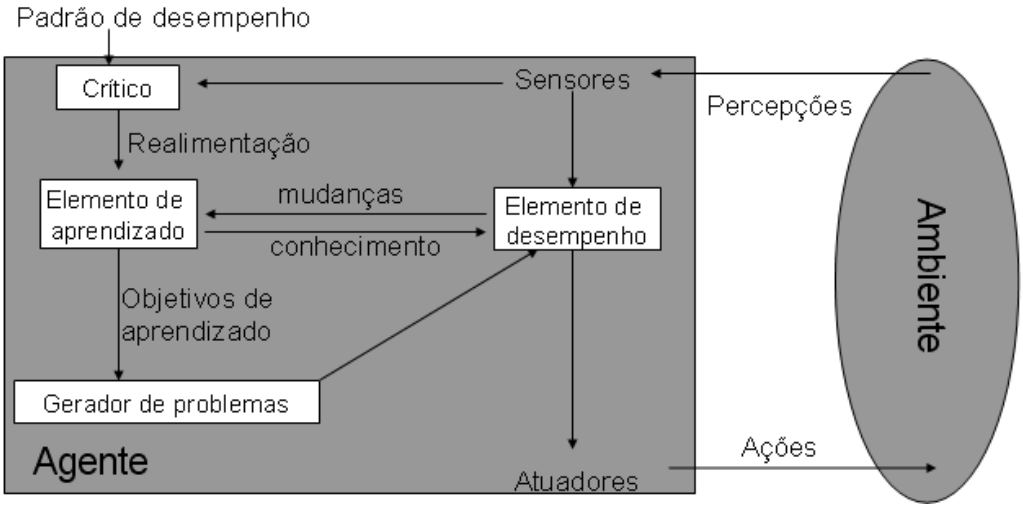


Figura 4: Um modelo geral de agentes com aprendizagem

3 Determinação de Caminhos

São discutidos agora os dois tipos gerais de determinação de caminhos, a busca *forward* e a busca *backward*, cada um com seus respectivos algoritmos.

3.1 Busca *Forward*

O problema de determinação de caminhos pode ser resolvido através de um processo de busca *forward*, ou, de expansão no espaço de estados do problema. A busca *forward*, ou direta, é aquela que determina um processo de exploração que caminha do estado inicial ou raiz do problema em direção aos estados finais que representam a solução do mesmo. Apesar deste processo caracterizar, normalmente, uma forma de planejamento, onde o conhecimento a priori do espaço de estados e das possíveis transições e respectivos custos sejam necessários, é possível a sua adaptação para a solução de problemas *online*, relacionados à determinação de caminhos em tempo real e em ambientes dinâmicos. Para isso, é importante descrever os algoritmos de busca.

Inseridos na busca *Forward* existem dois grandes conjuntos de tipos de buscas: as buscas sem informações, que não são abordadas por serem demasiadamente simplistas, e as buscas com informação, abordadas por possuírem melhor desempenho.

3.1.1 Algoritmo de Dijkstra

Dentro da área de otimização combinatória, o método mais usado para encontrar o caminho mínimo entre uma origem pré-fixada e os demais nós do grafo, quando não há arcos de custo negativo, é o **Algoritmo de Dijkstra**. Ou seja, o algoritmo de Dijkstra identifica, a partir de um nó do grafo, qual é o custo mínimo entre esse nó e todos os outros nós do grafo. A cada iteração m o algoritmo determina o caminho mínimo de um nó origem i até um nó k qualquer. Esse algoritmo segue o princípio de uma busca ordenada, portanto os custos dos caminhos tem valores crescentes, por esse motivo que não pode

haver arcos com custo negativo. Porém isso não chega a ser um grande problema, pois os custos dos arcos são geralmente grandezas físicas mensuráveis. O algoritmo de Dijkstra é de ordem máxima $O(n^2)$, ou seja, de complexidade quadrática.

O algoritmo de Dijkstra considera um grafo $G = (V, E)$, onde os nós pertencentes a V são divididos em três conjuntos: os já visitados(conjunto fechados), os candidatos(conjunto abertos) e os não-visitados(conjunto desconhecidos).

Seja $D^m_{i,k}$ a soma dos custos dos arcos para de i se chagar a k passando por um caminho qualquer, e m a m-ésima iteração. Tem-se que $D^m_{i,k} = Min\{D^{m-1}_{i,p}, D^{m-1}_{i,p} + C_{p,k}\}$. Onde p é um nó fechado na última iteração. O nó cujo $D^m_{i,k}$ foi calculado é colocado no conjunto fechados e seus arcos apontam para os nós que serão incluídos no conjunto abertos. O algoritmo de Dijkstra foi desenvolvido para resolver problemas em rede genéricas.

Os passos principais do algoritmo de Dijkstra, a cada iteração, envolvendo a escolha de um vértice para fechar e a atualização do vetor de distâncias, podem ser implementados por um algoritmo de busca ordenada conhecido na literatura de Inteligência Artificial como algoritmo *Best-First*.

3.1.2 Busca A*

Para tornar mais eficiente a determinação do caminho mínimo entre a origem e um vértice de destino fixo é possível adicionar uma componente heurística. Sua utilização produz um novo algoritmo de busca ordenada: o A*.

Dessa forma, o algoritmo A* se apresenta como a solução mais apropriada ao problema de Busca de Caminhos, pois encontra o caminho de menor custo de um vértice a outro examinando apenas os vizinhos mais promissores do vértice atual da busca. Segundo (PATEL, 2007) a busca A* é a melhor escolha na maioria dos casos.

O algoritmo A* é otimamente eficiente para qualquer função heurística dada, ou seja, nenhum outro algoritmo ótimo tem a garantia de expandir um número de nós menor que ele usando a mesma heurística. Os nós são avaliados de acordo com a equação:

$$f(n) = g(n) + h(n) \tag{3.1}$$

Onde *g(n)* corresponde ao custo exato do caminho desde o nó inicial até o nó n e *h(n)* o custo estimado do caminho de menor custo para ir do nó n até o objetivo. Então pode-se afirmar que *f(n)* é o custo estimado da solução de custo mais baixo passando por

n (RUSSEL; NORVING, 2004).

Como em todo processo de busca ordenada tem-se o nó com menor valor $f(n)$ para expandir, (Fig. 5), sendo este valor armazenado numa estrutura de nós já pesquisados: a lista de nós fechados; e seus filhos numa estrutura de nós a pesquisar: a lista de nós abertos.

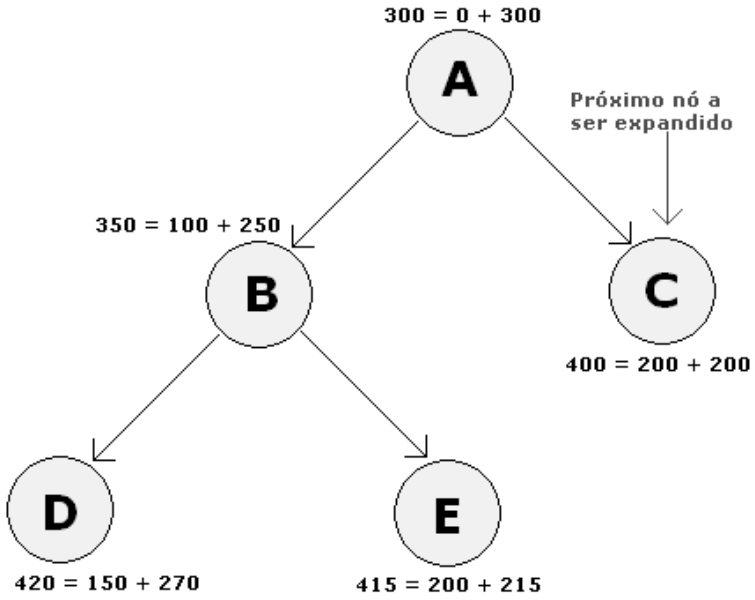


Figura 5: Esquema de uma expansão de nó em uma busca A*

Para preservar a otimalidade do A*, a heurística deve ser admissível. A admissibilidade de uma heurística pode ser comprovada pela propriedade da consistência, sendo a mesma suficiente para a admissibilidade, porém não necessária.

A consistência se baseia no seguinte fato: para um nó n e os seus sucessores n' gerados por uma ação a, o custo estimado de atingir o objetivo a partir de n não é maior que o custo de chegar a n' somado ao custo estimado de n' para o objetivo $h(n) \leq c(n, a, n') + h(n')$ (Fig. 6).

Mesmo com todas as vantagens da busca A*, o crescimento exponencial ocorrerá, a menos que o erro na função heurística não cresça com maior rapidez que o logaritmo do custo do caminho real. A condição para crescimento sub exponencial em notação matemática é:

$$|h(n) - h^*(n)| \leq O(log\ h^*(n)) \tag{3.2}$$

Porém, a maioria das heurísticas consistentes, não atende a condição para crescimento subexponencial. Por essa razão, com frequência é impraticável insistir em uma solução ótima (RUSSEL; NORVING, 2004). É possível usar variantes da busca A* que encontrem

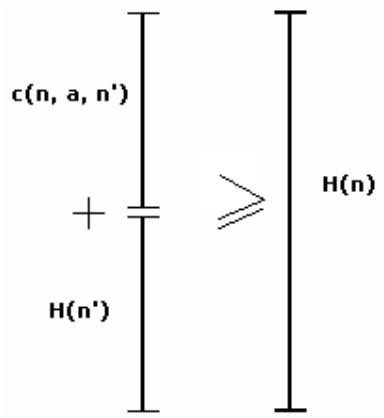


Figura 6: Condição para uma heurística consistente

rapidamente soluções não-ótimas, ou projetar heurísticas mais precisas, embora não estritamente admissíveis.

3.2 Solução *Backward*

A abordagem adotada pelo algoritmo para a solução do problema é do tipo inversa. O algoritmo inicia a partir do vértice objetivo, explorando todo o fecho transitivo inverso do mesmo, até que o vértice de origem seja alcançado.

Inicialmente, tem-se a formulação dual do problema do caminho mínimo, que apresentará uma relação direta com o equacionamento de Bellman e a sua solução por um processo de programação recursiva, estabelecendo uma relação de recorrência característica das técnicas de programação dinâmica e de aprendizado por reforço. A teoria da programação dinâmica e sua relação com o aprendizado por reforço pode ser encontrada no Apêndice A.

3.2.1 Conversão Primal-Dual

O problema do caminho mínimo pode ser analisado sob uma ótica diferente através de sua formulação dual obtida da formulação primal, apresentada na seção anterior. A conversão primal para dual apresenta os seguintes aspectos:

A cada restrição do problema primal relacionado à equação de equilíbrio de um vértice associa-se uma variável dual, denominada μ_i, indexada pelo respectivo vértice. Tem-se então um vetor de variáveis duais com n componentes. Como será visto posteriormente, cada variável dual representará o valor de um caminho do vértice de origem até o vértice

associado.

Como o problema primal do caminho mínimo é de minimização, o problema dual será um problema de maximização, sendo o vetor de custos correspondente ao vetor de demanda do problema primal. Convém lembrar que este vetor possui tamanho n.

Como visto no capítulo 2, seção 2.4, equações (2.1) e (2.2), as restrições do problema primal são de igualdade,, fazendo com que o vetor de variáveis duais, conhecido também como vetor de multiplicadores, tenha valores irrestritos. O vetor que representa o lado direito do sistema de restrições do problema dual será correspondente ao vetor de custos do problema primal, tendo portanto tamanho m.

O problema dual apresenta um sistema de m restrições na forma de inequações, considerando que as variáveis associadas do problema primal são limitadas inferiormente.

Considerando o produto da matriz de incidência $A_{n,m}$ pelo vetor de variáveis duais μ pelo lado esquerdo, pode-se descrever a formulação dual do problema de caminho mínimo entre dois vértices fixos x_1 e x_n na forma matricial:

Maximizar $\mu.[1, 0, 0, ..., 0, 0, -1]^T$

Sujeito à:

$\mu.A \leq c$

μ irrestrito e $\mu_1 = 0$

Na forma algébrica, pode-se reescrever a formulação dual como:

Maximizar $\mu_1 - \mu_n$

Sujeito à:

$\mu_k - \mu_j \leq c_{k,j}$, para todo arco (k, j) do grafo.

μ_i irrestrito, para $i = 1, ..., n$

$\mu_1 = 0$

Aplicando a mesma transformação para outros vértices de destino, permanecendo o vértice de origem x_1 fixo, tem-se a seguinte formulação do problema dual para o problema de caminho mínimo entre uma origem fixa e os demais vértices do grafo:

Maximizar $(\mu_1 - \mu_n) + (\mu_1 - \mu_{n-1}) + ... + (\mu_1 - \mu_2)$

Sujeito à:

$\mu_k - \mu_j \leq c_{k,j}$, para todo arco (k, j) do grafo.

μ_i irrestrito, para $i = 1, ..., n$

$\mu_1 = 0$

Invertendo o sinal de μ_i, ou seja, fazendo $\mu_i = -\mu_i$, e, tomando $\mu_1 = 0$, tem-se a seguinte formulação final para o problema dual:

Maximizar $\mu_n + \mu_{n-1} + ... + \mu_2$

Sujeito à:

$\mu_j - \mu_k \leq c_{k,j}$, para todo arco (k, j) do grafo.

μ_i irrestrito, para $i = 1, ..., n$

$\mu_1 = 0$

Se tanto o primal quanto o dual admitem soluções factíveis, então ambos têm soluções ótimas iguais.

3.2.2 Solução do Problema Dual

O algoritmo proposto para a solução do problema dual é baseado em uma técnica que mantém sempre uma solução primal viável equivalente a uma árvore geradora, e procura a cada iteração satisfazer as restrições do problema dual segundo uma técnica de relaxação. Dessa forma, quando for alcançada uma solução que seja ao mesmo tempo primal viável e dual viável fica garantido a obtenção de uma solução ótima.

3.2.3 Equações de *Bellman*

Princípio da otimalidade de Richard Bellman: uma sequência ótima de decisões tem a propriedade de que quaisquer que sejam o estado e a decisão inicial, as decisões remanescentes constituem uma sequência ótima de decisões com relação ao estado decorrente da primeira decisão, ou seja, toda sub caminho do caminho ótimo é ótima com relação a suas extremidades inicial e final (BELLMAN, 1957).

As equações de Bellman são descritas para grafos que podem apresentar arcos de peso negativo. Se baseiam no princípio da otimalidade de Richard Bellman que norteou o desenvolvimento da programação dinâmica (Apêndice A).

Considerando a possibilidade de decompor um caminho mínimo μ_j em um sub cami-

nho, também mínimo, μ_k, seguido de um arco (k, j), pode-se expressar o valor do caminho μ_j na forma: $\mu_j = \mu_k + c_{k,j}$, para todo vértice x_j diferente do vértice de origem x_1.

Deve-se achar qual vértice x_k deve pertencer à equação acima de modo a definir o valor do caminho μ_j.

Claramente, x_k deve ser escolhido de tal forma que o caminho μ_j seja o menor possível entre todas as possibilidades. Portanto, as equações de *Bellman* podem ser descritas na forma:

$\mu_1 = 0$

$\mu_j = Min_{k \neq j}\{\mu_k + c_{k,j}\}$, $j = 2, ..., n$

Vale ressaltar que a solução desse sistema é equivalente à solução do problema dual. Observando que a obtenção do valor mínimo de um conjunto de valores pode ser obtida por um problema de maximização parametrizado, associado ao fato de que o parâmetro deve ser menor ou igual a cada valor do conjunto, pode-se reescrever o sistema acima na forma:

Maximizar μ_j

Sujeito à:

$\mu_j \leq \mu_k + c_{k,j}$, para todo $k \neq j$

$\mu_1 = 0$

Generalizando a equação de *Bellman* para todo vértice x_j tem-se:

Maximizar $\mu_n + \mu_{n-1} + ... + \mu_2$

Sujeito à:

$\mu_1 = 0$

$\mu_j - \mu_k \leq c_{k,j}$, para todo arco (k, j) do grafo.

Tem-se que a solução das equações de *Bellman* para um grafo sem circuitos pode ser vista como um sistema triangular segundo a abordagem direta. Para um grafo acíclico é possível estabelecer uma enumeração no conjunto de vértices de tal forma que só existirá arco (k, j) se $k < j$. Assim, o sistema de inequações apresenta uma forma triangular superior, podendo ser resolvido diretamente pelo processo de eliminação e substituição de variáveis. Esse processo requer a realização de $(n-1) * n/2$ adições e $(n-1) * (n-2)/2$ comparações, sendo portanto de ordem máxima $O(n^2)$.

De outra maneira, pode-se analisar a solução das equações de *Bellman* por um processo recursivo, segundo a abordagem inversa, implementando um algoritmo baseado na técnica de divisão e conquista. Basicamente, esse algoritmo se divide em duas fases. A primeira é uma fase de decomposição no estilo *top-down*, na qual os sub caminhos são decompostos até que o sub caminho relacionado ao vértice de origem seja chamado recursivamente. Em seguida, o algoritmo apresenta uma fase de conquista no estilo *bottom-up*, onde os valores dos sub caminhos são conquistados sucessivamente até a determinação do caminho relacionado ao vértice objetivo. Caso o grafo apresente a existência de circuitos positivos a forma de solução do sistema se baseará na utilização de um método de aproximações sucessivas. A descrição do algoritmo em alto nível pode ser feita na forma:

Algoritmo DivisãoConquista;
Caminho(x_k: vértice);
Início
    Se $x_k = x_1$ Então
        $\mu_1 \leftarrow 0$; marcar x_k;
    Senão
        Para todo $v \in L^-[x_k]$ Faça
            Se v não marcado Então
                Caminho(v);
                $\mu_k \leftarrow$ Min $\{\mu_k, \mu_v + c_{v,k}\}$;
            Senão
                $\mu_k \leftarrow$ Min $\{\mu_k, \mu_v + c_{v,k}\}$;
            FimSe;
        FimPara;
        marcar x_k;
    FimSe;
Fim;
Início
    Para $i = 2, ..., n$ Faça
        $\mu_i \leftarrow \infty$;
    FimPara;
    Defina x_1 raiz;
    Defina x_n objetivo;
    Caminho(x_n);
Fim.

3.2.4 Algoritmo de *Bellman-Ford*

Este algoritmo propõe a solução das equações de *Bellman* quando o grafo apresenta circuitos de peso positivo, através de um método de aproximações sucessivas. Dessa forma tem-se, inicialmente, para a primeira iteração:

$\mu_1^{(1)} = 0$

$\mu_j^{(1)} = c_{1,j}$, para todo $j \neq 1$

Para as iterações seguintes, computa-se o valor de um caminho μ_j da iteração $m+1$ em função das aproximações obtidas até a iteração m. Portanto:

$\mu_1^{(m+1)} = 0$

$\mu_j^{(m+1)} = \text{Min } \{\mu_j^{(m)}, Min\{\mu_k^{(m)} + c_{k,j}\}\}$

Para cada vértice de x_j é necessário provar que as sucessivas aproximações são monotonicamente decrescentes, ou seja: $\mu_j^{(1)} \geq \mu_j^{(2)} \geq .... \geq \mu_j^{(n-1)}$, a fim de assegurar a convergência do algoritmo.

No passo inicial considera-se a hipótese válida para μ_1, pois seu valor é sempre zero. Na hipótese indutiva para um vértice x_j, $j \neq 1$, considera-se que $\mu_j^{(m)}$ seja o caminho de comprimento mínimo não contendo mais que m arcos.

No passo geral, admitindo que algum caminho mais curto, da origem para o vértice j, contenha mais que m arcos, então, o mesmo conteria $m+1$ arcos, equivalentes à soma de um arco (k, j) ao caminho $\mu_k^{(m)}$, considerando $\mu_k^{(m)}$ um caminho mínimo, pela hipótese indutiva. Dessa forma, minimizando $\mu_k^{(m)} + c_{k,j}$ para todas as escolhas possíveis, tem-se necessariamente: $\mu_j^{(m+1)} \leq \mu_j^{(m)}$.

É importante observar que existem no máximo $n-1$ melhorias para cada caminho do grafo, determinando uma cota superior para o algoritmo de $O(n^3)$.

Para a implementação do algoritmo de *Bellman-Ford*, foi reescrito o método de aproximações sucessivas na forma:

$\mu_1 = 0$

$\mu_j^{(m+1)} = \text{Min } \{\mu_j^{(m)}, (\mu_1^{(m)} + c_{1,j}), ..., (\mu_{k-1}^{(m)} + c_{k-1,j}), (\mu_{k+1}^{(m)} + c_{k+1,j}), ..., (\mu_n^{(m)} + c_{n,j})\}$

Como o processo de convergência é assegurado, o algoritmo pode ser implementado utilizando-se o aninhamento de três *loops* iterativos, na forma:

```
Algoritmo BellmanFord;
Início
    μ_1 ← 0;
    μ_j ← c_{1,j}, para todo j ≠ 1;
    melhora ← VERDADEIRO;
    Enquanto melhora Faça
            melhora ← FALSO;
            Para i ← 2 até n Faça
                Se Atualiza(μ_i, x_i) Então
                    melhora ← VERDADEIRO;
                FimSe;
            FimPara;
    FimEnquanto;
Fim.
Função Atualiza(var μ_i: real; x_i: vértice): lógico;
Início
    Atualiza ← FALSO;
    Para j ∈ L^-[x_i] Faça
            Se μ_i < μ_j + c_{j,i} Então
                Atualiza ← VERDADEIRO;
                μ_i ← μ_j + c_{j,i};
            FimSe;
    FimPara;
Fim.
```

3.3 Aprendizado por Reforço

Esta seção introduz os conceitos relacionados ao paradigma de aprendizado por reforço, usado no desenvolvimento dos algoritmos destinados à determinação e aprendizado de caminhos por parte de um agente móvel no ambiente de *grid* com obstáculos.

Inicialmente, descreve-se os principais aspectos do aprendizado por reforço, destacando o processo de interação do agente com o ambiente, o dilema exploração *versus* explotação e o aprendizado de uma política de longo prazo. Em seguida, apresenta-se os principais componentes do modelo de aprendizado por reforço, incluindo a sua política, função de transição, função de recompensa e função valor. Finalmente, descreve-se as duas

principais técnicas ou métodos utilizados para a solução deste problema de aprendizado na sua forma determinística.

3.3.1 O Problema de Aprendizado e sua Dinâmica

O modelo básico do aprendizado por reforço pode ser considerado, segundo (KAELBLING; LITTMAN; MOORE, 1996), como a atuação dinâmica de um agente móvel em um ambiente real ou simulado, no sentido de estabelecer uma política ou uma regra de conduta relacionada à satisfação de um conjunto de objetivos. Durante o processo ocorre uma interação entre o agente e o ambiente, sendo a interface do mesmo regulada por uma sequência de mecanismos de percepção, ação e recompensa, conforme o esquema apresentado na Figura 7.

A cada iteração ou instante de tempo o agente determina a execução de uma ação com base na política existente e na percepção do ambiente representada pelo estado atual. Provocando uma transição de estados responsável por uma mudança no ambiente. Em função desta mudança, o ambiente responde imediatamente com a sinalização de uma medida de recompensa a qual provocará uma alteração nos valores dos estados. A ocorrência de sucessivos ajustes nestes valores provocará ao longo do tempo uma correção da política, definindo desta forma um processo dinâmico e interativo de aprendizado.

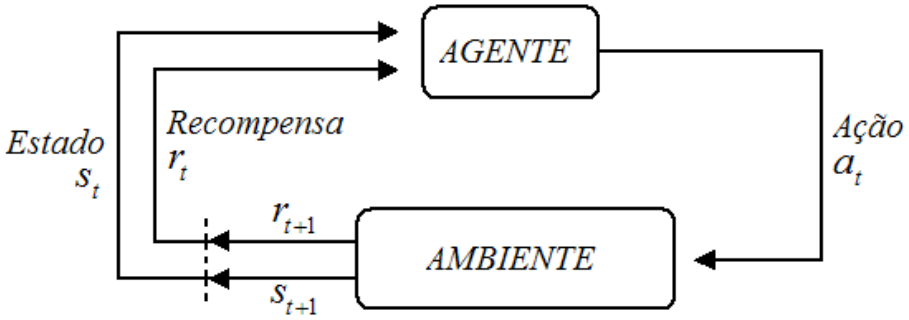


Figura 7: Interação do agente com o ambiente

3.3.2 *Feedback* e Recompensas

No mecanismo de percepção o agente recebe como entrada uma indicação do seu estado atual por meio de sensores, promovendo um processamento perceptual e gerando uma representação icônica do ambiente ao seu redor na forma de um vetor de características. Em seguida, o mesmo realiza o processamento de uma ação como saída. A ação executada

promove um efeito de mudança no ambiente e uma consequente transição de estados. Sendo este efeito transmitido na forma de $feedback$ ao agente por meio de um sinal de reforço, ou $reward$, medido por uma função de recompensa, ou função de retorno, segundo a denominação da Programação Dinâmica.

É possível ao agente aprender ou aprimorar a sua conduta por um processo de tentativa e erro, onde as ações que provocam um melhor retorno em longo prazo são escolhidas com maior frequência formando a base de aprendizado do agente. De uma forma sucinta, o agente formula uma política responsável pelo mapeamento de estados em ações que maximizam uma função de avaliação. Esta função consiste em uma sequência de retornos computados segundo o efeito das ações considerando um objetivo de longo prazo.

3.3.3 Aprendizado por Reforço *versus* Aprendizado Supervisionado

O aprendizado por reforço é diferente do aprendizado supervisionado. O aprendizado supervisionado consiste em aprender a partir de um conjunto de treinamento consistente e representativo, formado através de exemplos fornecidos por alguma fonte de conhecimento externo. Representa um paradigma importante de aprendizado, mais estudado e utilizado no aprendizado de máquinas, no reconhecimento de padrões e no projeto de redes neurais artificiais. Contudo, não é adequado para processos de aprendizado interativos. Nos processos interativos, é praticamente inviável obter exemplos de um comportamento que seja correto e representativo em todas as situações pelas quais o agente deverá interagir. Sendo assim, em um ambiente interativo, onde o aprendizado será necessário, o agente deverá aprender a partir, sobretudo, de sua própria experiência.

A ideia central do aprendizado por reforço consiste no fato do agente aprender como mapear situações ou estados em ações com o objetivo de maximizar uma recompensa que pode ser positiva ou negativa, sendo representada por alguma avaliação numérica. O aprendizado não é descobrir quais ações tomar, contrapondo-se a forma supervisionada e instrutiva de aprendizado de máquina, mas sim descobrir que ações maximizam a recompensa, experimentando se necessário todas as ações possíveis. As ações podem afetar não somente a recompensa imediata, mas também a recompensa imediatamente posterior e todas as recompensas subsequentes. A busca da maior recompensa através de tentativa e erro e a influência das ações nas recompensas futuras são as características principais que diferenciam o aprendizado por reforço.

Pode-se também empregar o aprendizado por reforço no aprendizado de heurísticas e

funções de avaliação, tornando-o uma técnica aliada nos sistemas de solução de problemas que empregam uma busca heurística no espaço de estados em algum domínio específico.

A principal vantagem do uso do aprendizado por reforço consiste na possibilidade de se utilizar os exemplos de treinamento obtidos diretamente da sequência temporal de ações considerando seus respectivos efeitos ou retornos, não sendo necessário o conhecimento da função de retorno dos estados finais ou de saída, característico de um estilo de correção baseado na ideia de punição e recompensa ou do aprendizado instrutivo.

Neste sentido, uma das técnicas deste paradigma de aprendizado conhecida como **método de diferenças temporais** realiza a correção dos parâmetros do modelo com base na diferença entre duas predições, ou valores de estados, sucessivamente, até se atingir um estado final do problema caracterizando uma forma de treinamento *online* e incremental que requer pouco esforço computacional. Esses métodos se relacionam sobretudo ao problema de predição de sistemas, no sentido de utilizarem experiências passadas como forma de se prever o comportamento dos sistemas no futuro. Constituem uma forma de aprendizado diferente do paradigma de aprendizado supervisionado, pois estão presentes em situações onde nem sempre é possível a existência de um conjunto de treinamento que possibilite o aprendizado supervisionado. Além disso, estão mais adaptadas às soluções de problemas cuja dinâmica exige uma forma contínua e sequencial de ajuste nos parâmetros do modelo, resultante de uma interação do agente com o seu ambiente.

3.3.4 Predição em um Único Estágio e em Multi-Estágios

Os problemas de predição podem ser classificados em um único estágio ou em múltiplos estágios. Nos problemas de um único estágio o efeito de cada predição sobre a saída é conhecido imediatamente, enquanto que, nos problemas de múltiplos estágios, o efeito da predição quanto a sua correção somente é conhecido no final do processo. Os problemas de predição de um único estágio estão diretamente relacionados com o aprendizado supervisionado.

Entretanto, a maior parte dos problemas se caracterizam como problemas de predição de múltiplos estágios. Como exemplo, tem-se a previsão de um resultado eleitoral, o qual se modifica a cada chegada de novas apurações, a estimativa de se chegar em casa, a qual se altera a cada mudança no trânsito, e a previsão de inflação anual em uma economia, a qual se corrige a cada mês.

3.3.5 Exploração *versus* Explotação

A ideia de que o aprendizado se dá através da interação com o ambiente é provavelmente a primeira a ocorrer. Quando uma criança brinca ou observa algo, não há um professor lhe auxiliando explicitamente, mas a criança tem uma conexão sensorial e motora direta com o ambiente. Exercitar esta conexão produz uma grande quantidade de informações relacionadas às causas e consequências, e ao que deve ser feito para atingir certos objetivos. Durante a vida, estas interações formam a principal fonte de conhecimento sobre o ambiente. Sob esse aspecto, o principal objetivo do aprendizado é controlar a dinâmica do ambiente ou o processo de interação através da prática de uma sequência correta de ações.

Um dos maiores desafios do aprendizado por reforço é a solução do dilema exploração *versus* explotação. **Exploração** significa o agente selecionar ações em que o valor da recompensa não é conhecido ou não é conhecido totalmente. **Explotação** significa o agente escolher a ação que possui a maior recompensa conhecida. O dilema se baseia no fato de que não é possível utilizar somente a exploração ou somente a explotação sem falhar no objetivo de maximizar a recompensa. Se o problema for estocástico, cada ação precisa ser selecionada muitas vezes para se conseguir uma estimativa confiável da recompensa esperada. Porém, é indispensável que o agente seja capaz de buscar soluções alternativas, mesmo que seja de forma aleatória, pois em uma dessas buscas ele pode encontrar uma solução melhor do que uma já existente. Por isso é de extrema importância haver um equilíbrio entre as tendências exploratórias e exploratórias.

3.4 O Modelo do Aprendizado por Reforço

Além do agente e do ambiente, citados anteriormente, é possível identificar quatro outros componentes em um modelo de aprendizado por reforço: uma política, uma função de transição, uma função de recompensa, uma função de valor e opcionalmente um modelo do ambiente.

3.4.1 Política π

Uma política define o modo como o agente se comporta durante o tempo. Basicamente, uma política é um mapeamento dos estados do ambiente em ações a serem selecionadas quando encontrados estes estados, com a finalidade de atingir o objetivo

do problema, determinando, no sentido psicológico, um conjunto de associações do tipo estímulo-resposta. Em alguns casos, a política pode ser uma simples função ou tabela, enquanto em outros, a mesma pode envolver muito poder computacional como, por exemplo, um processo de busca. A política é o centro do aprendizado por reforço, no sentido que, uma vez estabelecida, é suficiente para determinar o comportamento do agente. Em geral, as políticas podem ser estocásticas.

3.4.2 Função de Transição δ

A função de transição define as transições dos estados. Para um estado atual s_t, definido para um instante de tempo t, pode-se obter uma transição para um estado s_{t+1}, escolhendo para a função de transição uma ação a apropriada, ou seja:

$s_{t+1} = (s_t, a_t)$.

Em muitos problemas reais não tem-se o conhecimento à priori desta função, tornando o processo de aprendizado totalmente dinâmico e desprovido de qualquer forma de planejamento. Nos problemas onde é possível o conhecimento a priori da função de transição, é factível a idealização de um modelo mental ou modelo teórico do ambiente por parte do agente, permitindo a realização de planejamento que possibilitará a aceleração do processo de aprendizado.

3.4.3 Função de Recompensa r

Uma função de recompensa define os objetivos imediatos do problema de aprendizado por reforço. Ela mapeia os estados percebidos s, ou os pares estado-ação (s, a) do ambiente, em um único número que é a recompensa. O principal objetivo do agente é maximizar a recompensa total que é recebida ao longo do tempo. A função de recompensa define quais eventos são bons e quais são ruins para o agente.

3.4.4 Função de Valor V

Enquanto a função de recompensa indica qual ação é imediatamente mais vantajosa, a função de valor, geralmente associada a uma política, especifica o valor de um determinado estado como o total de recompensas que um agente pode esperar acumular durante todo o período de planejamento, partindo deste mesmo estado, podendo considerar ou não um fator de desconto, representado pelo parâmetro γ.

A função de valor indica a atratividade em longo prazo dos estados, levando em conta os estados mais prováveis de serem selecionados mais adiante e a respectiva recompensa desses estados. Esta função é importante devido ao fato de poder haver estados que apresentam, inicialmente, uma recompensa baixa, mas são seguidos de vários estados que possuem recompensa alta, elevando a função de valor dos estados anteriores significativamente. Entretanto, o contrário também pode acontecer, ou seja, estados com uma recompensa alta, mas que são seguidos de estados com uma recompensa baixa, e, portanto, possuem valores finais baixos, sendo pouco atrativos em longo prazo.

Sem as recompensas não existem valores, e a única maneira de se estimar valores é conseguindo o máximo de recompensas possível. Entretanto, é com os valores que a maior preocupação na hora de tomar as decisões, pois a escolha de ações deve ser feita com base no julgamento de valores e não nas recompensas imediatas.

Recompensas são fáceis de se determinar, valores são bem mais difíceis, pois têm que ser estimados a cada sequência de observações que um agente faz, para que sejam corrigidos e sua estimativa torne-se mais precisa. De fato, o componente mais importante de um algoritmo de aprendizado por reforço é um método eficiente de se estimar valores.

3.5 Algoritmo *Q-Learning*

A tarefa de aprendizado de um agente que interage com um ambiente pode ser definida como determinar uma política ótima que maximize os valores das recompensas em longo prazo, e estabeleça uma estratégia de controle que determina uma sequência de ações, ou seja, a partir de um conjunto de políticas:

$\pi : S \rightarrow A$

Responsáveis pelo mapeamento de um conjunto de estados S, em um conjunto de ações A, o agente deseja determinar uma política ótima, definida como:

$\pi* \equiv Max_{\pi}\{V^{\pi}(s), (\forall s)\}$

A função valor é computada na forma acumulativa, para um determinado estado s_t, a partir da soma ponderada dos retornos estabelecidos em função de uma política, considerando um fator de desconto e uma função de retorno r, para um determinado horizonte que pode ser finito ou infinito. Portanto:

$V^{\pi}(s_t)r_t + \gamma.r_{t+1} + \gamma^2.r_{t+2} + ...$, para $0 < 1$, sendo $r_t = r(s_t, a_t)$

Esse problema, relacionado ao aprendizado de uma política, não pode ser tratado sob a ótica do aprendizado supervisionado, já que os pares do conjunto de treinamento, definidos como $(s, a) = \pi(s)$, não são previamente conhecidos. O que se tem é uma sequência de retornos, proveniente da transição de estados resultantes da execução das ações por parte do agente, caracterizando um problema de atribuição de crédito temporal.

Tradicionalmente, quando se tem um conhecimento prévio de todo espaço de estados do problema, envolvendo todos os possíveis estados e suas respectivas transições, e um conhecimento completo da função de recompensa ou de retorno para cada transição, a definição de uma política ótima na forma estocástica se encontra fundamentada na teoria de Processos de Decisão Markovianos e, nos casos determinísticos, na solução de um problema de Programação Dinâmica, conforme descrito no Apêndice A.

3.5.1 Solução Aproximada e Função Q

Considerando-se a ausência das informações relacionadas às funções de transferência, e também o problema definido como a maldição da dimensionalidade que afeta a viabilidade computacional do algoritmo de Programação Dinâmica, explicado no Apêndice A, podem-se adotar como estratégia de solução uma abordagem sub-ótima para o tratamento do problema de aprendizado de um agente. Considerando o critério proposto por (BELLMAN, 1957), a escolha de uma ação ótima a^* no estado s, está relacionada à maximização de um valor equivalente à soma do retorno imediato produzido pela ação escolhida com os valores das recompensas futuras a partir do estado sucessor, associada ao fator de desconto. Portanto:

$\pi * (s) = argMax_a\{r(s, a) + \gamma.V^*(\delta(s, a))\}$

Para que esta formulação seja empregada, é necessário que o agente aprenda a função de valor V^*, o que exige o conhecimento prévio da função de transição e da função de recompensa r, para todas as transições possíveis, o que, como foi visto, nem sempre é possível. Para resolver este problema, (WATKINS; DAYAN, 1989) propõe o aprendizado de uma função alternativa, conhecida como função Q, que consiste de uma função de avaliação definida na forma:

$Q(s, a) \equiv r(s, a) + \gamma.V^*(\delta(s, a))$

Considerando:

$\pi * (s) = argMax_a Q(s, a)$ e

$V*(s) = Max_{a'}Q(s,a')$

Pode-se redefinir a função Q na forma recursiva como:

$Q(s,a) \equiv r(s,a) + \gamma.Max_{a'}Q(\delta(s,a),a')$

É importante observar que o aprendizado da função Q torna desnecessário o aprendizado da função valor $V*$ e, também, do conhecimento a priori das funções de recompensa r e de transição. Em uma forma de treinamento *online* os valores provenientes destas funções são obtidos diretamente da resposta do ambiente. Para estimar a função Q (WATKINS; DAYAN, 1989) propôs um algoritmo de treinamento baseado em correções por sucessivas diferenças temporais conhecido como *Q-Learning*. Neste sentido, define-se uma função de aproximação Q que é atualizada de forma iterativa da seguinte forma:

$Q(s,a) \leftarrow r(s,a) + \gamma.Max_{a'}Q(s',a')$, sendo $s' = \delta(s,a)$

Ou seja, no estado s o agente escolhe uma ação a que maximiza a sua escolha, considerando os valores aproximados da função Q entre todas as alternativas possíveis. Dessa forma, os valores mais recentes da função Q para o estado sucessor s' são utilizados para a atualização do valor da função Q para o estado s. Este algoritmo pode ser estendido para situações de não determinismo, onde as funções de recompensa e de transição possuem um caráter randômico.

3.5.2 *Q-Learning Incremental*

A atualização dos valores da função Q, considerando somente o efeito da ação e a consequente recompensa associada, implementa um processo de aprendizado baseado na adoção de um critério de correção por retorno ou diferença imediata.

Entretanto, é possível a atualização dos valores da função Q considerando o efeito de um conjunto de ações antecessoras associadas às suas respectivas recompensas, com a utilização do fator de desconto λ no sentido de incrementar ou tornar mais influente as decisões tomadas mais recentemente. Este processo, que também pode ser definido como um algoritmo de aprendizado de múltiplos passos, tem uma relação direta com o método mais geral de diferenças temporais, conhecido como $TD(\lambda)$.

Na nova implementação a única diferença em relação ao algoritmo clássico do *Q-Learning*, que atualiza um único estado por vez baseado no retorno imediato, está na forma de atualização dos valores da função Q. Considerando o valor da taxa de aprendizagem igual a um , $\beta = 1$, e a correção em dois passos:

$Q(x,a) = r(x,y) + \gamma.(Max_{a'}\{Q(y,a')\}) + \gamma.(Max_{a''}\{Q(z,a'')\})$

Para duas transições sucessivas, representadas pelas respectivas funções de transferência:

$z = \delta(y,a')$ e $y = \delta(x,a)$

A exemplo do método $TD(\lambda)$ considera-se a existência de um número variável de passos no processo de correção da função. Neste caso, o valor de cada estado fica ponderado pela n-ésima potência da taxa de desconto, ou seja multiplica-se por γ^n, considerando n o número de transições observadas.

Uma alternativa para melhorar a eficiência do algoritmo *Q-Learning* consiste na atualização simultânea de vários estados para a mesma ação escolhida, que possuam algum grau de similaridade com o estado atual. Neste caso, é necessária a implementação de um processo de matching ou de reconhecimento por algum tipo de técnica não supervisionada como, por exemplo, algoritmos de análise de *cluster* ou de memória associativa.

As maiores deficiências do algoritmo *Q-Learning* se referem a sua baixa taxa de convergência e a necessidade de armazenamento explícito dos valores da função Q.

4 Aprendizado Supervisionado

Neste capítulo são abordadas especificamente as redes neurais e o principal algoritmo de aprendizado deste trabalho, o *Perceptron*.

O aprendizado supervisionado é um sub-campo da Inteligência Artificial dedicado ao desenvolvimento de algoritmos e técnicas de aprendizado. O processo de treinamento supervisionado de redes neurais artificiais multicamadas é equivalente a um problema de otimização não-linear irrestrito, onde uma função de erro global é minimizada a partir do ajuste de parâmetros, ou pesos, da rede neural. O Apêndice B discorre brevemente sobre a teoria da otimização não-linear.

4.1 Redes Neurais Artificiais (RNA's)

Redes Neurais Artificiais apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência.

(HAYKIN, 2001) diz que as RNAs são processadores massivamente paralelos e distribuídos que têm uma propensão natural para armazenar o conhecimento proveniente da experiência e torná-lo útil. Assemelhando-se ao cérebro humano em dois aspectos. Primeiro, o de que o conhecimento é adquirido pela rede através de um processo de aprendizado. Segundo, o de que as intensidades das conexões entre neurônios, conhecidos como pesos sinápticos, são usados para armazenar o conhecimento.

As redes neurais artificiais possuem algumas características que as tornam satisfatoriamente aplicáveis, tais como aprender através de conjuntos de exemplos e apresentar respostas coerentes para entradas não vistas durante o treinamento, e adaptar-se ao seu novo ambiente através de alterações em seus pesos sinápticos, no caso de operarem em outro ambiente. Também podem ser projetadas para alterarem seus pesos em tempo real ou para operarem em ambientes que variem com o tempo.

O aprendizado em uma rede neural é um processo por meio do qual os parâmetros

livres são adaptados através de um processo contínuo de estímulos por um ambiente no qual a rede está inserida. O tipo de aprendizagem é definido pela maneira pela qual os parâmetros são modificados, (HAYKIN, 2001).

4.1.1 Características de uma Rede Neural

O elemento básico de uma RNA é o neurônio. Os neurônios artificiais são estruturas lógicas cujo objetivo é simular o funcionamento de um neurônio biológico, fazendo com que a rede atue de forma indutiva (NETO et al., 1997). Pode-se representar um neurônio de uma rede neural artificial como uma função matemática consistindo num somatório ponderado de várias entradas produzindo um determinado sinal de saída (Fig. 8), conhecido como o neurônio de McCulloch-Pitts, o primeiro modelo de redes neurais artificiais proposto (MCCULLOCH; PITTS, 1943).

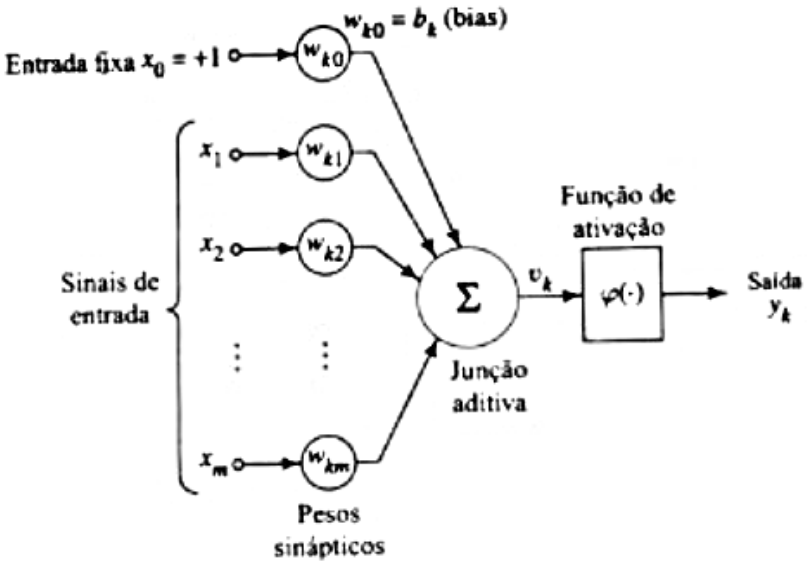


Figura 8: Modelo de um neurônio artificial

Cada sinal de entrada x_i é multiplicado por um peso w_{ki}, sendo k o neurônio em questão. Estes valores são somados na junção aditiva e depois é aplicada a função de ativação $\varphi(.)$ que restringe o valor da saída a um intervalo pré-definido, ou seja, se o resultado da junção aditiva exceder um certo limite, *threshold*, a unidade produz uma determinada resposta de saída. Tem-se que a entrada fixa $x_1 = 1$ e do bias $w_{k1} = b_k$ representam um peso externo que reforça ou diminui a entrada da função de ativação.

As equações podem ser descritas como:

$u_k = \sum w_{ki}.xi$

$y_k = \varphi(u_k)$

A função de ativação $\varphi(.)$ pode ser construída por diferentes funções matemáticas. Porém, é importante que tal função seja diferenciável, sendo assim, é possível saber seus pontos de máximo e mínimo. Comumente é estipulado que se $\varphi(u_k)$ ultrapassar o *threshold* então ocorre a ativação, $y_k = 1$, caso contrário, $y_k = 0$

Uma rede neural artificial pode ser composta por vários neurônios. Cada neurônio faz operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. É especificada, principalmente, pela sua topologia, pelas características dos nós e pelas regras de treinamento.

As duas classes de redes neurais mais usuais são a **monocamada** e a **multicamada**.

Na monocamada existe somente uma camada de neurônios ligados diretamente aos nós de entrada e eles mesmos já fornecem os dados de saída, só separa completamente duas classes se elas forem linearmente separáveis.

A **multicamada** possui uma ou mais camadas de neurônios escondidos. Sua função é processar os sinais de entrada antes de enviá-los aos neurônios de saída (Fig. 8). Apesar da maior complexidade, essa arquitetura possibilita uma melhor qualidade de treinamento pois há maior interação entre os neurônios . Quando todos os neurônios estão conectados a todos os outros neurônios essa rede é dita completamente conectada, caso contrário é chamada parcialmente conectada. Podem atuar em problemas não-linearmente separáveis.

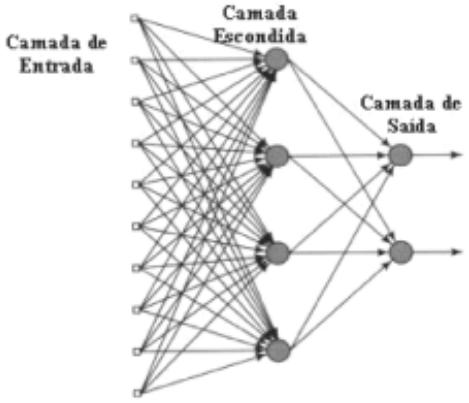


Figura 9: Modelo de uma rede neural artificial multicamada

Para que uma rede neural artificial possa fornecer seus resultados é necessário passar por uma fase de treinamento, onde seus pesos são ajustados de forma que ela se adapte aos diferentes estímulos de entrada de acordo com os padrões apresentados. Durante esta fase de treinamento, ocorre o seu aprendizado.

Denomina-se **ciclo** uma apresentação de todos os N pares (entrada e saída) do conjunto de treinamento no processo de aprendizado. A correção dos pesos num ciclo pode ser executado de duas formas. Primeira, a cada apresentação à rede de um exemplo do conjunto de treinamento, onde ocorrem N correções e cada correção de pesos baseia-se somente no erro do exemplo apresentado naquela iteração. Segunda, pelo ***modo batch***, onde apenas uma correção é feita por ciclo. Todos os exemplos do conjunto de treinamento são apresentados à rede, seu erro médio é calculado e a partir deste erro fazem-se as correções dos pesos.

4.2 Aprendizado por Correção de Erros

No intuito de ajustar os pesos das correções pode ser calculada a diferença entre a saída real gerada pela rede e a saída desejada, fornecida em um aprendizado supervisionado, obtendo assim o erro atual de uma rede neural.

Durante o aprendizado supervisionado, os erros vão sendo calculados sucessivamente, até que cheguem a um valor satisfatório definido a priori, geralmente um valor algumas dezenas de grandeza menor quando comparados com os valores atuantes na diferença. Sendo assim, surge uma curva de erros, a qual está diretamente relacionada à natureza do modelo de neurônio utilizado.

Este processo utiliza algoritmos para caminhar sobre a curva de erros, com o intuito de alcançar um erro menor do que o definido a priori. Devido a essa abordagem muitas vezes, o algoritmo não alcança este mínimo global, atingindo o mínimo local, porém esse fato não é de grande importância se o algoritmo conseguiu alcançar um mínimo local menor que o erro máximo estipulado. Tem-se então que:

$$e_k = d_k - y_k$$

Sendo e o erro; d a saída desejada apresentada durante o treinamento; y a saída real da rede; e k o estímulo em questão.

Para a correção do erro, os pesos da rede devem ser ajustados, de forma a aproximar a saída real à desejada. Uma das regras de aprendizado para RNA, bem conhecida é a regra *delta* (WIDROW, 1962), também conhecida como $LeastMeanSquare$ (LMS), que minimiza o erro médio quadrático. Esta é apresentada a seguir, e seu ajuste dependerá dos seguintes fatores: do próprio erro calculado, do valor do estímulo de entrada que é transmitido pelo peso a ser ajustado, e também da taxa de aprendizado, a qual relaciona-

se à cautela com que a curva de erros é percorrida. Para um dado estímulo k, no passo de treinamento n:

$\Delta w_{i,j} = \eta.e_k(n).x_j(n)$

Onde $\Delta w_{i,j}$ é valor de ajuste a ser acrescido ao peso $w_{i,j}$; η é a taxa de aprendizado; $e_k(n)$ é o valor do erro; e $x_j(n)$ é o valor do estímulo.

O valor atualizado do peso será:

$w(n+1) = w(n) + \Delta w_{i,j}(n)$

Logo, é possível minimizar a função de erro, também conhecida como função de custo, utilizando a regra *delta* para corrigir os valores dos pesos:

$\epsilon(n) = \frac{1}{2}.e^2(n)$

Onde $\epsilon(n)$ é o erro da rede no passo n do treinamento; e $e(n)$ é o valor da função de custo no passo n do treinamento.

Na regra *delta* generalizada ou algoritmo de retro propagação, a aprendizagem supervisionada ocorre através de exemplos, em tempo discreto e auxiliada por um método de gradiente descendente, para corrigir os erros.

Os pesos sinápticos são ajustados de acordo com o erro quadrático para todos os padrões do conjunto de treinamento. O processo de redução gradativa do erro tende à convergência, onde o erro é estável. A evolução do processo de aprendizagem ocorre, até que algum critério seja satisfeito, como um valor mínimo de erro global ou uma diferença sucessiva mínima entre erros.

4.3 *Perceptron*

O algoritmo *Perceptron* foi desenvolvido por (ROSENBLATT, 1958) e é composto por neurônios do modelo de McCulloch-Pitts (Fig. 8), com função de limiar, e aprendizado supervisionado. O *Perceptron* é a forma mais simples de um RNA usada para classificação cujos padrões estão em lados opostos de um hiperplano. Consiste de um único neurônio com pesos sinápticos ajustáveis e um possível *bias* b cujo efeito é deslocar a fronteira de decisão em relação a origem, aumentando ou diminuindo o tempo de conversão do algoritmo.

A soma do produto entre pesos w e entradas x_i, também chamado de vetor característica, alimenta o neurônio de saída, seu resultado é comparado com um valor limiar,

geralmente 0. A função de ativação, ou função de perda, neste caso é uma função limiar $J(u)$ que modela a característica tudo ou nada deste neurônio. Nos neurônios construídos com essa função, a saída será igual a 0, caso o valor da função φ seja negativo. E 1 nos casos em que o valor seja positivo ou 0. Matematicamente tem-se:

Função de perda 0 ou 1:

$$J(u) = \sum_i Max\{0, \varphi(-y_i < w, x_i >)\}$$

A função $\varphi(-y_i < w, x_i >)$ é uma função constante por partes e não diferenciável. Sendo $z = -y_i < w, x_i >$ tem-se $\varphi(z) = 1$ se $z \geq 0$, caso contrário $\varphi(z) = -1$. Onde y_i representa a saída desejada, ou seja, a classe a qual o estímulo pertence. Se y_i e $< w, x_i >$ pertencerem a mesma classe, ou seja, se $< w, x_i >$ tiver o mesmo sinal de y_i o resultado de φ será negativo, o que resultará num $J(u) = 0$ acarretando a não correção.

$$J(w) = \sum_i Max\{0, -y_i < w, x_i >\}$$

No método da descida mais íngreme (HAYKIN, 2001) os ajustes sucessivos ao vetor de peso w são na direção da descida mais íngreme, ou seja, em uma direção oposta ao vetor de gradiente. Com o gradiente local $\nabla J(w) = -y_i x_i$ tem-se que a atualização do vetor w se dá na forma: $w(t+1) = w(t) + \eta x_i y_i$. Onde η é a taxa de aprendizado $(0 < \eta \leq 1)$. Tem-se então que a correção só acontece se $J(w) = 1$, ou seja, se o elemento x_i está sendo classificado na classe errada. Se x_i já estiver sendo classificado na classe certa não há necessidade de correção.

Visto que o y_i representa a classe correta ao qual o elemento x_i deve pertencer e, desta forma, definindo se ocorrerá a correção e seu sinal. O enunciado acima poderia ser reescrito da forma que (HAYKIN, 2001) o descreveu:

Sejam C_1 e C_2 classes hipotéticas, no primeiro caso, se a amostra x_i é corretamente classificada pelo vetor de pesos w, então não há correção:

$w_{(t+1)} = w_{(t)}$ se $< w, x_i > > 0$ e x_i pertence a classe C_1

$w_{(t+1)} = w_{(t)}$ se $< w, x_i > \leq 0$ e x_i pertence a classe C_2

Caso contrário o vetor de pesos w é atualizado de acordo com a regra:

$w_{(t+1)} = w_{(t)} - \eta x_i$ se $< w, x_i > > 0$ e x_i pertence a classe C_2

$w_{(t+1)} = w_{(t)} + \eta x_i$ se $< w, x_i > \leq 0$ e x_i pertence a classe C_1

Durante o processo de treinamento do *Perceptron*, busca-se encontrar um conjunto de pesos que defina um hiperplano ortogonal a w que separe as diferentes classes, de forma que a rede classifique corretamente cada entrada x_i que está sendo incluída no processo de aprendizado.

Assim, facilitando a implementação do algoritmo, a adaptação do vetor do peso w pode ser resumida adequadamente na regra de aprendizagem por correção de erro, derivada da regra *delta*. Tem-se que se y_i for a classe resposta desejada (1 ou -1) e $d_i = sinal < w, x_i >$ for a classe calculada atual:

$$w_{(t+1)} = w_{(t)} + \eta \, sinal[y_{(t)} - d_{(t)}]x_i$$

O algoritmo de treinamento do *Perceptron* sempre chega a uma solução para o problema de separação de duas classes linearmente separáveis em um tempo finito. (ROSENBLATT, 1958) provou isto através do Teorema da Convergência do *Perceptron*, que enuncia que este algoritmo consegue encontrar um conjunto de pesos ideais, ou seja um w, para que a rede classifique corretamente as entradas desde que seja aplicado a duas classes linearmente separáveis, pois posiciona a superfície de decisão na forma de um hiperplano entre as duas classes (HAYKIN, 2001). Esta restrição não se aplica ao *Perceptron Multicamada*, pois este consegue tratar problemas não-linearmente separáveis. Problemas não-linearmente separáveis não são tratados neste trabalho.

4.4 *Perceptron com Margem*

(DUDA; HART; STORK, 2001) propõem uma versão alternativa para o algoritmo *Perceptron* incluindo a utilização de uma regra de incremento variável para uma função de perda quadrada e um valor fixo de margem γ no sentido de adaptar a sua solução ao método de relaxação. Considerando a introdução do parâmetro γ, a solução do problema consiste na determinação de uma solução viável para o sistema de inequações lineares na forma:

$$y_i(< w, \varphi(x_i) >) \geq \gamma$$

Entretanto, caso se utilize uma regra de incremento fixo e não seja possível limitar o valor da norma quadrática do vetor w, com a adição da restrição adicional de normalização $||w||_2 = 1$, o sistema de inequações, se linearmente separável, apresentará sempre uma solução viável considerando o crescimento da norma e, consequentemente, do valor do produto interno, para qualquer valor de margem γ. Para se resolver este problema é

necessário estabelecer alguma forma de regularização no sentido de controlar ou de limitar o valor da norma do vetor w (LEITE; NETO, 2008).

Assim, procurou-se uma nova formulação para o modelo *Perceptron* no sentido de garantir que o conjunto de exemplos guarde uma distância mínima em relação ao hiperplano separador sem limitar diretamente o valor da norma do vetor w.

Para tanto, é considerada a restrição de que cada amostra deva possuir um valor de margem geométrica correspondente superior ou igual ao valor estabelecido como margem fixa, sendo o valor da margem geométrica definido como o valor da margem funcional da respectiva amostra dividido pelo valor da norma euclidiana do vetor w. Isto equivale a realização do produto interno do vetor $\phi(x_i)$ pelo vetor unitário de direção w, representado por $w/||w||_2$.

Neste sentido, deve-se resolver o seguinte sistema de inequações não lineares para determinado valor de margem fixa representado pelo parâmetro γ_f (LEITE; NETO, 2008):

$$\frac{y_i(< w, \varphi(x_i) >)}{||w||_2} \geq \gamma_f \;\; ou \;\; y_i(< w, \varphi(x_i) >) \geq \gamma_f.||w||_2$$

Em função desta modificação, torna-se necessário reescrever a função de perda do modelo de forma a possibilitar a obtenção de uma nova regra de correção. A nova função será equivalente à soma dos valores das respectivas margens geométricas dos exemplos que forem menor que o valor da margem fixa, descontado o valor da margem. Ou seja:

$$J(w) = (\sum_i Max\{0, \gamma_f - \frac{y_i(< w, \varphi(x_i) >)}{||w||_2)}\}, \;\; (x_i, y_i) \in Z$$

Ou então, através de algumas manipulações algébricas:

$$J(w) = -(\sum_i y_i(< w, \varphi(x_i) >) - m.\gamma_f.||w||_2), \;\; (x_i, y_i) \in M \;\; e \;\; |M| = m$$

Portanto, ao contrário do algoritmo básico do perceptron, considera-se também como erro, aqueles exemplos que, embora classificados corretamente, não estejam a uma distância mínima, no sentido geométrico, do hiperplano separador. (KIVINEN; SMOLA; WILLIAMSON, 2002) definem este tipo de correção como a ocorrência de erros de margem.

A solução do sistema de inequações pode ser considerada como aquela que minimiza a função de erro J. Neste sentido, tomando-se o gradiente da função em relação ao vetor w, tem-se a seguinte regra de correção, caso ocorra um erro, $yi(< w, \varphi(xi) >) <$

$\gamma_f.||w||_2$, aplicada a uma determinada amostra $(x_i, y_i) \in M$, onde η se refere a taxa de aprendizagem:

$$w_{(t+1)} = w_{(t)} - \eta(\gamma_f.\frac{w}{||w||_2} - \varphi(xi).yi)$$

Esta equação de correção possui duas interpretações diretas. A primeira, baseada na observação de que o termo $w/||w||_2$ representa o vetor unitário de direção w, sugere que da parcela de correção do vetor normal relacionada ao exemplo apresentado seja retirada uma quantia relativa ao valor da margem fixa (Fig. 10). A segunda, relacionada à forma alternativa da equação:

$$w_{(t+1)} = w_{(t)}(1 - \frac{\eta.\gamma_f}{||w||_2}) + \eta.\varphi(x_i).y_i$$

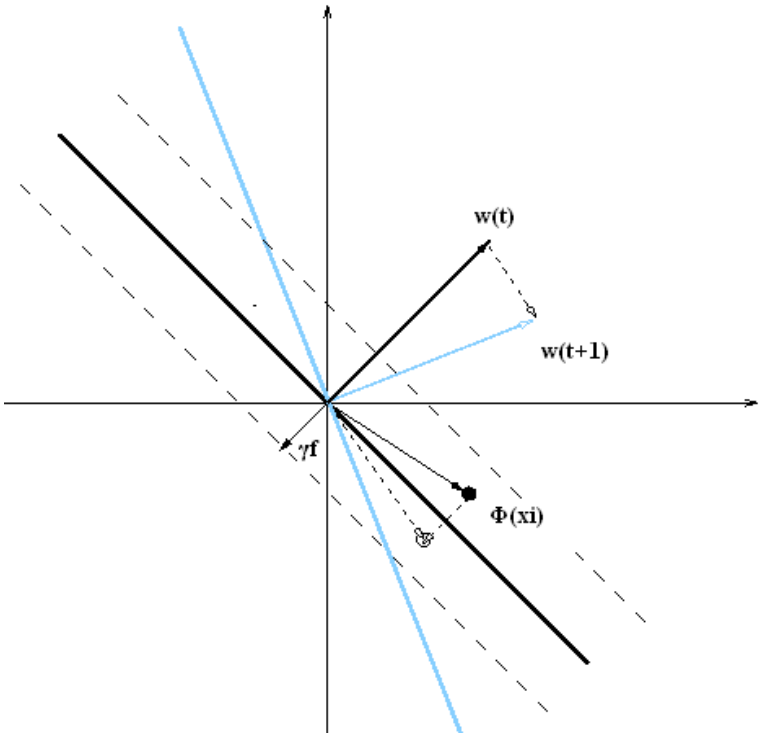


Figura 10: Interpretação geométrica da margem γ_f

Esta sugere que antes de cada correção do vetor w seja feito um escalonamento no valor do mesmo, proporcional ao valor da margem fixa projetada na sua respectiva direção. Neste sentido, pode-se afirmar que a forma de regularização empregada para o controle da norma consiste em uma espécie de decaimento no valor dos pesos. Também, devido ao emprego do conceito de margem geométrica percebe-se que esta regularização já esta implícita na própria função de perda a ser minimizada.

4.5 *Perceptron Multiclasse*

Como a maioria das outras técnicas para a formação classificadores lineares, o *Perceptron* generaliza naturalmente a classificação multiclasse. Aqui, a entrada x e a saída y são elaborados a partir de conjuntos arbitrários. A função de representação característica $f(x, y)$ mapeia cada possível par de entrada/saída para um vetor característica de valores reais de dimensão finita. Como antes, o vetor de característica é multiplicado por um vetor w de peso, mas agora a pontuação resultante é usada para escolher entre várias saídas possíveis:

$$\hat{y} = \text{argMax}_{y \in Y}\{f(x, y) \cdot w\}$$

Onde Y é o conjunto de todas as saídas possíveis.

O aprendizado novamente itera sobre os exemplos, prevendo uma saída para cada um, deixando o peso inalterado quando a saída prevista corresponde ao alvo, e alterá-los quando isso não acontece. A atualização é a seguinte:

$$w_{t+1} = w_t + f(x, y) - f(x, \hat{y})$$

Esta formulação multiclasse se reduz ao *Perceptron* original quando x é um vetor de valores reais, y é escolhido entre $(0, 1)$, e $F(x, y) = yx$.

Para certos problemas, As representação de entrada/saída e suas características podem ser escolhidas de modo que o $\text{argMax}_{y \in Y}\{f(x, y) \cdot w\}$ pode ser encontrado de forma eficiente, mesmo se y for escolhido de um conjunto muito grande ou infinito (COLLINS, 2002).

5 *Modelo de Predição de Dados Estruturados*

Nesse capítulo são vistas as formulações matemáticas envolvidas na criação dos modelos estruturados e sua predição.

5.1 Modelos Estruturados

Os tipos de dados estruturados possuem um conjunto de tipos de dados definidos como elementares de tal modo que exista uma relação estrutural entre seus valores.

Um **modelo de predição estruturado** é aquele cuja saída não é um valor discreto y_j, mas sim um conjunto estruturado de dados ou valores $y = (y_1, ..., y_L)$, tendo um número de variáveis fixo L (TASKAR, 2004). O termo estruturado se refere a existência de um conjunto de restrições e correlações que moldam um espaço de saída $Y \subseteq Y_1 \times ... \times Y_L$ definido como um subconjunto do produto de espaços de saída de variáveis elementares. Num exemplo simples tem-se que cada y_j é alguma letra do alfabeto, Y_j é o alfabeto, e Y é o dicionário. Podendo-se ainda definir regras de combinações entre os elementos, tais como, antes de p ou b nunca deve vir a letra n. Um espaço, como o citado por exemplo, com suas restrições e correlações recebe o nome de **espaço estruturado**.

Problemas complexos podem envolver saídas como árvores, grafos ou *strings*, e podem vir a depender também de cada tipo de entrada, neste caso $Y = Y(x)$, onde x é algum dado ou conjunto de dados.

A principal questão, segundo (BAKIR et al., 2006), quando se trata de predizer dados estruturados está no porquê de se fazer predições de um conjunto estruturado de valores ao invés de simplesmente predizer cada saída individualmente. A resposta está justamente em que ao se adotar uma saída estruturada, leva-se em conta as interdependências observadas na forma de restrições e correlações no conjunto estruturado de saída Y.

5.1.1 Modelos Lineares para Predição Estruturada

Na classe de modelo estruturado H, ao invés de definir uma saída única, se assume que uma apropriada associação $f : X \times Y \rightarrow \Re^m$ está disponível (BAKIR et al., 2006).

Dado uma função f, pode-se então definir H como o conjunto de funções lineares g, parametrizadas por um vetor de pesos w, como segue:

$$g(x, y) = \langle w, f(x, y) \rangle$$

A função f também pode ser indicada como a função de compatibilidade entra entradas e saídas. A função de compatibilidade implicitamente define um mapeamento G de entradas para saídas, da seguinte forma:

$$G(x) = argMax_{y \in Y}\{g(x, y)\}$$

Pode-se ver tal definição aplicada no exemplo abaixo de (TASKAR et al., 2005):

Considere a modelagem da tarefa de atribuição de revisores de artigos como um problema de correspondência de peso máximo bipartido. Onde o peso representa o grau de conhecimento do revisor em relação ao artigo em questão.

Em outras palavras, existem R revisores por artigo onde cada um assina um número A de artigos. Para cada artigo e revisor tem-se um valor $s_{i,j}$ indicando o nível de qualificação do revisor j para a avaliação do artigo k.

O objetivo é achar uma atribuição para os revisores de artigos que maximiza o peso total. A correspondência é representada por um conjunto binário de variáveis $y_{j,k}$ representando o valor 1 se o revisor j assinou o artigo k e 0 caso contrário. O valor de uma atribuição é o seguinte somatório: $s(y) = \sum_{j,k} s_{j,k} y_{j,k}$. O espaço de saída Y é o conjunto de correspondências bipartidas para um número dado de artigos A e revisores R.

O peso máximo, $argMax_{y \in Y} s(y)$ pode ser resolvido usando um algoritmo combinatorial da seguinte programação linear:

$Max \sum_{j,k}\{s_{j,k} \cdot y_{j,k}\}$

Sujeito a:

$\sum_j y_{j,k} = R$, $\sum_k y_{j,k} \leq A$, $0 \leq y_{j,k} \leq 1$

A solução deste problema é garantida por ter soluções de integral (0/1) para qualquer

valor da função $s(y)$, visto que P e R são inteiros (SCHRIJVER, 2003).

Porém, num problema mais complexo, que é levado em conta as palavras que aparecem no *site* do revisor e sua comparação com as palavras do artigo a ser revisado, há também a necessidade de aumentar o peso de certas palavras, dependendo do grau de relevância das mesmas.

Sendo $webpage(j)$ o conjunto de palavras que ocorrem na *homepage* do revisor j, e $resumo(k)$ as palavras que aparecem no resumo do artigo k. Então $x_{j,k}$ denota a intersecão do conjunto de palavras como $webpage(j) \bigcap resumo(k)$. Tem-se agora que $s_{j,k} = \sum_d w_d \Phi(palavra_d \in x_{j,k})$, onde Φ é uma função indicadora ou função característica, ou seja, indica se o elemento pertence ao subconjunto $palavra$, e w_d é o peso referente a essa palavra. Define-se então $f_d(x_{j,k}) = \Phi(palavra_d \in x_{j,k})$ e $f_d(x,y) = \sum_{j,k} y_{j,k} \Phi(palavra_d \in x_{j,k})$ como o número de vezes que a palavra d estava em ambos, tanto na *homepage* do escritor quanto no artigo, onde $y_{j,k}$ é determinado revisor j para algum artigo k.

Dado isso, pode-se representar o objeto $s(y)$ como uma combinação ponderada de um conjunto de características $w^T f(x,y)$, onde w é um conjunto de parâmetros ou pesos, e $f(x,y)$ é o conjunto de características. E seu modelo linear de predição estruturado H corresponde a seguinte família linear:

$$h_w(x) = argMax_{y \in Y}\{w^T f(x,y)\}$$

Agora será visto mais a fundo a formulação dessa predição estruturada juntamente com a definição de suas restrições.

5.2 Predição de Dados Estruturados

Segundo (TASKAR et al., 2005) de forma geral, nos problemas de predição estruturada, a entrada $x \in X$ é um objeto estruturado arbitrário e a saída é um vetor de valores $y = (y_1, ...y_L)$. O tamanho de L e a estrutura de y dependem deterministicamente da entrada x. No exemplo da correspondência bipartida citado anteriormente, o espaço de saída é definido por um número de artigos e revisores, respectivamente A e R. Denotando o espaço de saída para um x como $Y(x)$ e o espaço completo de saída como $Y = \bigcup_{x \in X} Y(x)$.

Pode-se definir o espaço de saída para uma estrutura exemplo x usando um conjunto de funções de restrição: $\Psi(x,y) : X \times Y \longmapsto \Re$

E como já visto, a classe dos modelos de predição estruturado H corresponde a se-

guinte família linear:

$$h_w(x) = argMax_{y \in Y}\{w^T f(x,y)\}$$

Onde a saída y está sujeita a alguma função $\Psi(x,y)$ restritiva e a função $f(x,y)$ é um vetor de funções na forma $f : X \times Y \longmapsto \Re^m$. Como exemplo pode-se ter como regra restritiva $Y(x) = \{y : \Psi(x,y) \leq 0\}$, onde $\Psi_d = (x,y) : X \times Y \rightarrow \Re^m$, ficando então:

$$h_w(x) = argMax_{y:\Psi(x,y)\leq 0}\{w^T f(x,y)\}$$

Esta formulação é bem geral, para muitas escolhas f e Ψ achar o y ótimo dado um x é intratável. Isto acarreta a busca por modelos onde o problema de otimização pode ser tratado em tempo polinomial. Isto inclui certo tipos de redes de Markov e gramáticas livre de contexto; bem como problemas de otimização convexa linear ou quadrática, com relação as duas últimas uma explicação resumida pode ser encontrada no Apêndice B.

O objetivo agora passa a ser estimar esse vetor w de modo que $h_w(x)$ retrate, mesmo que de forma aproximada, alguma saída desejada y pré-determinada. E para isso são necessários um **conjunto de treinamento**, um **conjunto de testes** e o ajuste de pesos do vetor w, onde a correção pode ser calculada com base na diferença entre a saída real gerada e a saída desejada, de forma muito semelhante ao aprendizado por correção de erros de uma rede neural, baseada no aprendizado supervisionado.

5.3 Formulação de Máxima Margem

Aqui é descrito uma abordagem para resolver o problema da determinação do vetor de pesos w. Tal problema pode ser visto como um problema de otimização convexa (abordado no Apêndice B).

Dado um conjunto de treinamento $S = \{(x(i), y(i)), i = 1, ..., m\}$ formado por uma coleção de pares, sendo cada par formado por uma amostra representada por um objeto estruturado $x(i)$ e uma solução desejada $y(i)$ deseja-se obter um vetor de parâmetros w tal que:

$$argMax_{y \in Y(i)}\{w^T.f(x(i),y)\} \approx y(i),\ \ i{=}1,\ldots,m \tag{5.1}$$

Sendo $Y(i)$ o espaço de todas as soluções ou exemplos possíveis dependente do objeto estruturado $x(i)$. Muito embora a cardinalidade de $Y(i)$ possa ser muito elevada, é possível a utilização de uma técnica de geração que resolve de forma eficiente a determinação do

falso-exemplo y.

O aprendizado do vetor de parâmetros w permite que a melhor solução encontrada para cada par da coleção seja exatamente a solução proposta no conjunto de treinamento.

Foi assumido que as características são completas e variadas o suficiente para satisfazer qualquer tipo de restrição, e como pode ser observado, tal caso é análogo aos casos de separabilidade da formulação das máquinas de vetores suportes de (VAPNIK, 1998), então pode-se expressar a solução ótima do vetor w para cada instância i como o problema de programação quadrática abaixo:

$Min \frac{1}{2}\|w\|^2$

Sujeito a:

$w^T.f_i(y(i)) \geq w^T f_i(y) + l_i(y),\ i = 1, \ldots, m\ \forall y \in Y(i)$

A dificuldade com a equação acima está no fato de se calcular cada $f_i(y)$ para depois compará-lo com todos os $f_i(y(i))$ para cada instância i. O número de comparações a serem efetuadas caem no caso exponencial. Este problema pode ser resolvido, pois se a minimização está sujeita a $w^T f_i(y(i)) \geq w^T f_i(y)$ para cada $f_i(y)$, então basta que $w^T f_i(y(i))$ seja maior que o máximo valor possível de $w^T f_i(y)$.

A abordagem mais utilizada na solução deste problema, relacionado à predição de dados estruturados, se baseia na generalização do princípio da formulação de máxima margem (TASKAR et al., 2005) empregado nas máquinas de vetores suportes (VAPNIK, 1998) e implica na solução do seguinte problema de programação quadrática:

$Min \frac{1}{2}\|w\|^2$

Sujeito a:

$w^T.f_i(y(i)) \geq Max_{y \in Y(i)}\{w^T f_i(y) + l_i(y)\},\ i = 1, \ldots, m$

Sendo $f_i(y) = f(x(i), y)$ e a função $l_i(y) = l(y(i), y)$ definida como uma função de perda que escalona o valor geométrico de margem requerido para o falso-exemplo y em relação à amostra $y(i)$.

A margem de uma amostra $(x(i), y(i))$ sobre qualquer outro exemplo $y \in Y(i)$ é interpretada como:

$$w^T.(f_i(y(i)) - Max_{y \in Y(i)}\{f_i(y)\})/||w||_2$$

Esta formulação provê um algoritmo em tempo polinomial para muitas variantes deste

problema. Em uma versão modificada pode-se considerar a utilização de um parâmetro de margem identificado por γ o qual deve ser maximizado com a restrição adicional de controle da norma do vetor de parâmetros w. Assim o problema de otimização convexa, sugerido em (TSOCHANTARIDIS et al., 2005), pode ser reescrito na forma:

$Max\ \gamma$

$\|w\| = 1$

Sujeito a:

$w^T.f_i(y(i)) - Max_{y \in Y(i)}\{w^T.f_i(y) + l_i(y)\} \geq \gamma,\ i = 1, \ldots, m$

Desta forma, a minimização da norma do vetor w ou a equivalente maximização da margem γ resulta na obtenção de uma solução de máxima margem que satisfaz o conjunto de restrições. Para os casos de não separabilidade pode-se admitir a introdução de variáveis de folga como utilizado nas máquinas de vetores suporte. É interessante observar que o problema: $Max_{y \in Y(i)}\{w^T.f_i(y) + l_i(y)\}$ tem precisamente a mesma forma do problema de predição para o qual se necessita aprender os parâmetros. Em geral, este problema é de complexidade polinomial e de fácil solução, tal como um problema de caminho mínimo sobre um grafo de estados.

Esta formulação é um problema de programação quadrática convexo em w desde que $Max_{y \in Y(i)}\{w^T.f_i(y) + l_i(y)\}$ seja convexo em w, ora, mas o máximo de uma função afim é uma função convexa (TASKAR et al., 2005). O próximo capítulo aborda as técnicas de solução necessárias para resolver o problema de programação quadrática convexa apresentado neste capítulo.

6 Técnicas de Solução

Os métodos de soluções abordados são o do subgradiente, que segundo (BERTSEKAS, 2003) pode ser implementado de forma simples, o *Perceptron estruturado* e o *Perceptron estruturado com Margem*, os dois últimos desenvolvidos neste trabalho. O ferramental necessário para a solução do problema de maximização, sob o ponto de vista da otimização pode ser encontrado nos Apêndice B e C.

6.1 O Método de Subgradiente

Os Métodos Subgradientes são pioneiros em otimização não diferenciável. Foram originalmente desenvolvidos por (BERTSEKAS, 1985), na União Soviética, nas décadas de 60 e 70. Esses métodos, também chamados de Métodos Gradientes Generalizados, são uma generalização dos Métodos Gradientes no qual o gradiente da função é substituído por um subgradiente para obter uma nova direção de busca. Possuem uma estrutura muito simples que não utiliza busca linear. O tamanho do passo pode ser fixado ou pode mudar com as iterações, porém dependendo do passo escolhido não se tem a garantia de convergência global (SOUZA, 2008).

Considere o problema de minimizar a função $f(w)$ com $w \in \mathbb{R}$. O processo de atualização básica do subgradiente segundo (BERTSEKAS, 1985) é a seguinte:

Seja $f : \mathbb{R}^n \to \mathbb{R}$ uma função convexa com domínio $\mathbb{R}^n$. O método usado na iteração do subgradiente é:

$$w^{(k+1)} = w^{(k)} - \alpha_k d^{(k)}$$

Onde $d^{(k)}$ denota o subgradiente de uma função f em $w^{(k)}$. Se f é diferenciável, o único subgradiente é o próprio gradiente do vetor ∇f. Pode acontecer de $d^{(k)}$ não ser uma direção de descida para f em $w^{(k)}$, ou seja, a direção oposta ao subgradiente pode não ser uma direção de descida, o que é um grande problema. Por isso, (BOYD; L.; MUTAPCIC, 2003) apresenta a proposta de manter uma lista f_{melhor} que controla o menor valor da

função objetivo encontrado até agora, mantendo somente os melhores valores da função a cada iteração, ou seja, mantendo o melhor $w^{(k)}$ encontrado:

$$\text{Descida: } f^{(k)}_{\text{melhor}} = Min\{f^{(k-1)}_{\text{melhor}}, f(w^{(k)})\}$$

É fundamental uma escolha adequada do tamanho do passo $\alpha_k > 0$ em cada iteração. Uma das maiores dificuldades está na escolha do tamanho do passo α_k (NEMHAUSER; WOLSEY, 1999). Se os passos forem muito pequenos, w^k se aproximará muito lentamente do ponto ótimo. Por outro lado, segundo (LIMA, 2007), se forem excessivamente largos, o método poderá oscilar desnecessariamente em torno da solução.

Assim, para garantir a convergência global do método, pode-se escolher o tamanho do passo α_k , chamado de Passo da Série Divergente (RODRIGUES, 1994), de forma que $\lim_{k\to\infty} \alpha_k = 0$ e $\sum_{k=1}^{\infty} \alpha_k = +\infty$, sua demonstração pode ser encontrada em (LEMARECHAL, 1989).

Outra dificuldade dos Métodos Subgradientes é estabelecer um critério de parada uma vez que os subgradientes são encontrados arbitrariamente e por isso, não contem informação sobre a condição de otimalidade. Testes práticos devem ser aplicados observando a especificidade do problema. Um critério de parada para os Métodos Subgradientes pode ser dado pelo número máximo de iterações atingido ou pela condição $w^{k+1} \approx w^k$, ou ainda se $f(w^{k+1}) \approx f(w^k)$ (SOUZA, 2008).

O vetor $d^{(k)}$ é a direção do subgradiente de $f(w)$ para w^k. A cada iteração k é dado um passo do ponto corrente w^k em direção oposta ao subgradiente.

O algoritmo básico do método do subgradiente funciona da seguinte maneira:

$w = w^0$

$k = 1$

Enquanto critério de parada não for satisfeito faça

Resolver (d^k)

$w^{k+1} = w^k - \alpha_k(d^k)$

$k \leftarrow k + 1$

Fim-enquanto

6.2 *Perceptron estruturado*

Utilizando os conceitos apresentados no Capítulo 5 sobre dados estruturados juntamente com os princípios da formulação *Perceptron* foi elaborado o *Perceptron Estruturado* (COELHO; NETO; BORGES, 2009).

Espaços estruturados de saídas são tipicamente aprendidos usando extensões de algoritmos de classificação para simples estruturas gráficas nos casos que a busca e os parâmetros de estimação são exatos. Porém, em muitos problemas complexos, essa exatidão é inviável.

Segundo (DAUMé III; MARCU, 2005) o coração dos algoritmos de aprendizado com saídas estruturadas é a seguinte formulação:

$$\widehat{y} = argMax_{y \in Y} f(x, y; w)$$

Onde $\widehat{y}$ é alguma saída estruturada do conjunto de todas as estruturas possíveis Y e $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ uma função convexa, com domínio $\mathbb{R}^n$.

Dado um conjunto de treinamento $S = \{(x(i), y(i)), i = 1, ..., m\}$ formado por uma coleção de pares, sendo cada par formado por uma amostra representada por um objeto estruturado $x(i)$ e uma solução estruturada desejada $y(i)$ deseja-se obter um vetor de parâmetros w tal que:

$$\text{argMax}_{\ y \in Y}\{w^t f(x(i), y)\} \approx y(i)$$

Em vez de achar o $y(i)$ ótimo para a estrutura $x(i)$, usa-se a base do *Perceptron* e compara-se todas as outras saídas do conjunto de treinamento com a saída atual, obtendo, como demonstrado posteriormente nos resultados experimentais, uma boa aproximação, chamado também de y^*, apesar de não ser o ótimo global. Dessa forma tem-se que a complexidade é linear, relacionada a quantidade de elementos y.

Tem-se então que cada estrutura $x(i)$ tem sua saída desejada $y(i)$, e que y^* é a melhor solução analisada atual. O processo de atualização básica do *Perceptron Estruturado*, que é uma variante do algoritmo primal do *Perceptron*, é a seguinte:

Se $w^T.(f(x(i), y(i)) - f(x(i), y^*)) > 0$

Correção: $w^{(k+1)} = w^{(k)} - \eta d_i^{(k)}$

Onde $d^{(k)_i} = f(x(i), y(i)) - f(x(i), y^*)$

Isto pode ser resumido da seguinte forma: sendo η uma taxa de aprendizado constante $(0 < \eta < 1)$ e o melhor y determinado através da comparação direta de todo conjunto Y. Assim, y representa sempre a melhor alternativa, desconsiderando o $y(i)$ correto. Caso a diferença não seja positiva certifica-se que o y^* escolhido possui um valor inferior a melhor alternativa, não sendo necessário neste caso a correção do vetor w. O critério de parada é a correlação $(x(i), y(i))$ sendo satisfeita para todo o conjunto de treinamento.

A regra de correção obtida para o algoritmo *Perceptron Estruturado* pode ser facilmente derivada seguindo o raciocínio descrito a seguir. Seja o vetor diferença definido para cada mapa da coleção na forma:

$$d_i^{(k)} = f(x(i), y(i)) - f(x(i), y^*) \tag{6.1}$$

A condição de viabilidade do algoritmo *Perceptron* é dada por:

$$-w^T.d_i^{(k)} \geq 0,\ i = 1, ..., m \tag{6.2}$$

Desta forma, a função de perda relacionada à estratégia de minimização de custos será dada por:

$$J(w) = \sum_i Max\{0, w^T.d_i^{(k)}\} \tag{6.3}$$

Esta função deve ser minimizada. Portanto, pode-se definir o gradiente local relativo a *i-ésima* amostra como:

$$\nabla_w J(w) = d_i^{(k)} = f(x(i), y(i)) - f(x(i), y^*) \tag{6.4}$$

Assim, caso ocorra um erro relacionada a *i-ésima* amostra, ou seja:

$$-w^T.d_i^{(k)} < 0\ ou\ w^T.(f(x(i), y(i)) - f(x(i), y^*)) > 0, \tag{6.5}$$

utiliza-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w - \eta.(f(x(i), y(i)) - f(x(i), y^*)),\ 0 < \eta \leq 1 \tag{6.6}$$

De outra forma, para um problema de maximização de recompensas tem-se:

$$d_i^{(k)} = Max_{y \in Y(i)}\{(f(x(i), y) - (f(x(i), y(i))\} = f(x(i), y^*)) - (f(x(i), y(i)) \tag{6.7}$$

A condição de viabilidade do algoritmo *Perceptron* permanece a mesma sendo dada

por:

$$-w^T.d_i^{(k)} \geq 0,\ i = 1, ..., m \tag{6.8}$$

Assim tem-se a mesma função de perda apresentada em (6.4) definindo o gradiente local:

$$\nabla_w J(w) = d_i^{(k)} = f(x(i), y^*) - f(x(i), y(i)) \tag{6.9}$$

Portanto, caso ocorra um erro relacionada a *i-ésima* amostra, ou seja:

$$-w^T.d_i^{(k)} < 0\ ou\ w^T.(f(x(i), y^*) - f(x(i), y(i))) > 0, \tag{6.10}$$

utiliza-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w - \eta.(f(x(i), y^*) - f(x(i), y(i))),\ 0 < \eta \leq 1 \tag{6.11}$$

6.3 *Perceptron Estruturado com Margem*

Através da união da teoria do *Perceptron com Margem* e do *Perceptron Estruturado* obtemos a regra de correção para o algoritmo *Perceptron Estruturado com Margem*. Ela pode ser derivada a partir do seguinte raciocínio. Seja o mesmo vetor diferença definido para cada mapa da coleção na forma:

$$d_i^{(k)} = f(x(i), y(i)) - f(x(i), y^*) \tag{6.12}$$

A condição de viabilidade do algoritmo *Perceptron* agora é dada por:

$$-w^T.d_i^{(k)} \geq \gamma.||w||_2,\ i = 1, ..., m \tag{6.13}$$

Desta forma, a função de perda relacionada à estratégia de minimização de custos será dada por:

$$J(w) = \sum_i Max\{0, \gamma - \frac{w^T.d_i^{(k)}}{||w||_2}\} \tag{6.14}$$

Entretanto, segundo (KIVINEN; SMOLA; WILLIAMSON, 2002), minimizar este tipo de função J pode levar ao *overfitting* (funções complexas que se encaixam bem no treinamento de dados mas não são boas generalizações para novos dados). Uma maneira de resolver isso é penalizar J e minimizá-lo de maneira relaxada. Portanto, pode-se definir

o gradiente local relativo a *i-ésima* amostra como:

$$\nabla_w J(w) = \frac{\gamma.w^T}{||w||_2} - \frac{d_i^{(k)}}{||w||_2} = \frac{\gamma.w^T}{||w||_2} - \frac{f(x(i), y(i)) - f(x(i), y^*)}{||w||_2} \tag{6.15}$$

Assim, caso ocorra um erro relacionada a *i-ésima* amostra, ou seja:

$$-w^T.d_i^{(k)} < \gamma.||w||_2 \; ou \; w^T.(f(x(i), y(i)) - f(x(i), y^*)) > \gamma.||w||_2, \tag{6.16}$$

utiliza-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w.(1 - \frac{\eta.\gamma}{||w||_2}) + \eta(f(x(i), y(i)) - f(x(i), y^*)), \; 0 < \eta \leq 1 \tag{6.17}$$

De outra forma, para um problema de maximização de recompensas tem-se:

$$d_i^{(k)} = Max_{y \in Y(i)}\{\frac{(f(x(i), y) - (f(x(i), y(i))}{||w||_2} - \gamma\} = f(x(i), y^*)) - (f(x(i), y(i)) - \gamma \tag{6.18}$$

A condição de viabilidade do algoritmo *Perceptron* permanece a mesma sendo dada por:

$$-w^T.d_i^{(k)} \geq \gamma.||w||_2, \; i = 1, ..., m \tag{6.19}$$

Assim tem-se a mesma função de perda apresentada em (6.15) definindo o gradiente local:

$$\nabla_w J(w) = \frac{\gamma.w^T}{||w||_2} - d_i^{(k)} = \frac{\gamma.w^T}{||w||_2} - f(x(i), y^*) - f(x(i), y(i)) \tag{6.20}$$

Portanto, caso ocorra um erro relacionada a *i-ésima* amostra, ou seja:

$$-w^T.d_i^{(k)} < \gamma.||w||_2 \; ou \; w^T.(f(x(i), y^*) - f(x(i), y(i))) > \gamma.||w||_2, \tag{6.21}$$

utiliza-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w.(1 - \frac{\eta.\gamma}{||w||_2}) - \eta(f(x(i), y^*) - f(x(i), y(i))), \; 0 < \eta \leq 1 \tag{6.22}$$

No próximo capítulo será abordado a predição de dados estruturados utilizando a formulação de máxima margem já inserido dentro do problema de predição de custos.

7 *Problema de Predição de Custos*

O problema de predição de custos foi formulado no contexto de uma coleção de mapas e caminhos tendo como objetivo a maximização de recompensas. Neste problema cada mapa é representado por um grafo de estados formando um conjunto de pares estados-ações, e cada estado dos mapa é caracterizado por um conjunto de características. Duas abordagens para resolução deste problema são discutidas e explicadas, o método do subgradiente e o método do *Perceptron Estruturado*.

7.1 Equacionamento do Problema de Predição de Custos

Primeiramente deve-se observar como a formulação nó-arco foi aplicada ao problema de predição de custo, a matriz que representa tal formulação é a matriz μ. Cada caminho é representado por um vetor de frequências em uma matrizμ que indica a ocorrência ou não das diversas ações ou arcos para cada possível par estado-ação ou transição definido pelo mapa. Cada mapa é representado por uma matriz F de características, distribuída para todo conjunto de pares estado-ações. Na Figura 11, pode-se ver uma representação gráfica dessas matrizes.

Assim o produto $Fi.\mu$ representa a quantidade de cada característica presente no i-ésimo mapa em função da escolha do caminho representado por μ. Finalmente o produto $w^T.F_i\mu$ representa a recompensa total do caminho μ no *i-ésimo* mapa computado em função dos valores apresentados pelo vetor de parâmetros w. O equacionamento deste problema foi apresentado por (RATLIFF; BAGNELL; ZINKEVICH, 2006), tendo a forma:

$Min\ \frac{1}{2}\|w\|^2$

$Sujeito\,a$:

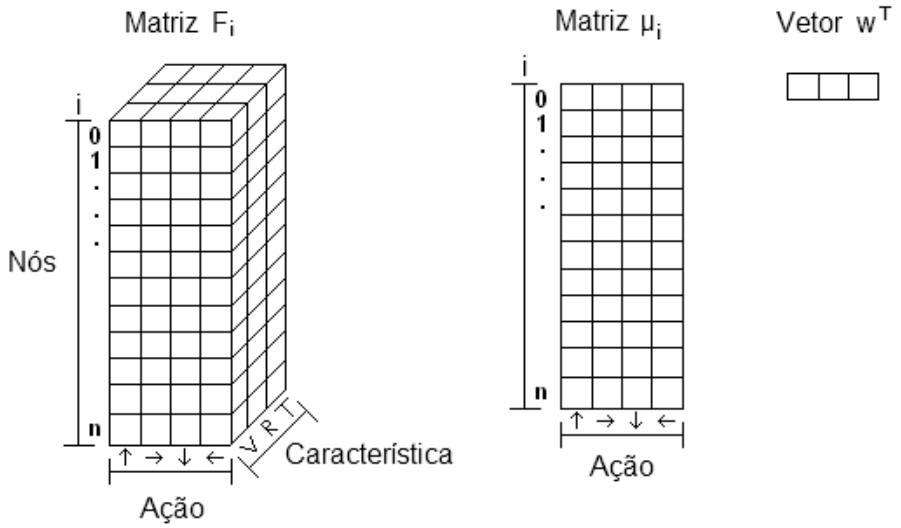


Figura 11: Representação gráfica das matrizes

$w^T.F_i\mu_i \geq Max_{\mu \in G(i)}\{w^T F_i\mu + l_i^T.\mu\},\ i = 1, \ldots, m$

Sendo $G(i)$ a representação de todas as escolhas possíveis de caminhos referentes ao *i-ésimo* mapa.

Sem perda de generalidade optou-se neste trabalho pelo problema de minimização de custos. Assim, o equacionamento toma a forma:

$Min\ \frac{1}{2}\|w\|^2$

$Sujeito\,a:$

$w^T.F_i\mu_i \leq Min_{\mu \in G(i)}\{w^T F_i\mu + l_i^T.\mu\},\ i = 1, \ldots, m$

7.2 Método do Subgradiente Aplicado na Predição de Custos

Para a solução do problema de maximização de recompensas apresentado em (RATLIFF; BAGNELL; ZINKEVICH, 2006) propõem a minimização de uma função objetiva não diferenciável mas convexa, obtida da relaxação do problema de programação quadrática, com o uso de uma técnica de sub-gradiente. Assim deve-se minimizar a função:

$$L(w) = \frac{1}{2}||w||^2 + C.\sum_i Max_{\mu \in G(i)}\{w^T.F_i\mu + l_i^T.\mu\} - w^T.F_i\mu_i \qquad (7.1)$$

Esta função possui como sub-gradiente:

$$g = w + C.(\sum_i F_i\mu^* - F_i\mu_i) \tag{7.2}$$

Computado em função da determinação da política ótima μ^* relativa ao caminho ótimo da matriz de custos de cada mapa de entrada obtida da equação: $w^T.F_i\mu + l_i^T$. Para tanto se utiliza o algoritmo A* que apresenta complexidade quadrática em relação à quantidade de estados ou células.

Assim o vetor w fica atualizado na forma:

$$w \leftarrow w - \eta.g, \;\; ou \;\; w \leftarrow w.(1-\eta) - \eta.(C.(\sum_i F_i\mu^* - F_i\mu_i)) \tag{7.3}$$

Para uma taxa η de aprendizado reduzida de forma gradativa de acordo com o número de iterações e uma parâmetro de regularização positivo C.

7.3 Método do *Perceptron Estuturado* Aplicado na Predição de Custos

Se não for imposta à condição de maximização da margem pode-se reformular o problema de predição com minimização de custos como um problema de viabilidade de um sistema de inequações na forma:

Encontrar um vetor w viável, tal que:

$$w^T.F_i\mu_i \leq Min_{\mu \in G(i)}\{w^T.F_i\mu + l_i^T.\mu\}, \;\; i = 1, \ldots, m, \tag{7.4}$$

Ou, de modo alternativo:

$$w^T.F_i\mu_i - Min_{\mu \in G(i)}\{w^T.F_i\mu + l_i^T.\mu\} \leq 0, \;\; i = 1, ..., m \tag{7.5}$$

Para a solução deste problema pode-se empregar uma variante do algoritmo primal do *Perceptron* o qual foi denominado *Perceptron Estruturado* com a seguinte regra de correção:

Para cada mapa representado pelo índice $i = 1, \ldots, m$:

$$Se(w^T.F_i\mu_i - Min_{\mu \in G(i)}\{w^T.F_i\mu + l_i^T.\mu\}) > 0 \tag{7.6}$$

Ou seja, se uma inequação não está sendo satisfeita, então atualizar o vetor w, sendo $0 < \eta \leq 1$, na forma:

$$w \leftarrow w - \eta.(F_i\mu_i - F_i\mu^*) \tag{7.7}$$

Sendo η uma taxa de aprendizado constante e o vetor μ^* determinado através da comparação direta de todo conjunto de caminhos existentes propostos no conjunto de treinamento S, ou seja:

$$\mu^* = argMin_{\mu \in S \ e \ \mu \neq y(i)}\{w^T.F_i\mu + l_i^T.\mu\} \tag{7.8}$$

Assim, o vetor μ^* representa sempre a melhor alternativa de caminho desconsiderando o caminho correto ou escolhido $y(i)$ associado a cada mapa fornecido.. Caso a diferença não seja positiva certifica-se que o caminho escolhido tem um custo inferior a melhor alternativa não sendo neste caso necessária a correção do vetor de parâmetros w. Observa-se que neste caso a função de perda $l_i()$ não escalona o valor geométrico da margem mas atua como um fator de penalidade com o objetivo de evitar a ocorrência de custos negativos nos mapas. Isto é necessário devido à utilização do algoritmo A* para a determinação dos caminhos ótimos relacionados ao conjunto de teste.

A vantagem de se utilizar a escolha da melhor alternativa de caminho entre um conjunto de caminhos existentes e não sobre todos os caminhos possíveis está na redução do esforço computacional. Neste novo processo a complexidade é linear e relacionada à quantidade de mapas existentes no conjunto de treinamento, lembrando que os custos dos caminhos em um mapa são resultados do produto interno do vetor w pelos vetores pré-computado de características dos caminhos representados por $F_i\mu$.

A regra de correção obtida para o algoritmo *Perceptron Estruturado* pode ser facilmente derivada seguindo o raciocínio descrito a seguir. Seja o vetor diferença definido para cada mapa da coleção na forma:

$$\delta_i = F_i\mu_i - F_i\mu^* \tag{7.9}$$

A condição de viabilidade do algoritmo *Perceptron* é dada por:

$$-w^T.\delta_i \geq 0, \ i = 1, ..., m. \tag{7.10}$$

Desta forma, a função de perda relacionada à estratégia de minimização de custos

será dada por:

$$J(w) = \sum_i Max\{0, w^T.\delta_i\} \tag{7.11}$$

Esta função deve ser minimizada. Portanto, pode-se definir o gradiente local relativo a *i-ésima* amostra como:

$$\nabla_w J(w) = \delta_i = F_i\mu_i - F_i\mu^* \tag{7.12}$$

Assim, caso ocorra um erro relacionada a *i-ésima* amostra, ou seja:

$$-w^T.\delta_i < 0 \; ou \; w^T.(F_i\mu_i - Fi\mu^*) > 0 \tag{7.13}$$

Utiliza-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w - \eta.(Fi\mu_i - Fi\mu^*), \; 0 < \eta \leq 1, \tag{7.14}$$

equivalente, portanto a equação de correção proposta em (7.7).

De outra forma, para um problema de maximização de recompensas tem-se:

$$\delta_i = Max_{\mu \in G(i)}(F_i\mu + l_i\mu) - F_i\mu_i = F_i\mu^* - F_i\mu_i \tag{7.15}$$

A condição de viabilidade do algoritmo *Perceptron* permanece a mesma sendo dada por:

$$-w^T.\mu_i \geq 0, \; i = 1, ..., m \tag{7.16}$$

Assim tem-se a mesma função de perda apresentada em (7.11) definindo o gradiente local:

$$\nabla_w J(w) = \delta_i = F_i\mu^* - F_i\mu_i \tag{7.17}$$

Portanto, caso ocorra um erro relacionada a i-ésima amostra, ou seja:

$$-w^T.\delta_i < 0 \; ou \; w^T.(F_i\mu^* - F_i\mu_i) > 0 \tag{7.18}$$

Utiliza-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w - \eta.(F_i\mu^* - F_i\mu_i), \; 0 < \eta \leq 1 \tag{7.19}$$

A equação (7.13), relacionada ao erro da amostra, pode ser ajustada da seguinte maneira: $w^T.F_i\mu_i > w^T.Fi\mu^*$). E sua interpretação é a seguinte: se o custo do caminho escolhido pelo especialista for maior do que o custo dos outros caminhos do conjunto de

treinamento, então deve-se corrigir o vetor w.

É interessante observar a similaridade entre a equação (7.19) e a equação de correção do algoritmo MMP representada em (7.3). A diferença principal está relacionada à redução dos valores das componentes do vetor w e consequentemente no valor da sua norma no sentido de atender ao objetivo de minimização da função apresentada em (7.1).

7.4 Método do *Perceptron Estruturado com Margem* Aplicado na Predição de Custos

Agora tem-se a inclusão da margem na formulação do problema de encontrar um vetor w viável:

$$w^T.F_i\mu_i \leq Min_{\mu \in G(i)}\{w^T.F_i\mu + l_i^T.\mu\} + \gamma,\ i = 1, \ldots, m, \tag{7.20}$$

Ou, de modo alternativo:

$$w^T.F_i\mu_i - Min_{\mu \in G(i)}\{w^T.F_i\mu + l_i^T.\mu\} \leq \gamma,\ i = 1, \ldots, m, \tag{7.21}$$

A estrutura dos termos feitas nesta seção aplicadas ao *Perceptron Estruturado com Margem* são as mesmas da seção anterior, portanto aqui é explicado diretamente como se obtém a regra de correção. Ela pode ser facilmente derivada seguindo o raciocínio descrito a seguir. Seja o vetor diferença definido para cada mapa da coleção na forma:

$$\delta_i = F_i\mu_i - F_i\mu^* \tag{7.22}$$

A condição de viabilidade do algoritmo é dada por:

$$-w^T.\delta_i \geq \gamma.||w||_2,\ i = 1, ..., m. \tag{7.23}$$

Desta forma, a função de perda relacionada à estratégia de minimização de custos será dada por:

$$J(w) = \sum_i Max\{0, \gamma - w^T.\delta_i\} \tag{7.24}$$

Esta função deve ser minimizada. Assim, caso ocorra um erro relacionada a *i-ésima* amostra, ou seja:

$$-w^T.\delta_i < \gamma.||w||_2 \ ou\ w^T.(F_i\mu_i - Fi\mu^*) > \gamma.||w||_2 \tag{7.25}$$

Utiliza-se a seguinte regra de correção para o vetor de custos. Assim como explicado na seção *Perceptron Estruturado com Margem*, (KIVINEN; SMOLA; WILLIAMSON, 2002) aplica uma penalidade em J e a minimiza, desta forma obtém-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w.(1 - \frac{\eta.\gamma}{||w||_2}) - \eta(Fi\mu_i - Fi\mu^*),\ 0 < \eta \leq 1 \tag{7.26}$$

De outra forma, para um problema de maximização de recompensas tem-se:

$$\delta_i = Max_{\mu \in G(i)}(F_i\mu + l_i\mu) - F_i\mu_i = F_i\mu^* - F_i\mu_i \tag{7.27}$$

A condição de viabilidade do algoritmo é dada por:

$$-w^T.\mu_i \geq \gamma.||w||_2,\ i = 1, ..., m \tag{7.28}$$

Caso ocorra um erro relacionada a i-ésima amostra, ou seja:

$$-w^T.\delta_i < \gamma.||w||_2\ ou\ w^T.(F_i\mu^* - F_i\mu_i) > \gamma.||w||_2 \tag{7.29}$$

Utiliza-se a seguinte regra de correção para o vetor de custos:

$$w \leftarrow w.(1 - \frac{\eta.\gamma}{||w||_2}) - \eta(F_i\mu^* - F_i\mu_i),\ 0 < \eta \leq 1 \tag{7.30}$$

7.5 Algoritmo Incremental

Após a primeira execução do algoritmo *Perceptron Estruturado com Margem* há uma grande possibilidade da margem de parada não ser a máxima, então o que pode ser feito é incrementar a margem a cada execução do algoritmo, retendo os valores do vetor de custos w calculados e inicializando uma nova iteração com um valor de margem acima da última testada, no intuito de maximizá-la.

Nesta busca pela máxima margem dobra-se seu valor a cada execução do algoritmo *Perceptron Estruturado com Margem*. Quando a solução torna-se inviável, tem-se uma margem atual que não determina corretamente os caminhos para os respectivos mapas em um determinado limite de iterações. Utiliza-se então um processo semelhante a uma busca binária como segue.

O algoritmo calcula a média da margem anterior, que e viável, com a margem incre-

mentada. Se a média obtida for uma margem viável tem-se um novo limite inferior. Do contrário tem-se um novo limite superior.

O número de vezes que o algoritmo *Perceptron Estruturado com Margem* será executado depende somente do grau de precisão que pretende-se obter na margem. Deve-se ter cuidado para o algoritmo ter sempre armazenado a última margem viável.

Usando uma outra abordagem, ao invés do cálculo das médias pode-se usar um passo fixo Δ de incremento. A equação então adquire a seguinte forma:

$$\gamma \leftarrow \gamma_m + \Delta \tag{7.31}$$

Sendo γ_m a margem de menor valor, ou seja, $\gamma_m = Min_i\{\gamma_i\}$, quando comparados todos as mapas dois a dois. Tem-se que o γ_i viável para todos os mapas quando analisados em conjunto é justamente o γ_m, caso contrário haveria um $\gamma_i < \gamma_m$, tornando γ_m inviável justamente na comparação dos dois mapas que geraram esse γ_i. Então podemos reescrever a equação acima simplesmente como:

$$\gamma \leftarrow \gamma + \Delta \tag{7.32}$$

Nos exemplos do algoritmo *Perceptron Estruturado com Margem* do próximo capítulo foram utilizados os conceitos de margem incremental aqui expostos, o tipo de incremento que apresentou os melhores resultados foi o incremento através da média das margens, devido a isso o mesmo foi utilizado na comparação com os outros algoritmos.

8 Resultados Experimentais

8.1 Conceituação do Problema

Para validar os conceitos e algoritmos desenvolvidos neste trabalho optou-se pela solução de um problema prático de planejamento relacionado à determinação de caminhos em um mapa quadriculado. O problema foi definido como um problema de minimização de custos ou de obtenção do caminho mínimo. Os mapas foram simulados com o uso de matrizes 5x5 e 10x10. Cada componente de uma matriz define uma célula ou estado contendo um tipo de terreno. Para a realização de movimentos foram permitidas quatro ações possíveis relacionadas, respectivamente, as direções: norte, leste, sul e oeste. Também, determinou-se uma célula de origem e uma célula de destino para todos os caminhos a fim de conseguir uma melhor visualização e comparação dos resultados.

No primeiro exemplo, foram utilizados seis mapas 5x5 para o conjunto de treinamento. Já o segundo exemplo foi realizado com quatro mapas 10x10.Cada mapa contém as características do terreno e o melhor caminho dado pelo especialista do domínio. Para o conjunto de teste, no exemplo 1, foram utilizados doze mapas, já no exemplo 2, foram utilizados também quatro mapas, todos eles contendo somente as características dos terrenos.

O melhor caminho de cada mapa do conjunto de teste foi determinado com a utilização do algoritmo *Q-learning* . A matriz de custos de cada mapa foi obtida com a utilização dos vetores de custos das características computados pelos dois algoritmos de aprendizado implementados. Nestes vetores a primeira componente se refere ao custo da rocha, a segunda ao custo da vegetação e a terceira se refere ao custo da trilha.

Nos mapas pode-se ter a presença ou não de cada característica bem como a sua intensidade. Desta forma pode-se combinar a utilização de características na definição de um tipo de terreno de cada célula. Como exemplo pode-se mencionar a existência de um terreno contendo um pouco de vegetação, um pouco de rocha e rastros de uma antiga

trilha. Para uma melhor visualização e simplificação dos resultados optou-se somente pela presença ou não de cada característica e suas combinações possíveis, definindo assim oito tipos diferentes de terreno de acordo com a cardinalidade do conjunto potência. As características e suas combinações utilizadas estão representadas na Figura 12:

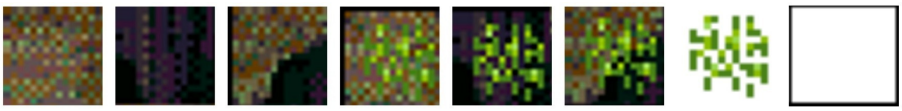


Figura 12: No primeiro quadro tem-se somente trilha; no segundo, somente rocha; no terceiro: trilha e rocha; no quarto: trilha e vegetação; no quinto: rocha e vegetação; no sexto: trilha, rocha e vegetação (trilha abandonada); no sétimo, somente vegetação e, finalmente, no oitavo tem-se a ausência de características.

8.2 Resultados do Conjunto de Treinamento

8.2.1 Exemplo 1

A Figura 13 apresenta os mapas 5x5 utilizados no conjunto de treinamento incluindo os caminhos traçados pelo especialista. Pode-se observar nesta figura que todos os caminhos possuem a célula 1 correspondente a entrada (1,1) da matriz como origem e a célula 22 correspondente a entrada (5,2) da matriz como destino. Pelos resultados alcançados constatou-se que ambos os algoritmos convergiram para os mesmos caminhos definidos pela estratégia do especialista, indicando desta forma que o aprendizado foi bem sucedido e as matrizes de custos obtidas refletem a estratégia de planejamento adotada pelo mesmo.

Como parâmetros do algoritmo MMP foram utilizados: taxa de aprendizado 0.5, valor de perda 2 e parâmetro de regularização 1. O vetor w foi inicializado com $[0.1, 0.1, 0.1]$. O algoritmo MMP necessitou de 17 iterações para convergir com tempo de execução aproximado de 0.031 segundos. Também foi estipulado uma máximo de 1000 iterações a fim de analisar o efeito sobre a maximização da margem, ou seja, como o algoritmo e sua margem se comportam tendo um alto número de iterações fixos. Seu tempo de execução é 0.402 segundos.

Como parâmetros do algoritmo *Perceptron Estruturado* foram utilizados: taxa de aprendizado 0.2, valor de penalidade 2. O vetor w também foi inicializado com $[0.1, 0.1, 0.1]$. O algoritmo necessitou de somente 4 iterações para convergir com tempo de execução aproximado de 0.015 segundos.

Os parâmetros do algoritmo *Perceptron Estruturado com Margem* foram os mesmos

do *Perceptron Estruturado* e a separação das classes se dá de maneira idêntica, com mesmo número de iterações e tempo. Foi também estipulado um máximo de 1000 iterações para analisar a maximização da margem e o tempo de execução foi de 0.125 segundos.

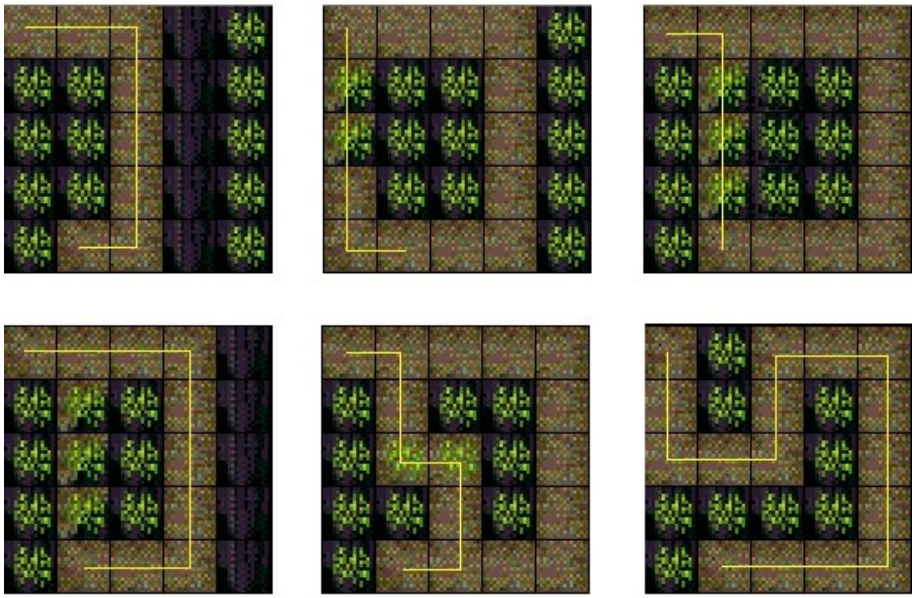


Figura 13: Mapas de treinamento 5x5 com seus respectivos caminhos traçados pelo especialista do domínio.

Na análise da Figura 13 pode-se concluir que, segundo o especialista, geralmente é melhor escolher um caminho que segue uma trilha sem vegetação e sem rocha. Entretanto, caso o caminho requeira uma trilha de maior comprimento o especialista poderá sugerir que o caminho passe por uma trilha com vegetação, com rocha ou com vegetação e rocha. Pode-se perceber também que o especialista evita fortemente as células contendo vegetação e rocha, mas sem a existência de trilha.

É importante ressaltar que os resultados obtidos atendem a um processo de escolha relacionado à técnica de planejamento por máxima margem. Isto indica que a escolha do especialista reflete também o melhor caminho ou aquele de menor custo quando comparado a todas alternativas presentes no conjunto de treinamento considerando as matrizes de custos aprendidas pelos algoritmos. Ou seja, pode-se citar o fato de que, para cada mapa, nenhuma escolha alternativa dos caminhos apresentados possuirá menor custo que o caminho sugerido pelo especialista para o mesmo. Para demonstrar esta propriedade do planejamento foram elaboradas 4 tabelas (Tabelas 1, 2, 3 e 4) que mostram os valores de custos geométricos para todos os mapas e caminhos do conjunto de treinamento para cada algoritmo. Estes valores são obtidos dividindo-se os valores dos custos dos caminhos pelas respectivas normas do vetor w.

Sendo que, para o algoritmo MMP até a separação das classes obteve-se o vetor de custos w, $w[0] : 0.819397$, $w[1] : 0.209730$, $w[2] : -5.461417$, com $norma = 5.526524$, já com as 1000 iterações obteve-se: $w[0] : 0.530317$, $w[1] : 0.450290$, $w[2] : -5.367245$, com $norma = 5.409307$.

Para o algoritmo *Perceptron Estruturado* obteve-se o vetor de custos: $w[0] : 1.100000$, $w[1] : 0.900000$, $w[2] : -4.700000$, com $norma = 4.910193$. Para o *Perceptron com margem* em suas 1000 iterações obteve-se: $w[0] : 0.942207$, $w[1] : 0.813118$, $w[2] : -4.900445$ com $norma = 5.056014$ e margem máxima final: 0.171629.

Uma medida de qualidade da solução proposta por cada treinamento pode ser considerada como a diferença entre o custo geométrico do caminho do especialista e o custo geométrico do caminho alternativo de menor custo. Os valores em negrito correspondem aos custos geométricos referentes ao caminho do especialista e os custos geométricos dos caminhos alternativos de menor custo.

Tabela 1: *Custos geométricos do algoritmo MMP.*

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Caminho 5	Caminho 6	Margem
1	**0.68218**	5.18501	4.01057	6.55951	**3.03105**	13.13026	**2.34887**
2	4.20548	**0.85970**	4.01057	**0.87708**	5.37992	11.22972	**0.01738**
3	4.20548	5.18501	**1.04591**	4.40039	**3.40348**	6.34577	**2.35756**
4	4.20548	5.18501	**1.04591**	**0.87708**	3.40348	12.02819	**0.16883**
5	1.89456	5.18501	**1.69965**	4.40039	**0.75807**	5.06101	**0.94157**
6	3.03105	**2.83614**	4.01057	5.57482	4.20548	**1.46181**	**1.37432**

Tabela 2: *Custos geométricos do algoritmo MMP com 1000 iterações.*

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Caminho 5	Caminho 6	Margem
1	**0.82247**	5.27904	4.10615	6.50595	**3.16825**	13.32495	**2.34578**
2	4.34114	**0.94985**	4.10615	**1.05746**	5.51403	11.50793	**0.10760**
3	4.341142	5.27904	**1.13104**	4.57613	**3.53062**	6.63518	**2.39958**
4	4.34114	5.27904	**1.13104**	**1.05746**	3.53062	12.08367	**0.07357**
5	2.07855	5.27904	**1.84356**	4.57613	**0.98885**	5.44749	**0.85470**
6	3.16825	**2.93326**	4.10615	5.74902	4.34114	**1.76243**	**1.17082**

Dentre os menores valores dos diferentes algoritmos apresentados nas tabelas, o algoritmo *Perceptron Estruturado com Margem* apresentou o maior, e por isso, o melhor valor de margem: 0.17162.

Tabela 3: *Custos geométricos do algoritmo Perceptron Estruturado.*

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Caminho 5	Caminho 6	Margem
1	**1.85328**	6.78181	5.41730	8.28887	**4.58230**	17.24983	**2.72901**
2	5.94681	**2.13840**	5.41730	**2.38279**	7.31132	15.70202	**0.24439**
3	5.94681	6.78181	**2.54572**	6.47632	**5.39693**	9.83668	**2.85121**
4	5.94681	6.78181	**2.54572**	**2.38279**	5.39693	15.74276	**0.16292**
5	3.40108	6.78181	**2.87157**	6.47632	**2.21987**	8.43144	**0.65170**
6	4.58230	**4.05279**	5.41730	7.84083	5.94681	**3.97133**	**0.08146**

Tabela 4: *Custos geométricos do algoritmo Perceptron Estruturado com Margem.*

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Caminho 5	Caminho 6	Margem
1	**1.52232**	6.35300	5.03659	7.73519	**4.15513**	16.10454	**2.63281**
2	5.47154	**1.78172**	5.03659	**1.95727**	6.78794	14.48772	**0.17554**
3	5.47154	6.35300	**2.12890**	5.90649	**4.84948**	8.87492	**2.72058**
4	5.47154	6.35300	**2.12890**	**1.95727**	4.84948	14.65284	**0.17162**
5	2.99955	6.35300	**2.56460**	5.90649	**1.84396**	7.53298	**0.72063**
6	4.15513	**3.72018**	5.03659	7.22289	5.47154	**3.26211**	**0.45806**

8.2.2 Exemplo 2

A Figura 14 apresenta os mapas 10x10 utilizados no conjunto de treinamento incluindo os caminhos traçados pelo especialista. Pode-se observar nesta figura que todos os caminhos possuem a célula 1 correspondente a entrada (1,1) da matriz como origem e a célula 98 correspondente a entrada (10,8) da matriz como destino. Pelos resultados alcançados constatou-se que ambos os algoritmos convergiram para os mesmos caminhos definidos pela estratégia do especialista, indicando desta forma que o aprendizado foi bem sucedido e as matrizes de custos obtidas refletem a estratégia de planejamento adotada pelo mesmo.

Como parâmetros do algoritmo MMP foram utilizados: taxa de aprendizado 0.5, valor de perda 6 e parâmetro de regularização 1. O vetor w foi inicializado com $[0.1, 0.1, 0.1]$. O algoritmo MMP necessitou de 2 iterações para convergir com tempo de execução aproximado de 0.015 segundos. Também foi estipulado uma máximo de 1000 iterações a fim de analisar o efeito sobre a maximização da margem e seu tempo de execução é 0.937 segundos.

Como parâmetros do algoritmo *Perceptron Estruturado* foram utilizados: taxa de aprendizado 0.2, valor de penalidade 6. O vetor w também foi inicializado com $[0.1, 0.1, 0.1]$. O algoritmo necessitou de 3 iterações para convergir com tempo de execução aproximado de 0.015 segundos.

Os parâmetros do algoritmo *Perceptron Estruturado com Margem* foram os mesmos do *Perceptron Estruturado* e a separação das classes se dá de maneira idêntica, com mesmo número de iterações e tempo. Foi também estipulado um máximo de 1000 iterações para analisar a maximização da margem e o tempo de execução foi de 0.265 segundos.

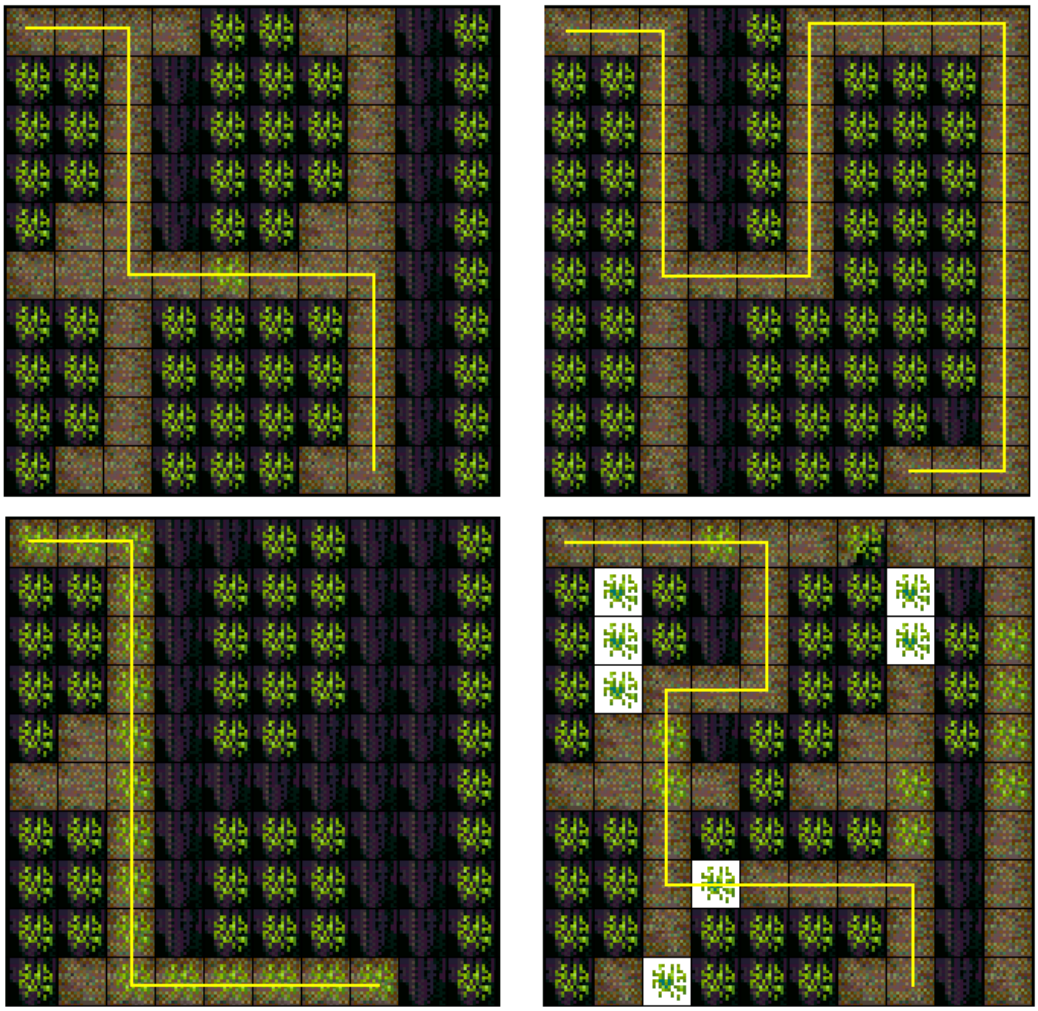


Figura 14: Mapas de treinamento 10x10 com seus respectivos caminhos traçados pelo especialista do domínio.

Sendo que, para o algoritmo MMP até a separação das classes obteve-se o vetor de custos w, $w[0] : 7.662500$, $w[1] : -2.712500$, $w[2] : -14.962500$, com $norma = 17.027859$, já com as 1000 iterações obteve-se: $w[0] : 2.516739$, $w[1] : 1.347019$, $w[2] : -13.359360$, com $norma = 13.659022$.

E para o algoritmo *Perceptron Estruturado* obteve-se o vetor de custos: $w[0] : 3.700000$, $w[1] : 3.100000$, $w[2] : -13.100000$, com $norma = 13.961018$. Para o *Perceptron com margem* em suas 1000 iterações obteve-se: $w[0] : 4.365194$, $w[1] : 3.589140$, $w[2] : -16.230822$ com $norma = 17.186518$ e margem máxima final: 2.865498.

Tabela 5: *Custos geométricos do algoritmo MMP.*

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Margem
1	**2.694849**	25.390744	**6.362368**	15.580350	**3.667519**
2	8.701182	**5.351525**	**7.691073**	18.078462	**2.339548**
3	**12.209404**	30.759886	**0.305382**	17.281973	**11.904022**
4	**5.725177**	14.200993	9.101996	**3.809199**	**1.915978**

Tabela 6: *Custos geométricos do algoritmo MMP depois de 1000 iterações.*

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Margem
1	**5.536673**	31.530185	**9.220282**	19.207751	**3.683609**
2	11.741726	**10.196467**	**10.382447**	21.630638	**0.185980**
3	**15.523892**	38.129467	**7.015028**	22.123424	**8.508864**
4	**9.614510**	21.057823	13.015325	**8.169801**	**1.444709**

Tabela 7: *Custos geométricos do algoritmo* Perceptron Estruturado.

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Margem
1	**5.837683**	34.539030	**9.891829**	20.829427	**4.054146**
2	12.742624	**10.529318**	**11.095180**	23.458175	**0.565861**
3	**17.018817**	42.332158	**9.168386**	24.568410	**7.850431**
4	**10.780016**	23.529803	14.347092	**8.846060**	**1.933956**

Tabela 8: *Custos geométricos do algoritmo* Perceptron Estruturado com Margem.

#	Caminho 1	Caminho 2	Caminho 3	Caminho 4	Margem
1	**1.855873**	26.802047	**5.868689**	15.713296	**4.012815**
2	8.683122	**3.088197**	**7.067071**	18.318895	**3.978874**
3	**12.904772**	34.464635	**4.988392**	19.363068	**7.916380**
4	**6.704027**	15.852928	10.254019	**3.838529**	**2.865498**

Novamente, percebe-se que os dois algoritmos apresentaram resultados bem semelhantes podendo se observar que o algoritmo de máxima margem MMP tem uma ligeira supremacia em relação ao valor da margem para alguns mapas, porém perde para o algoritmo *Perceptron Estruturado* em relação a outros (Tabelas 5, 6, 7 e 8).

Dentre os menores valores dos diferentes algoritmos apresentados nas tabelas, o algoritmo *Perceptron Estruturado com Margem* apresentou o maior, e por isso, o melhor valor de margem: 2.865498.

8.3 Resultados do Conjunto de Teste

8.3.1 Exemplo 1

Na Figura 15 são apresentados os resultados dos algoritmos aplicados ao conjunto de teste. Primeiramente, utilizou-se o algoritmo MMP e, em seguida, utilizou-se o algoritmo *Perceptron Estruturado*. Para a obtenção da matriz de custos associada a cada mapa multiplicou-se o vetor w de custos pela sua matriz F de características. A matriz de características possui dimensões 3x25x4 representando a ocorrência das 3 características para cada um dos possíveis pares estado-ação do mapa, estando os valores das características relacionados ao tipo de terreno para o qual a ação associada ao estado incide. Assim, após a realização do produto $w.F$ tem-se uma matriz 25x4 representando o custo de transição relativo a cada par estado-ação a qual finalmente é reduzida para uma matriz 5x5 representando o custo final de cada célula ou estado. Esta redução é possível considerando o fato de que todas as transições para uma mesma célula possuem o mesmo custo associado obtido diretamente do custo da célula ou do estado sucessor.

A linha contínua amarela são os caminhos achados com base no vetor de custo do algoritmo MMP até a separação das classes, ou seja, 17 iterações. Na maioria dos mapas, os caminhos coincidiram para todos os vetores de custos dos vários algoritmos, porém quando não ocorreu, foram traçados desvios com formas diferentes para os quatro algoritmos como segue. O caminho com quadrados laranjas, mapas três e doze, pertence aos desvios feitos pelo algoritmo MMP com suas 1000 iterações. O caminho com a linha tracejada branca representa o algoritmo *Perceptron Estruturado*, mapas cinco, dez e doze. O algoritmo *Perceptron Estruturado com Margem*, mapas cinco, nove, dez e doze, representado pelo caminho com cruzes azuis claras, como observado na fig. 15.

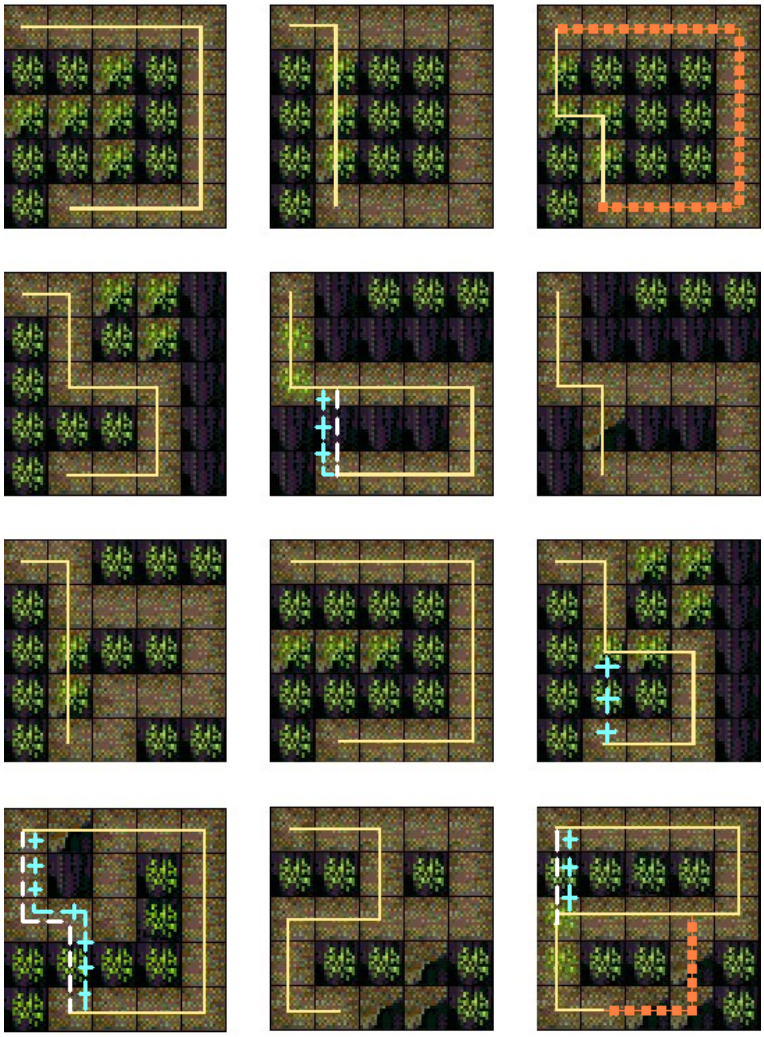


Figura 15: Mapas de testes 5x5 com os custos já associados a cada característica e os caminhos traçados pelo algoritmo *Q-Learning*.

8.3.2 Exemplo 2

Na Figura 16 foram apresentados os resultados dos dois algoritmos aplicados ao conjunto de teste. Primeiramente, utilizou-se o algoritmo MMP e, em seguida, utilizou-se o algoritmo *Perceptron Estruturado*. Para a obtenção da matriz de custos associada a cada mapa multiplicou-se o vetor w de custos pela sua matriz F de características. A matriz de características possui dimensões 3x100x4 representando a ocorrência das 3 características para cada um dos possíveis pares estado-ação do mapa, estando os valores das características relacionados ao tipo de terreno para o qual a ação associada ao estado incide. Assim, após a realização do produto $w.F$ tem-se uma matriz 100x4 representando o custo de transição relativo a cada par estado-ação a qual finalmente é reduzida para uma matriz 10x10 representando o custo final de cada célula ou estado. Esta redução é possível considerando o fato de que todas as transições para uma mesma célula possuem

o mesmo custo associado obtido diretamente do custo da célula ou do estado sucessor.

O tipo e formato dos mapas e caminhos é o mesmo do exemplo 1, com a linha contínua amarela representando os caminhos achados com base no vetor de custo do algoritmo MMP até a separação das classes, em 2 iterações. Todos os caminhos foram iguais no mapa três e no primeiro mapa só o MMP de duas iterações fez um caminho diferente. Como observado na fig. 16.

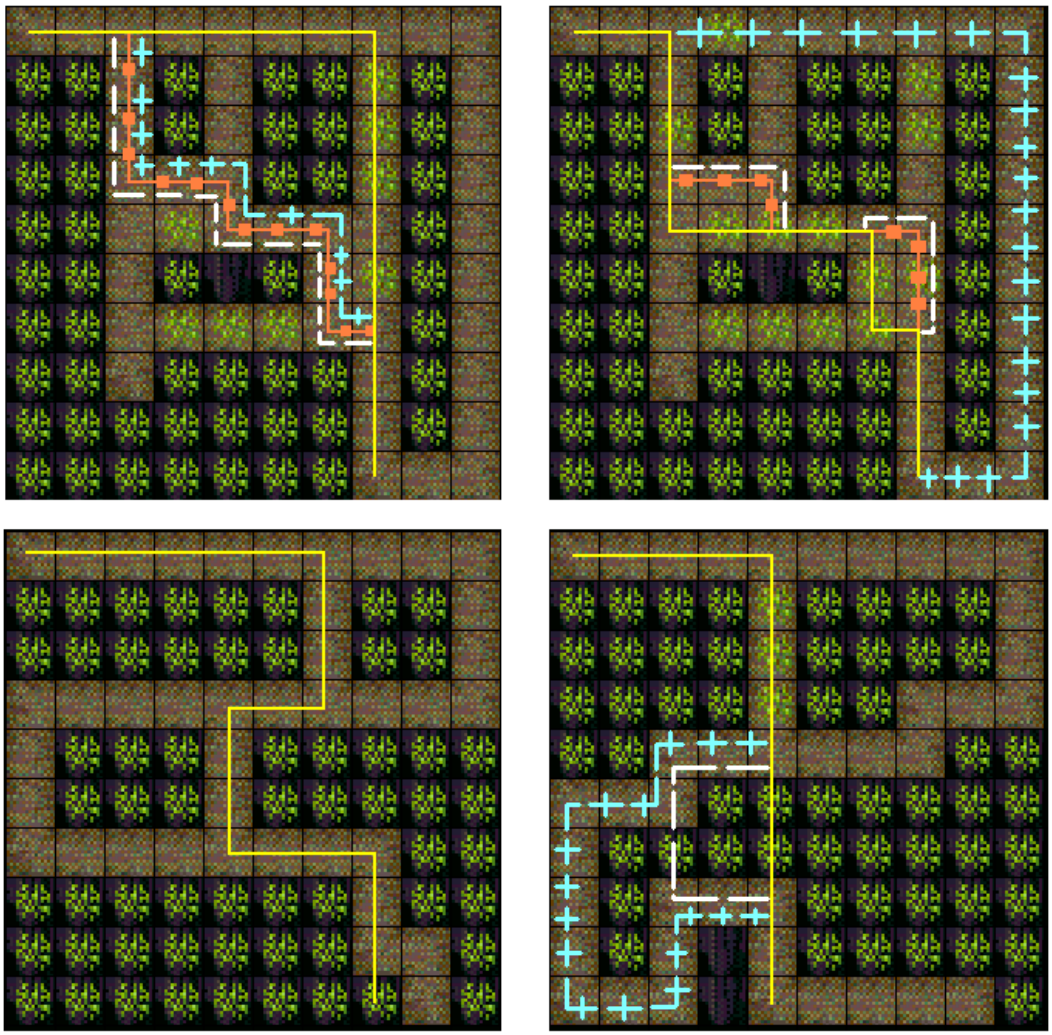


Figura 16: Mapas de testes 10x10 com os custos já associados a cada característica e os caminhos traçados pelo algoritmo *Q-Learning*.

8.3.3 Análise dos Resultados

Na análise dos resultados pode-se observar que os caminhos traçados corresponderam às expectativas do especialista, onde a avaliação de dar a volta ou seguir pela trilha abandonada correspondeu à estratégia proposta pelo mesmo. Além disso, pode-se notar

que os caminhos escolhidos quase sempre evitaram as células sem trilha e com vegetação e rocha.

Percebe-se que todos os algoritmos apresentaram resultados semelhantes com relação aos custos geométricos e as margens. Porém, pode-se observar que o algoritmo *Perceptron Estruturado com Margem* tem uma ligeira supremacia em relação ao valor da margem mínima tanto em relação ao exemplo 1 quanto ao 2.

O caminho ótimo ou caminho mínimo foi obtido com a utilização primeiramente do algoritmo A* e posteriormente sua solução foi comparada com o algoritmo *Q-Leraning incremental* e ambos obtiveram os mesmos resultados. Para tanto, a matriz de custos não deve conter custos negativos no sentido de observar a condição de monótona crescente da função de avaliação. Observando a fig. 16 pode-se constatar que os resultados obtidos com o algoritmo MMP e o *Perceptron* Estruturado foram bastante semelhantes e satisfatórios diferenciando somente na escolha do caminho relativo ao décimo mapa. Neste caso, o caminho tracejado de branco representa o caminho sugerido pelo algoritmo *Perceptron Estruturado* e o caminho contínuo de amarelo representa o caminho sugerido pelo algoritmo MMP. A seguir na figura 17 podemos ver o programa do *Q-Leraning incremental* sendo executado no segundo mapa do primeiro exemplo do conjunto de testes, a célula vermelha é o agente e a azul o objetivo.

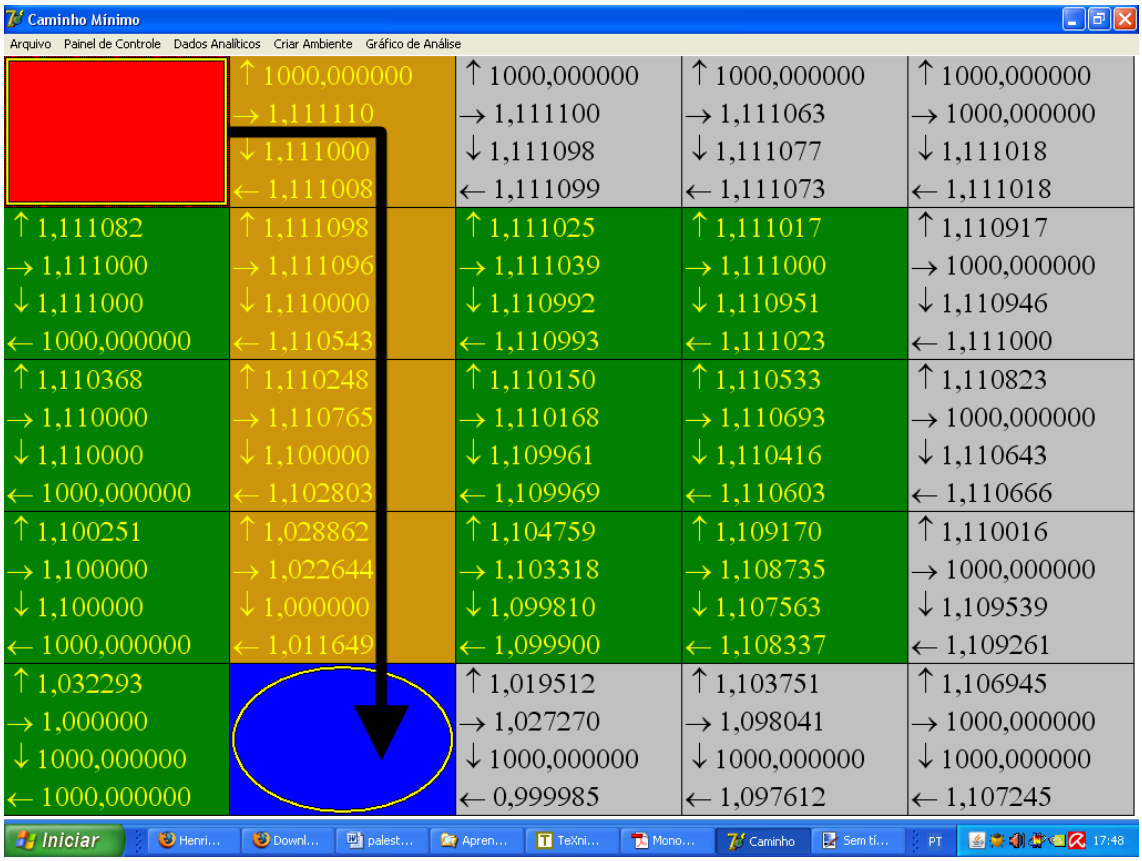


Figura 17: Algoritmo *Q-Learning* sendo executado.

9 Conclusão

Neste trabalho foi apresentado o algoritmo *Perceptron Estruturado*, uma alternativa a técnica de solução do algoritmo MMP. Como foi observado tanto nas Fig. 12 e 15, quanto nas Tabelas 1 e 2, os algoritmos apresentaram resultados bem semelhantes, todavia o algoritmo *Perceptron Estruturado* exige um menor gasto de memória e processamento, o que o deixou mais rápido. Isto se deveu principalmente ao fato do mesmo não precisar executar o algoritmo A* em seu processo de aprendizado.

Além disso, abordou-se o problema de determinação do caminho mínimo entre dois pontos fixos, que está incluído entre os problemas clássicos do mundo real. Foi feita uma análise de diferentes algoritmos, que utilizaram desde técnicas de busca através de um processo exploratório até as técnicas de aprendizado por reforço da Inteligência Artificial, de uma forma evolutiva.

A utilização de técnicas de aprendizado no problema de determinação do caminho mínimo entre dois pontos fixos aliado ao problema de predição de custos permitiu que o problema fosse solucionado sem qualquer tipo de planejamento, ou seja, sem um conhecimento prévio do ambiente nem dos custos de suas características onde o problema está inserido. Como foi visto anteriormente, a utilização dessa técnica nos permite resolver problemas muito mais complexos, onde não existe um conhecimento prévio do ambiente, e que demandariam um grande esforço computacional se fossem resolvidos através de técnicas clássicas da computação.

No final de todo processo, foi utilizado o algoritmo *Q-Learning Incremental*, que é uma variante do *Q-Learning Tabular*, sendo esse último um dos algoritmos mais utilizados na implementação do aprendizado por reforço. O algoritmo *Q-Learning Incremental* se mostrou até 8 vezes mais eficiente do que o *Q-Learning Tabular*, uma vez que ele atualiza mais de um estado a cada transição de estados.

Constatou-se, também, que o algoritmo *Perceptron Estruturado* prediz eficientemente os mapas de custos de novos ambientes a partir de um pequeno número de mapas de

treinamento, possibilitando que a sua utilização seja estendida para outras aplicações envolvendo o problema de predição de dados estruturados.

Finalizando, os resultados alcançados corresponderam ao esperado. Demonstrando sempre uma forte associação entre a saída obtida representada pelos custos aprendidos e a determinação de novos caminhos seguindo a estratégia definida pelo especialista para o conjunto de treinamento.

Como trabalho futuro tem-se a perspectiva de se trabalhar com a representação dual do modelo perceptron, também chamada de representação dependente dos dados, e pode ser interpretada como um classificador *kernel*.

APÊNDICE A – Programação Dinâmica

Nos problemas de aprendizado por reforço, é necessário definir políticas ótimas e funções de valor ótimas. Quando o agente possui um completo conhecimento sobre as funções de transição e de retorno, uma política ótima pode ser determinada de forma eficiente com o uso da Programação Dinâmica. Essa técnica de solução deve ser aplicada principalmente nos sistemas considerados *offline*, nos quais é possível a idealização de um modelo interno e a simulação de um processo de busca que determine a solução do problema, ao contrário dos sistemas *online*, onde o conhecimento é adquirido a partir da interação do agente com o meio.

No capítulo 3 foi visto como a determinação de caminhos pode ser implementada como um processo de análise *forward* ou *backward* do espaço de estados. De fato, o problema de caminho mínimo analisado sobre o enfoque da programação dinâmica tem uma forte similaridade com o paradigma de aprendizado por reforço.

A.1 Sistemas de Decisão

A Programação Dinâmica é uma técnica de otimização que busca encontrar a melhor solução entre várias alternativas em um processo decisório de multi-estágios. É de grande utilidade no tratamento de problemas relacionados ao controle de sistemas dinâmicos discretos no tempo, planejamento dinâmico com equacionamento espaço-estado e, também, na solução de problemas de programação discreta e otimização combinatória envolvendo quase sempre a estrutura de grafos. No processo de decisão em multi-estágios as decisões são tomadas de forma sequencial, uma de cada vez, contrapondo a forma mais comum de otimização conjunta, nas quais as decisões são tomadas simultaneamente, características dos métodos de Programação Matemática. Entretanto, ao contrário das técnicas míopes, onde as decisões tomadas são irreversíveis, a Programação Dinâmica permite uma maior exploração do conjunto de soluções potenciais, no sentido de preservar a otimalidade da solução.

Portanto, a programação Dinâmica consiste em transformar um processo decisório formado por um simples estágio em um processo multi-estágio, ou seja, decompondo um problema com n variáveis de decisão em n sub-problemas de uma única variável, preservando a otimalidade da solução encontrada. A exemplo das equações de *Bellman* aplicadas à solução do problema do caminho mínimo, a programação Dinâmica se baseia, também, no princípio da otimalidade de Bellman.

Um sistema de decisão de um único estágio é representado esquematicamente como:

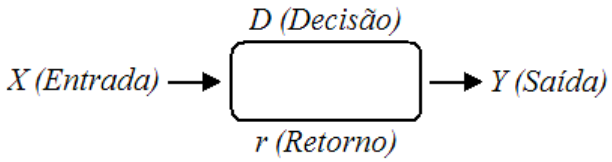


Figura 18: Sistema de decisão de um único estágio

Considerando:

X: como o estado que descreve o sistema no início do estágio. Também pode ser descrito como X_t, ou seja, representando o estado do problema no instante de tempo t, se for introduzido um sistema discreto de tempos.

Y: como o estado que descreve o sistema no final do estágio. Também pode ser descrito como X_{t+1}, representando o estado do problema no instante de tempo t+1.

D: O conjunto de variáveis de decisão, o qual fornecerá a escolha da ação apropriada.

δ: uma função de transição que expressa o estado de saída em função do estado de entrada e das variáveis de decisão. Ou seja: $Y = \delta(X, D)$

r: uma função de retorno, também chamada de recompensa na ótica do aprendizado por reforço, sendo $r = r(X, D, Y) = r(X, D, \delta(X, D)) = r(X, D)$

V: uma função de valor que mede o mérito de cada estado, ou seja, o retorno ótimo em função do conjunto de variáveis de decisão.

Com isso, o interesse está em se determinar qual será o retorno ótimo para um determinado estado de entrada X, ou seja, deseja-se computar: $V(X) = r(X, D(X)) = r(X, D^*) = Min_D r(X, D)$, onde D^* se refere a uma política de decisões ótimas.

Por outro lado, um sistema de decisões em multi-estágio consiste de um conjunto de estágios conectados sequencialmente, na forma:

Para todos os n estágios deste sistema, considerando $n = 1, ..., N$, pode-se expressar

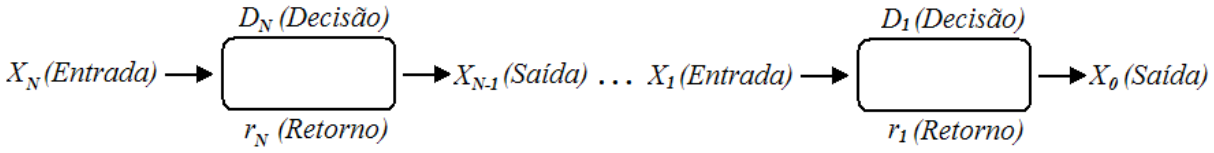


Figura 19: Sistema de decisão em N estágios

as transformações de estágios na forma:

$X_{n-1} = n(X_n, D_n)$

$r_n = r_n(X_n, D_n)$

Considerando a transitividade dos estágios tem-se:

$X_n = \delta_{n+1}(X_{n+1}, D_{n+1}) = \delta_{n+1}(\delta_{n+2}(X_{n+2}, D_{n+2}), D_{n+1}) = ... = \delta_{n+1}(X_N, D_N, ..., D_{n+1})$

Portanto, a determinação de X_n depende somente do estado inicial X_N e do conjunto de decisões tomadas a priori: $(D_N, D_{N-1}, ..., D_{n+1})$.

Da mesma forma, para a função de retorno tem-se:

$r_n = r_n(X_N, D_N, D_{N-1}, ..., D_n)$

A.2 Solução Recursiva

Na sua forma mais geral o problema de decisão de N-estágios consiste em maximizar a função de retorno $r_n(X_N, D_N, D_{N-1}, ..., D_1)$, ou seja, em se determinar o retorno máximo em função do estado inicial X_N. Definindo a função valor $V_N(X_N)$ como uma função dos retornos individuais de cada estágio e $D_n^* = D_n(X_n), X_n^* = \delta_n(X_n)$ como as decisões ótimas e respectivos estados, pode-se escrever a função valor de um estado na forma: $V_N(X_N) = f(r_N(X_N, D_N^*), r_{N-1}(X_{N-1}, D_{N-1}^*), ..., r_1(X_1, D_1^*))$ Esse resultado pode ser determinado através da solução de um problema de otimização descrito como:

$Max_{D_N, D_{N-1}, ..., D_1}\{f(r_N(X_N, D_N), r_{N-1}(X_{N-1}, D_{N-1}), ..., r_1(X_1, D_1))\}$

Sujeito a:

$X_{n-1} = \delta_n(X_n, D_n)$, para $n = 1, ..., N$

A forma mais comum de solução deste problema consiste em se decompor o mesmo em N subproblemas, cada qual contendo somente uma variável de estado e uma variável de decisão. Progressivamente, se determina a solução ótima em cada estágio e, finalmente, combina-se as diversas decisões dos N estágios no, sentido de gerar a solução final.

Para a decomposição do problema, é necessário que a função f tenha a propriedade de separabilidade.

Quando a análise procede do estágio N para o estágio 1 e o retorno ótimo é determinado em função do estado de saída, denomina-se a recursão de *forward* ou direta. Caso contrário, quando o retorno ótimo é determinado em função do estado de entrada, tem-se a recursão *backward* ou inversa.

A.3 Relação de Simples Recorrência

Utilizando o processo de recursão *backward*, o caminho ótimo, cujo valor se equivale ao valor do retorno ótimo que retrata a política ótima do problema, é fornecido pelo estado de entrada. Tratando-se de um grafo sem circuitos pode-se determinar a priori o número de estágios do problema cujo valor é equivalente ao número de níveis do grafo.

Em problemas onde ocorre uma maior interdependência entre vários estágios, a solução do problema de Programação Dinâmica fica comprometida, estabelecendo uma função de complexidade exponencial em relação ao número de estágios envolvidos.

A.4 Complexidade e Aproximação Heurística

Quando a programação dinâmica estabelece uma simples relação de recorrência, a exemplo da solução das equações de Bellman no problema de caminho mínimo acíclico, tem-se sua solução, por uma técnica recursiva, em tempo quadrático. Ou seja, a complexidade do algoritmo é de ordem máxima $O(n^2)$, onde n se refere ao número de estados do problema, caracterizando a interdependência somente entre dois estágios sucessivos. Nesse caso, a solução do problema em um estágio é realizada tomando-se somente a solução obtida no estágio anterior.

Para o caso mais geral, onde existe uma interdependência entre todos os estágios subsequentes do problema, a complexidade do algoritmo é de ordem exponencial, tendo como expoente o número de estágios envolvidos que representam as variáveis de estado do problema. Como solução para este problema adotam-se duas técnicas básicas. A primeira é utilizada na solução das equações de Bellman na sua forma mais geral e consiste na utilização de um método de aproximações sucessivas, onde os valores das variáveis de estado são aproximados a cada iteração. Está relacionada aos métodos de diferença temporais característicos do aprendizado por reforço. Neste caso, considerando uma representação

tabular para o armazenamento da aproximação das funções valores, para um problema de minimização em um estágio $n+1$, tem-se:

$V_a(X_{n+1}) = Min_{D_{n+1}}\{r_{n+1}(X_{n+1}, D_{n+1}) + V_a(X_n)\}$

onde:

$X_n = \delta_{n+1}(X_{n+1}, D_{n+1})$

$V_a(X_n) = V(X_n)$, para $n = N$

Considerando $V_a(Xn)$ o valor aproximado armazenado para $V(X_n)$.

Nesse caso, pode-se considerar que, a cada transição, a aproximação da função valor de um estado é obtida em função da aproximação da função valor do estado subsequente acrescido do custo da transição. Caso esta segunda aproximação seja escolhida como a melhor entre todas as transições associadas ao estado subsequente, tem-se uma implementação semelhante ao método de aproximações sucessivas, o qual constitui a base para o algoritmo *Q-learning* tabular que será visto adiante.

A segunda, utilizada nos algoritmos de busca de Inteligência Artificial, consiste na elaboração de uma aproximação heurística para os valores das variáveis de estado. O cômputo destes valores, ou seja, da heurística, pode ser feito de forma *forward*, a exemplo do algoritmo A*. Ou seja, para um problema de minimização em um estágio $n+1$, tem-se como valor de mérito de um estado:

$f(X_{n+1}) = g(X_{n+1}) + h(X_{n+1})$

Considerando:

$h(X_{n+1}) = V(X_{n+1})$

Sendo:

$V(X_{n+1}) = Min_{D_{n+1}}\{r_{n+1}(X_{n+1}, D_{n+1}) + V(X_n)\}$

Onde:

$X_n = \delta_{n+1}(X_{n+1}, D_{n+1})$

$V(X_n) = h(X_n)$

Ou seja, calcula-se a função valor do estado sucessor como uma aproximação heurística computada pela função h. Obviamente, para um estado solução X_N, tem-se:

$V(X_N) = h(X_N) = 0$

Essa heurística pode ser computada de forma *backward*, através do emprego de um processo de controle recursivo com o uso de uma técnica míope para a aproximação da função valor do estado. Nesse caso, a solução trivial consiste em se considerar a otimização em cada estágio somente de uma variável, fixando as demais variáveis anteriores. Desta forma, considera-se a escolha de uma variável ótima em cada estágio, cujo valor é fixado para as demais otimizações subsequentes. Assim, pode-se definir o cálculo da função heurística como:

$h(X_n) = Min_{D_n}\{r_{n+1}(X_{n+1}, D_{n+1}) + h(X_{n+1})\}$, para $n = 1, ..., N-1$

$h(X_n) = 0$, para $n = N$

Fazendo:

$X_n = \delta_{n+1}(X_{n+1}, D*_{n+1})$

APÊNDICE B – Otimização

Aqui é apresentado o ferramental necessário para a solução do problema de maximização, sob o ponto de vista da otimização. Sendo abordado desde a área geral de otimização não-linear, e aos poucos aumentando a especificação, sendo visto a otimização convexa e a programação quadrática.

B.1 Otimização Não-Linear

A Otimização é a área da Programação Matemática que trata de problemas cujo interesse consiste em encontrar pontos de máximo ou de mínimo de funções. Programação ou otimização não-linear é o processo de resolver um sistema de igualdades e desigualdades, juntamente com suas restrições, sobre um conjunto de variáveis reais desconhecido, juntamente com uma função objetivo a ser maximizada ou minimizada, onde algumas das restrições ou a função objetivo é não-linear. Matematicamente o problema pode ser indicado como (BRINKHUIS; TIKHOMIROV, 2005):

$$\max_{x \in X} f(x) \text{ ou } \min_{x \in X} f(x)$$

Onde $f : R^n \rightarrow R$ e $X \subseteq R^n$.

A seguir será visto a otimização convexa, necessária para o entendimento posterior das formulações da predição de dados estruturados.

B.2 Otimização Convexa

Otimização convexa é um subcampo da otimização matemática e estuda o problema de minimizar ou maximizar funções convexas.

O convexidade de $\mathcal{X}$ e f tornam aplicáveis poderosas ferramentas de análise convexa. O teorema de Hahn–Banach e a teoria de subgradientes conduzem a teoria de condições

necessárias e suficientes para otimalidade, a teoria dual generaliza isso para a programação linear e efetivos métodos computacionais (BOYD; VANDENBERGHE, 2004).

A minimização convexa tem aplicações em uma vasta gama de disciplinas, como o controle automático de sistemas, processamento de sinais e desenhos de circuitos eletrônicos.

Uma característica importante da função convexa utilizada aqui é a seguinte: o problema de maximizar uma função convexa pode ser reformulado equivalentemente como um problema de minimização convexa. E tem-se também a seguinte propriedade das funções convexas: se existe um mínimo local, então ele é o mínimo global.

B.3 Multiplicadores de Lagrange

Achar pontos extremos de funções é muito importante quanto se deseja otimizar algo. Existem muitos métodos, tanto determinísticos como interativos para resolver estes problemas. Um desses importantes métodos é o de Lagrange, ele é específico para problemas que se conhece o domínio de trabalho, ou seja, que tenha suas restrições bem definidas.

Uma maneira para achar os pontos de máximo é igualar a zero todas as derivadas parciais. Se não houvesse vínculos, isto seria o mesmo que impor $df = 0$, onde df, o diferencial da função f, é dado por:

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz$$

Uma vez eliminado z por meio do vínculo, tem-se, em lugar desta última, a equação:

$$dF = \frac{\partial F}{\partial x}dx + \frac{\partial F}{\partial y}dy = 0$$

Ou seja, não aparece mais o diferencial dz, indicando que a função F não depende de z. O método de Lagrange oferece uma técnica mais eficiente e simétrica para eliminar a dependência em z, ou seja, para se livrar do termo em dz na expressão do diferencial da função cujos máximos se procura.

Considere o diferencial da função f:

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz$$

E, como $g(x, y, z) = 0$, tem-se:

$$dg = \frac{\partial g}{\partial x}dx + \frac{\partial g}{\partial y}dy + \frac{\partial g}{\partial z}dz = 0$$

Seja λ um número qualquer, de valor a ser determinado posteriormente. Adiciona-se a df a quantidade λdg, que é zero. Logo:

$$df = df + \lambda dg$$

Portanto, pode-se escrever:

$$df = \left(\frac{\partial f}{\partial x} + \lambda\frac{\partial g}{\partial x}\right)dx + \left(\frac{\partial f}{\partial y} + \lambda\frac{\partial g}{\partial y}\right)dy + \left(\frac{\partial f}{\partial z} + \lambda\frac{\partial g}{\partial z}\right)dz$$

Mas, como λ é indeterminado, pode-se determiná-lo agora impondo que o coeficiente de dz na expressão anterior seja nulo, ou seja:

$$\frac{\partial f}{\partial z} + \lambda\frac{\partial g}{\partial z} = 0$$

Com isso, tem-se agora um df independente de z, e pode-se localizar seus pontos de máximo impondo que $df = 0$, ou, mais precisamente, que $df + \lambda dg = 0$. Mas isso dá as condições:

$$\frac{\partial f}{\partial x} + \lambda\frac{\partial g}{\partial x} = 0$$
$$\frac{\partial f}{\partial y} + \lambda\frac{\partial g}{\partial y} = 0$$

Como, adicionalmente, tem-se a condição dada pela Equação $\frac{\partial f}{\partial z} + \lambda\frac{\partial g}{\partial z} = 0$, nota-se que o conjunto das equações que determinam os pontos de máximo (bem como o valor de λ) é obtido da seguinte maneira: igualem-se a zero as derivadas parciais da função:

$$f + \lambda g$$

A generalização é imediata. Seja $f(x, y, z, u, v)$ a função cujos pontos de máximo deseja-se localizar, e sejam $g(x, y, z, u, v) = 0$ e $h(x, y, z, u, v) = 0$ condições subsidiárias. Então igualam-se a zero as derivadas parciais da função:

$$f + \lambda_1 g + \lambda_2 h$$

onde λ_1 e λ_2 são coeficientes a determinar. Se houver n condições subsidiárias $g_i = 0$,

igualem-se a zero as derivadas parciais da função:

$$f + \sum_i \lambda_i g_i$$

Os λ_i são denominados multiplicadores de Lagrange.

Os multiplicadores de Lagrange possuem a seguinte definição. Considere f com suas m restrições g. Sejam elas deriváveis em primeira ordem, contínuas, e que $\nabla g \neq 0$ em qualquer circunstância. Se f tiver um extremo relativo dentro de suas restrições, este ponto ocorre em um ponto $P(x_1^*, x_2^*, ..., x_n^*)$, tal que P pertença a uma superfície de restrição de f na qual os gradientes $\nabla f(x_1^*, x_2^*, ..., x_n^*)$ e $\nabla g(x_1^*, x_2^*, ..., x_n^*)$ são paralelos, ou seja, existe λ tal que a seguinte condição seja satisfeita:

$$\nabla f(x_1^*, x_2^*, ..., x_n^*) = \lambda \nabla g(x_1^*, x_2^*, ..., x_n^*)$$

Apesar de sua solução ser bem simples, para que se ache a sua exata solução nesta forma é necessário que as restrições sejam estritamente na forma de equações, e não inequações como se apresenta em nosso problema de predição de dados estruturados. Porém pode-se utilizar um método conhecido como Relaxação Lagrangeana nas inequações, neste caso usa-se o subgradiente com os multiplicadores de Lagrange, ao invés do gradiente, para achar uma solução bem aproximada. Tal abordagem é devidamente explicada neste mesmo capítulo nas próximas seções.

B.4 Programação Quadrática

A programação quadrática é um tipo especial de problema de otimização matemática. Representa o problema de otimizar uma função quadrática de diversas variáveis estando sujeita a restrições lineares sobre essas variáveis.

O problema de programação quadrática pode ser formulado como (NOCEDAL; WRIGHT, 2006):

Assuma que $x \in \mathbb{R}^n$. A matriz $Q_{n \times n}$ é simétrica, e c é um vetor $(n \times 1)$.

Minimizar (em relação a x)

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T Q \mathbf{x} + \mathbf{c}^T \mathbf{x}$$

Sujeita a uma ou mais restrições na forma:

$$A\mathbf{x} \leq \mathbf{b} \text{ (restrição de desigualdade)}$$

$$E\mathbf{x} = \mathbf{d} \text{ (restrição de igualdade)}$$

Onde $\mathbf{x}^T$ indica o vetor transposto de $\mathbf{x}$. A notação $Ax \leq b$ significa que todas as entradas do vetor Ax são menores ou iguais que a entrada correspondente do vetor $\mathbf{b}$.

Se Q é uma matriz positiva semi definida, então $f(\mathbf{x})$ é uma função convexa. Tem-se então que a programação quadrática convexa é um caso especial do problema geral de otimização convexa, e tem um mínimo global se existe pelo menos um vetor x que satisfaça as restrições e $f(\mathbf{x})$ está limitada em baixo na região viável. É condição suficiente para ter um ponto $\mathbf{x}$ como um mínimo global a função $f(\mathbf{x})$ ser convexa. Esse mínimo global é único. Se $Q = 0$ então tem-se um problema de programação linear.

O dual de um problema de programação quadrática também é um problema de programação quadrática. Para demonstrar isso considere o caso onde $c = 0$ e Q é positivo definido. Pode-se escrever o Lagrangiano (NOCEDAL; WRIGHT, 2006):

$$L(x, \lambda) = \tfrac{1}{2} x^T Q x + \lambda^T (Ax - b)$$

Para calcular a função $g(\lambda)$, definida como $g(\lambda) = \inf_x L(x, \lambda)$, calcula-se o ínfimo de L, com $\nabla_x L(x, \lambda) = 0$:

$$x^* = -Q - 1A^T\lambda$$

Portanto, a função dual é:

$$g(\lambda) = -\tfrac{1}{2}\lambda^T A Q^{-1} A^T \lambda - b^T \lambda$$

Então a função do problema dual de programação quadrática é:

$$\text{Max: } -\tfrac{1}{2}\lambda^T A Q^{-1} A^T \lambda - b^T \lambda$$

$$\text{Sujeito a: } \lambda \geqslant 0$$

APÊNDICE C – Subgradiente

C.1 Subderivadas, Subgradientes e Subdiferenciais

Em matemática, os conceitos de subderivada, subgradiente, e subdiferencial surgem em análise convexa, e estão frequentemente relacionados à otimização convexa. Algumas propriedades importantes de uma função convexa: uma função convexa em $[a, b]$ é sempre contínua em (a, b); se uma função convexa possui um mínimo local, ele também será um mínimo global; o máximo de funções convexas também é uma função convexa (ROCKAFELLAR, 1970).

Em matemática, uma função f de $[a, b]$ em R é dita **convexa** se a região sobre o seu gráfico, ou seja, o conjunto $\{(x, y) \in \mathbb{R}^2 \,|\, y \geq f(x)\}$, for um **conjunto convexo** (Fig. 20). Isto equivale a afirmar que, para quaisquer x e y pertencentes a $[a, b]$ e para todo $t \in [0, 1]$, tem-se:

$$f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y)$$

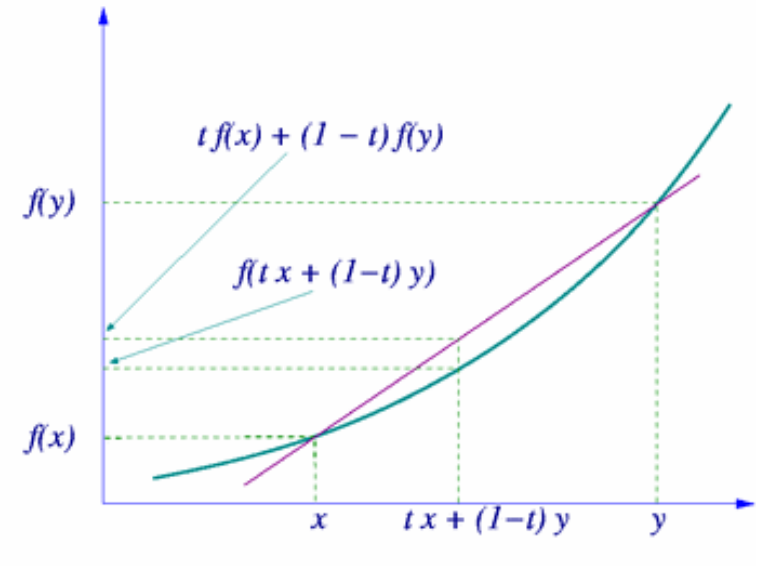


Figura 20: Gráfico de uma função convexa

Uma função continuamente diferenciável de uma variável é convexa num intervalo, se e só se para $f(y) \geq f(x) + f'(x)(y - x)$, para todos x e y no intervalo. Porém, fazendo-se $f : I \rightarrow R$, onde I é um intervalo real, ser uma função convexa definida sobre um intervalo aberto na reta dos reais, tem-se que tal função pode não ser necessariamente diferenciável em todos os pontos, como por exemplo, o valor absoluto, $f(x) = |x|$. Entretanto, para qualquer x^0 no domínio da função pode-se traçar uma linha a qual cruza o ponto $(x^0, f(x^0))$ e em qualquer lugar toca ou passa abaixo do gráfico de f (Fig. 21). O coeficiente angular desta linha é chamado de **subderivada**, e ao contrário da derivada em um ponto, esta pode ter mais de um valor (URRUTY; LEMARECHAL, 2001).

Rigorosamente, uma **subderivada** de uma função convexa $f : I \rightarrow R$ em um ponto x^0 em um intervalo aberto I é um número real c onde $f(x) - f(x_0) \geq c(x - x_0)$, $\forall x \in I$. O conjunto $[a, b]$ de todas subderivadas é chamada de **subdiferencial** da função f em x^0.

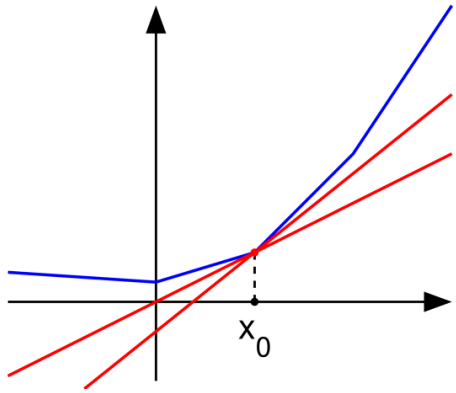


Figura 21: Subderivadas de uma função convexa

Como exemplo de alguns valores de **subdiferencial** pode-se considerar a função $f(x) = |x|$ a qual é convexa. Então, o subdiferencial na origem é o intervalo $[-1, 1]$, em qualquer ponto que $x^0 < 0$ é o valor -1, e se $x^0 > 0$, 1. Se o subdiferencial tiver um valor único em um ponto, a função é diferenciável neste ponto, ou seja, o subdiferencial é igual a derivada.

Os conceitos de **subderivadas** e **subdiferenciais** podem ser generalizados para funções de muitas variáveis. Se $f : U \rightarrow R$ é um valor real de uma uma função convexa definida em conjunto aberto convexo no espaço Euclidiano R^n, um vetor g neste espaço, (Fig. 22), é chamado de **subgradiente** em um ponto $x^0 \in R^n$ se $\forall x \in R^n$:

$$f(x) \geq f(x^0) + g^T(x - x^0); \forall x \in R^n$$

O conjunto de todos os subgradientes de f em x^0 é chamado de subdiferencial de f

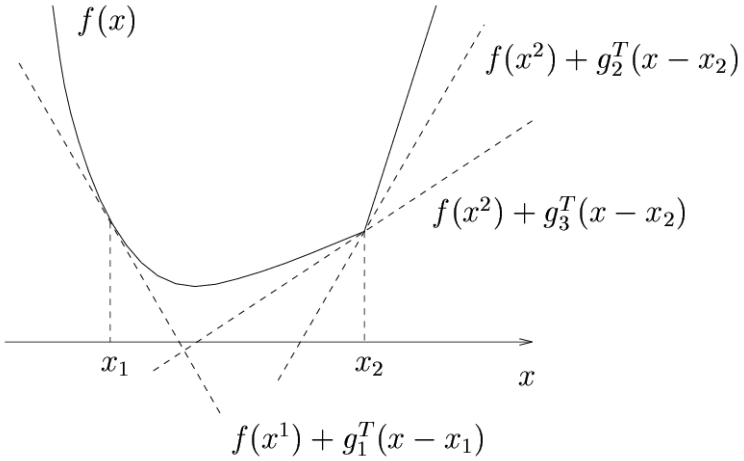


Figura 22: Exemplos de alguns subgradietes

em x^0 e é denotado como $\partial f(x^0)$. na prática, o conhecimento de qualquer elemento de $\partial f(x)$ nos pontos necessários é suficiente para a implementação de inúmeros métodos de otimização.

Referências

BAKIR, G. et al. *Predicting Structured Data*. 1. ed. [S.l.]: MIT Press, 2006. ISBN 9788573077186.

BELLMAN, R. E. *Dynamic Programming*. 1. ed. [S.l.]: Princeton University Press, 1957. ISBN 0486428095.

BERTSEKAS, D. *Minimization Methods for Non-differentiable Functions*. 1. ed. [S.l.]: Shor, Naum Z., 1985. ISBN 0-387-12763-1.

BERTSEKAS, D. *Convex Analysis and Optimization*. 1. ed. [S.l.]: Athena Scientific, 2003. ISBN 1886529450.

BOYD, B.; L., X.; MUTAPCIC, A. Subgradient methods, stanford university, autumn. *Notes for EE392o*, 2003.

BOYD, S.; VANDENBERGHE, L. *Convex Optimization*. 1. ed. [S.l.]: Cambridge University Press, 2004. ISBN 0521833787.

BRINKHUIS, J.; TIKHOMIROV, V. *Optimization: Insights and Applications*. 1. ed. [S.l.]: Princeton University Press, 2005. ISBN 0691102872.

COELHO, M. A. N.; NETO, R. F.; BORGES, C. C. H. Predição de dados estruturados utilizando a formulação de máxima margem com aplicação em planejamento de caminhos. *Congresso Ibero-Latino Americano de Métodos Computacionais em Engenharia - 30º CILAMCE*, Novembro 2009.

COLLINS, M. Discriminative training methods for hidden markov models: Theory and experiments with the perceptron algorithm. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2002.

CORMEN, T. H.; RIVEST, C. L. end R. *Introduction to Algorithms*. 2. ed. [S.l.]: McGraw-Hill, 2000. ISBN 0262032937.

DAUMé III, H.; MARCU, D. Learning as search optimization: approximate large margin methods for structured prediction. In: *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM, 2005. p. 169–176. ISBN 1-59593-180-5.

DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification*. 2. ed. [S.l.]: Wiley-Interscience, 2001. ISBN 0471056693.

FERNANDES, J. H. C. Ciberespaco: Modelos, tecnologias, aplicacões e perspectivas. In: . [S.l.: s.n.], 2000.

GOLDBARG, M. C.; LUNA, H. P. L. *Otimização Combinatória e Programação Linear*. 2. ed. [S.l.]: Campus / Elsevier, 2005. ISBN 8535215204.

HAYKIN, S. *Redes Neurais Princípios e Práticas*. 2. ed. [S.l.]: Bookman Companhia ED, 2001. ISBN 9788573077186.

KAELBLING, L.; LITTMAN, M.; MOORE, A. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4*, p. 237–285, 1996.

KIVINEN, J.; SMOLA, A. J.; WILLIAMSON, R. C. Online learning with kernels. *IEEE Transactions on Signal Processing*, n. 52, p. 2165–2176, Agosto 2002.

LEITE, S. C.; NETO, R. F. Incremental margin algorithm for large margin classifiers. *Neurocomput.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 71, n. 7-9, p. 1550–1560, 2008. ISSN 0925-2312.

LEMARECHAL, C. Nondifferentiable optimization. *Elsevier Science Publishers, North-Holland*, p. 529–572, 1989.

LIMA, A. Uma Estratégia de Decomposição por Relaxação Lagrangeana para a Otimização da Programação Diária da Operação de Sistemas Hidrotérmicos com Modelagem Detalhada da Rede Elétrica. In: . [S.l.]: Aplicação ao Sistema Brasileiro, Tese (Doutorado), COPPE/UFRJ, 2007.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, p. 115–133, 1943.

NEMHAUSER, G.; WOLSEY, L. *Integer and Combinatorial Optimization*. 1. ed. [S.l.]: Wiley-Interscience Series in Discrete Mathematics and Optimization, 1999. ISBN 978-0471359432.

NETO, L. et al. Very short term load forecasting system using neural networks. *The Seventeenth Annual International Symposium on Forecasting*, n. 3, p. 51, Junho 1997.

NOCEDAL, J.; WRIGHT, S. J. *Numerical Optimization*. 2. ed. [S.l.]: Cambridge University Press, 2006. ISBN 978-0-387-30303-1.

PATEL, A. *Amit's Thoughts on Pathfinding*. [S.l.]: Stanford University, Internet, 2007.

RATLIFF, N.; BAGNELL, J. A.; ZINKEVICH, M. Maximum margin planning. *Twenty Second International Conference on Machine Learning. ICML06*, p. 729–736, 2006.

ROCKAFELLAR, R. T. *Convex analysis*. 1. ed. [S.l.]: Princeton University Press, 1970. ISBN 0691015864.

RODRIGUES, S. Relaxação Lagrangeana e Subgradientes com Dilatação de Espaço Aplicados a um Problema de Grande Porte. In: . [S.l.]: Tese, COPPE/UFRJ), 1994.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. In: *Psychological Review*. [S.l.: s.n.], 1958. p. 386–408.

RUSSEL, S. J.; NORVING, P. *Inteligência Artificial*. 2. ed. [S.l.]: Editora Campus, 2004. ISBN 8535211772.

SCHRIJVER, A. *Combinatorial Optimization: Polyhedra and efficiency*. 1. ed. [S.l.]: Springer Verlag NY, 2003. ISBN 3540443894.

SOUZA, T. C. A. d. Métodos subgradientes em otimização convexa não diferenciável. In: . [S.l.]: Universidade Federal de Juiz de Fora, Dissertação (Mestrado em Modelagem Computacional), 2008.

TASKAR, B. Learning Structures Prediction Models: A Large Margin Approach. In: . [S.l.]: Dissertação (Doctor of Philosophy), Stanford University, 2004.

TASKAR, B. et al. Learning structures prediction models: A large margin approach. *Twenty Second International Conference on Machine Learning. ICML05*, p. 896–903, 2005.

TASKAR, B.; GUESTRIN, C.; KOLLER, D. Max-margin markov networks. *Neural Information Processing systens Foundation*, 2003.

TSOCHANTARIDIS, I. et al. arge margin methods for structured and interpendent output variables. *Journal of Machine Learning*, p. 1453–1484, 2005.

URRUTY, J. B. H.; LEMARECHAL, C. *Fundamentals of Convex Analysis*. 1. ed. [S.l.]: Springer, 2001. ISBN 3-540-42205-6.

VAPNIK, V. *Statistical Learning Theory*. 1. ed. [S.l.]: Wiley and Sons Inc, 1998. ISBN 0471030031.

WATKINS, C.; DAYAN, P. Q-learning. *Machine Learning*, p. 279–292, 1989.

WESTON, J.; WATKINS, C. Multi-class support vector machines. *URL citeseer.ist.psu.edu/article/weston98multiclass.html*, 2003.

WIDROW, B. Generation and information storage in networks of adaline neurons. *SelfOrganizing Systems*, p. 435–461, 1962.

WINK, O.; NIESSEN, W.; VIERGEVER, M. Minimum cost path determination using a simple heuristic function. In: SANFELIU, A. et al. (Ed.). *Image, speech and signal processing*. Los Alamitos: IEEE computer society press, 2000. v. 3, p. 1010–1013.

YANN, L. et al. Energy-based models: Structured learning beyond likelihoods. *Neural Information Processing systens Foundation*, 2006.