# INTELIGÊNCIA ARTIFICIAL PARA JOGOS DE TABULEIRO

Marcelo Nunes de Carvalho

DISSERTAÇÃO DE MESTRADO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
UNIVERSIDADE DE SÃO PAULO

Curso: Ciência da Computação
Área de Concentração: Inteligência Artificial
Orientador: Flávio Soares Corrêa da Silva

– São Paulo, SP – Março de 2004 –

# Resumo

Neste trabalho, foram estudadas técnicas modernas de Inteligência Artificial para jogos de tabuleiro de 2 jogadores, não-triviais, de soma-zero, populares e que possuam requisitos de habilidade, bem como o histórico da área e como os algoritmos utilizados são aplicados à área de jogos e que problemas eles resolvem. Para tanto, o trabalho é dividido em uma abordagem orientada ao problema solucionado: Aberturas, onde são descritas técnicas que se fazem presentes nos primeiros momentos de um jogo; Meio de jogo, onde são mostradas técnicas que realmente simulam a inteligência do jogo, onde é discutido o paralelo da busca e do conhecimento, e são mostrados algoritmos de busca seqüenciais e paralelos; Fim de jogo, onde são descritas técnicas utilizadas para vencer na etapa final de um jogo de tabuleiro convergente; e Aprendizado, onde são discutidas questões relativas ao aprendizado do controle de busca, das funções de avaliação (supervisionado, comparativo e por reforço), de padrões, de planejamento e de modelagem de oponente.

# Abstract

In this work, were studied modern techniques of Artificial Intelligence for board games that are two-player, non-trivial, zero-sum, popular and require ability, as well as the historic of the area and how the algorithms are applied to the game area and which problems they solve. For that, the work is divided into a problem-oriented approach: Openings, where the techniques presented in the first moments of a game are described; Middlegame, where techniques which actually simulate the intelligence of the game are described, the search versus knowledge trade-off is discussed, and sequential and parallel algorithms are shown; Endgame, where techniques used to win at the final phase of convergent games are described; and Learning, where are discussed aspects related to the learning of search control, of evaluation functions (supervised, comparison and reinforcement), of patterns, of planning and opponent modelling.

# Agradecimentos

Agradeço ao meu orientador, Dr. Flávio Soares Corrêa da Silva, por sua ajuda, sem a qual esse trabalho não seria possível.

Aos professores de Mestrado do IME/USP pelos ensinamentos passados.

A minha família, por tudo que são para mim.

A todas as pessoas que, de alguma forma, colaboraram com a realização desse projeto.

E agradeço principalmente a Deus por mais esta oportunidade na minha vida.

# Índice

# Capítulo 1

# Introdução

Os jogos de computador exercem um grande fascínio pelas suas possibilidades de entretenimento que podem ser simples (como testes de habilidade manual) ou até bastante complexas (como grandes desafios intelectuais). Eles diferem de outros softwares pela sua subjetividade, ou seja, não são precisos, pois seu objetivo é divertir (essa diversão pode vir associada a outros objetivos como educar, treinar e selecionar). Mesmo os que são baseados em jogos reais totalmente determinísticos são subjetivos uma vez que a interface e o nível de características de jogo são itens que se baseiam em mera concepção do criador e são ajustados de forma a satisfazer o jogador.

Este capítulo inicia com uma discussão sobre a taxonomia de jogos de computador. Em seguida, é descrita a relação entre os jogos de computador e as pesquisas realizadas em Inteligência Artificial. Nas seções seguintes, são discutidos assuntos relativos aos jogos de tabuleiro, foco desta dissertação, tais como propriedades, espaço de estados, árvore de jogo e resolução, que servirão de referência para os capítulos que se seguem. Por fim, é apresentada uma descrição de como esta dissertação foi dividida.

## 1.1 Taxonomia de jogos de computador

Jogos podem ser classificados segundo muitos critérios diferentes. As categorias utilizadas comercialmente são bastante heterogêneas. Existem diversos atributos que isoladamente classificam os jogos. De acordo com a quantidade de jogadores, existe uma divisão entre 1 jogador, 2 jogadores e mais do que 2 jogadores. Jogos com mais de 2 jogadores envolvem negociação ou cooperação. De acordo com o fator sorte, os jogos podem variar de um jogo sem nenhum fator sorte (Damas, Xadrez, etc) até um jogo completamente determinado pela sorte (Caça-níqueis, Bingo, etc). Outras questões podem ser levantadas para classificar os jogos tais como: Quão profunda é a estratégia? Qual a dificuldade de aprendizado das regras? O jogo é relativamente abstrato ou tenta simular algum aspecto de realidade? Os jogadores são eliminados com o passar do jogo?

Apesar de tudo isso, não existe uma classificação única e universal dos jogos. Ainda assim, abaixo estão listadas 2 sugestões de classificação segundo o gênero.

Esta primeira classificação foi criada pelo autor desta dissertação através da compilação de diversas categorizações encontradas comercialmente em sites de grandes publicadores membros da ESA [32]:

- *Ação/arcade*: O usuário controla um personagem (humano, animal, objeto) que se movimenta, efetua ações e interage com o ambiente. Aqui estão incluídos os FPS ("First Person Shooter"), jogos de tiro em primeira pessoa (Ex: Half-life).

- *Carta*: Como o próprio nome diz, qualquer jogo que possa ser jogado utilizando baralhos (Ex: Truco).
- *Cassino*: Jogos de azar onde os usuários perdem estatisticamente um pequeno percentual de tudo que apostam (Ex: Bingo)
- *Corrida*: O usuário controla um veículo e enfrenta outros usuários ou NPCs (personagens não controlados pelo usuário) em um determinado trajeto objetivando chegar na frente dos demais.
- *Educativo*: usando a diversão como facilitador, esse tipo de jogo desenvolve habilidades no seu usuário tais como raciocínio lógico ou memória visual (Ex: Coelho sabido)
- *Esporte*: Jogos que são baseados em esportes do mundo real (Ex: FIFA soccer).
- *Estratégia*: O usuário controla múltiplos personagens em múltiplas localidades e precisa conceber operações em planos de conjunto (Ex: Warcraft).
- *Puzzle*: Jogo onde não há adversários. O objetivo é solucionar um determinado problema (Ex: Resta 1).
- *Simulador*: Jogos que abstraem os caracteres de uma atividade real específica e os reproduzem em um mundo virtual (Ex: Flight Simulator).
- *Tabuleiro*: Jogos baseados nos jogos reais jogados em tabuleiros, onde cada participante joga somente na sua vez e costuma ser representado por marcas ou peças (Ex: Xadrez).
- *Trívia/Quiz*: Jogos de perguntas e respostas (Ex: Show do Milhão)

Cada uma dessas categorias oferece um desafio diferente para o usuário e para o criador. Comercialmente, estes jogos também são classificados segundo o público alvo (infantil, família, casual e "heavy user") ou segundo a plataforma em que executam (PC, Macintosh, videogame, PDA, celular, DVD e fliperama).

A segunda classificação de jogos desta dissertação é proposta por Anderson/Moore e Brian Sutton-Smith [78]. A idéia aqui é classificar os jogos segundo o nível de desafio/habilidade requerido (cálculos abstratos, sorte, coordenação, velocidade, blefe, verbalização, anagramatização, etc). Neste caso, os jogos podem ser classificados em:

- Jogos de Habilidade: cujo resultado é determinado principalmente por habilidades físicas e mentais, em vez de sorte.

- Jogos de Sorte: cujo resultado é fortemente influenciado por algum dispositivo aleatório (dados, roletas, etc).

- Jogos de Estratégia: cujo resultado é influenciado por interação entre o ambiente e os jogadores.

- Jogos de estados: correspondem a uma combinação das três categorias anteriores.

A primeira classificação tenta ser mais específica, enquanto a segunda é mais abrangente. Nenhuma das duas formas de categorizar está livre do problema de um jogo poder estar em mais de uma categoria. Devido à subjetividade dos jogos, é possível

acreditar que, para qualquer classificação criada, sempre poderá ser criado um novo jogo que esteja em mais de uma categoria.

## 1.2 – Jogos e Inteligência Artificial

A área de jogos e a área de Inteligência Artificial (IA) têm uma longa tradição de contribuição entre elas mútua. Os jogos proporcionam para a IA uma grande variedade de problemas a serem solucionados, são processados em um domínio restrito e seus resultados são geralmente fáceis de avaliar [56]. Jogos podem conter um mundo totalmente dinâmico, mas a grande diferença do desafio de IA dos mesmos é que eles precisam ser executados em tempo real (no caso de jogos de ação, por exemplo) ou, no pior caso, responder rapidamente (no caso dos jogos de tabuleiro). A IA, por sua vez, proporciona para os jogos recursos técnicos para construção de interessantes atributos, tais como agentes mais realistas para interagir com usuários humanos e imprevisibilidade de comportamento. A IA também acaba por contribuir com a comunidade de jogadores de jogos de tabuleiro, uma vez que descobertas realizadas em estudos podem passar a fazer parte da literatura do jogo. Um exemplo da contribuição dos jogos para a IA é o projeto "Robocup", um campeonato mundial de futebol de robôs. Um exemplo da contribuição da IA para os jogos é o jogo "FIFA Soccer" da Eletronic Arts.

Nesta dissertação, os esforços foram concentrados no estudo de IA para jogos de tabuleiro. A escolha desse estudo deve-se ao fato desses jogos serem uma forma pura e abstrata de competição que requer inteligência, sendo também de fácil representação em termos de ação e estados. Os jogos dessa natureza também são tópicos desse tipo de estudo desde os primórdios da IA há décadas atrás. Uma das possíveis razões de tanto estudo é que um computador vencendo um humano em um jogo dessa natureza seria supostamente a prova de uma máquina executando uma tarefa que requer inteligência. É o caso, por exemplo, dos programas que jogam Xadrez que evoluíram tanto com o passar dos anos que uma máquina (IBM – "Deep Blue") foi capaz de derrotar um campeão mundial humano. O desempenho do "Deep Blue" é normalmente creditado ao seu poderoso hardware, que também é impressionante, mas a chave para o sucesso do mesmo é a combinação da velocidade do hardware com sofisticados algoritmos para busca em árvore, avaliação de posição e geração de movimentos [41]. Outro campeão mundial, neste caso no jogo de Damas, é o "Chinook" (produzido pela Universidade de Alberta). Mesmo o nível fácil da versão simplificada deste programa já oferece um grande desafio para um bom jogador [82].

Existem 3 diferentes formatos para um programa que implementa jogos de tabuleiro. O primeiro é o de representação tabular, onde a informação do jogo é previamente produzida e fica armazenada em um banco de dados. O segundo é o de representação algorítmica compacta, onde não há informação além das regras do jogo e a busca determina a decisão de onde jogar. O terceiro é a representação por conselho baseado no conhecimento (similar ao modelo cognitivo humano de jogo) e é excelente para detalhar o porquê de uma determinada jogada, mas costuma ser o mais fraco [53]. A maioria dos programas campeões mundiais costuma utilizar os 3 formatos acima. Os resultados decorrentes de cada um desses formatos será melhor esclarecido no decorrer desta dissertação.

O estudo de jogos de tabuleiro foi também motivado, entre outros fatores, pela vontade dos pesquisadores de criar programas de jogos que pudessem derrotar os campeões humanos. Outros fatores também importantes foram os fatores comerciais (por exemplo, a promoção da marca IBM no caso do "Deep Blue"), estudo de aspectos matemáticos ou sociológicos e o interesse na resolução dos jogos.

Para os jogos mundialmente tradicionais como Damas e Xadrez, os programas já atingiram o estado da arte. Era possível que isso acabasse por desmotivar os pesquisadores a prosseguir estudando técnicas uma vez que a meta principal já foi atingida e que não seria tão interessante resolver um jogo que fosse criado somente com esse objetivo e não fosse do interesse de um grande público, mas o jogo de "Go", um jogo muito popular no Japão, manteve o desafio aceso. Isso porque a maioria dos programas utiliza a força bruta (busca por exaustão) para jogar percorrendo vários níveis da árvore de próximas jogadas possíveis do jogo, mas essa estratégia não tem muito resultado no jogo de "Go" por causa do seu fator de ramificação (número de jogadas possíveis na vez de um jogador) ser muito alto (cerca de cem vezes maior que o de Xadrez) e portanto, a árvore de busca possui muito mais nós a cada nível para serem avaliados pelo programa. O fato é que novas técnicas são estudadas para o jogo de "Go" e acabam sendo aproveitadas para outros jogos também.

Os principais algoritmos de sucesso para a estratégia da força bruta foram desenvolvidos há muito tempo que é o caso do Minimax e do corte Alpha-Beta, utilizado para diminuir a quantidade de nós avaliados sem o risco de informação útil não ser considerada. Desde então foram sugeridos inúmeros novos algoritmos sendo que a maioria caiu por chão no momento dos testes comparativos com os primeiros. É o caso, por exemplo, do Scout [1] que, segundo os próprios autores, é bem menos eficiente do que o Alpha-Beta. Esses algoritmos que caíram em desuso não serão assunto desta dissertação por não oferecerem nenhum avanço direto para a área.

O que realmente continuou avançando foi a criação de heurísticas de sucesso específicas para cada jogo, inclusive para realizar cortes na árvore que diminuam a quantidade de posições avaliadas e garantam o máximo de segurança nesses cortes. Essas heurísticas permitem soluções mais eficientes, mas estão sujeitas a falhas. Por não serem técnicas gerais de utilização em jogos de tabuleiro e sim, um conhecimento específico de um determinado jogo e por existirem aos milhares, essas heurísticas terão uma participação coadjuvante nesta dissertação sendo citadas e exemplificadas por amostragem.

A Inteligência artificial, por sua vez, acabou contribuindo com as comunidades de jogadores, uma vez que os estudos realizados em cima dos jogos mostraram a existência de novas possibilidades interessantes de jogo. É o caso de jogos como o Othello e o Gamão onde pesquisadores relataram situações em que os grandes mestres do jogo aprenderam novas estratégias para vencer, enfrentando programas jogadores. Este tipo de conhecimento acaba entrando para a literatura desses jogos e abrindo novos horizontes.

## 1.3 – Classificação e propriedades dos jogos de tabuleiro

Os jogos de tabuleiro podem ser classificados pela quantidade de jogadores (1,2 ou mais jogadores), pela informação (perfeita ou imperfeita) que os jogadores possuem e pelo fator sorte. Os jogos, segundo o fator sorte, podem ser classificados em: determinísticos, onde não existe o elemento de sorte (Xadrez, Damas, Go, Othello, etc); e probabilísticos, onde existe o elemento de sorte (Gamão, Ludo, etc). No entanto, se forem considerados os

artigos publicados nas últimas conferências bienais "Computer and Games", a maior concentração de esforços dos pesquisadores de jogos de tabuleiro permanece no estudo de jogos determinísticos de informação perfeita com 2 jogadores. Isso provavelmente porque não incluem o fator sorte e não necessitam de cooperação.

Esta seção descreve algumas importantes propriedades dos jogos de tabuleiro. Essas informações serão utilizadas para descrever mais especificamente o escopo desta dissertação ainda neste capítulo e também serão utilizadas no decorrer do texto para explicar quais tipos de técnicas se aplicam a quais jogos.

- **Número de jogadores**

Corresponde ao número total de jogadores do jogo. Alguns jogos permitem diferentes quantidades de jogadores. Costumam ser divididos em 3 categorias: 1 jogador, 2 jogadores ou múltiplos (mais de 2) jogadores. Alguns costumam considerar a categoria 0 jogadores (Exemplo: "Conway's life").

- **Soma-zero**

Corresponde a jogos onde a derrota de um significa a vitória do outro. Considerando o "dilema do prisioneiro" como um jogo, este não é soma-zero.

- **Trivialidade**

Corresponde a jogos onde a melhor estratégia de jogo pode ser estabelecida trivialmente por enumeração ou análise matemática (Exemplo: Jogo da velha).

- **Popularidade**

Corresponde a jogos que foram jogados por um grande número de pessoas em vários países do mundo. Muitos jogos matemáticos e variações obscuras de jogos conhecidos são considerados não-populares.

- **Habilidade**

Corresponde a jogos onde existe uma forte relação entre habilidade do jogador e suas chances de vitória, mesmo que haja influência do fator sorte. Alguns jogos são apenas um passatempo e não requerem habilidade do jogador.

- **Fator Sorte**

Segundo a existência ou não do fator sorte, os jogos podem ser classificados em determinísticos ou probabilísticos.

- **Informação**

Segundo a informação, os jogos de tabuleiro podem ser classificados em jogos de informação perfeita ou jogos de informação imperfeita. Em um jogo de

informação perfeita, a qualquer momento do jogo, todos os jogadores têm acesso a todas as informações que definem o estado do jogo e suas possíveis continuações. Todos os outros jogos são considerados de informação imperfeita. A tabela a seguir mostra alguns jogos classificados segundo a informação e o fator sorte.

| | **Determinísticos** | **Probabilísticos** |
|---|---|---|
| **Informação perfeita** | Xadrez, Damas, Othello, Trilha | Gamão, Ludo |
| **Informação imperfeita** | | Scrabble |

Tabela 1 – Exemplos de jogos segundo informação e fator sorte

- **Convergência**

  Dividindo o espaço de estados de todas as posições legais de um jogo em classes disjuntas, onde cada classe contém todas as posições com um mesmo número de peças no tabuleiro.

  Definindo um grafo orientado G no qual cada classe é um nó, e um arco existe entre as classes A e B se e somente se uma posição P existe em A tal que um movimento existe de P levando a uma posição Q em B.

  Um jogo converge se, para a maioria dos arcos de A para B em G, a cardinalidade de A é maior do que a cardinalidade de B. Um jogo diverge se, para a maioria dos arcos de A para B em G, a cardinalidade de B é maior do que a cardinalidade de A. Um jogo é considerado imutável quando o mesmo não converge nem diverge.

- **Morte súbita**

  Corresponde a jogos que podem encerrar abruptamente pela aparição de um padrão de um conjunto de padrões pré-especificados. Um exemplo de jogo que possui a propriedade de morte súbita é o Go-moku: o jogo é finalizado se um dos jogadores criar uma linha com cinco peças de sua cor. Já no caso do jogo de Othello não é assim, o jogo só termina quando acabam os movimentos de ambos jogadores ou um dos jogadores não possui mais discos no tabuleiro.

- **Complexidade**

  Esta propriedade, no que diz respeito a jogos de tabuleiro, pode denotar duas medidas diferentes: complexidade do espaço de estados e complexidade da árvore de jogo. Essas medidas são melhor compreendidas na seção seguinte.

## 1.4 – Espaço de estados e Árvore de jogo

O Espaço de estados de um jogo corresponde ao conjunto de todas as posições de um jogo que podem ser obtidas através de movimentos válidos a partir da posição inicial. O número de posições existentes no espaço de estados corresponde à sua complexidade [3].

A árvore de jogo é um termo matemático que se refere a um grafo direcionado ilustrando todas as posições possíveis em um determinado jogo, diagramando o modo como o jogo segue de posição a posição à medida que é jogado [86]. A complexidade de uma árvore de jogo corresponde ao número total de folhas da árvore. Observar que a complexidade da árvore de jogo pode ser maior que a complexidade do espaço de estados uma vez que podem existir folhas iguais em diferentes pontos da árvore.

Para exemplificar, uma árvore de jogo do jogo da velha inicia com 9 ramos, porque o primeiro jogador pode escolher qualquer um dos 9 espaços. O segundo jogador pode escolher, para cada jogada do anterior, 8 espaços. Isso significa para um primeiro movimento completo, uma árvore de jogo com 9x8=72 folhas. Seguindo esta conclusão, a árvore completa teria 9!=362.880 folhas, alguns delas significando vitória para o primeiro jogador, outras para o segundo e outras, empate. Esta conclusão, no entanto, corresponde apenas a uma estimativa por alto, uma vez que o jogo pode ser ganho com 5 jogadas, se o primeiro jogador colocar todas as marcas em uma única linha. Muitos ramos, portanto, não continuam até o nível 9 da árvore.

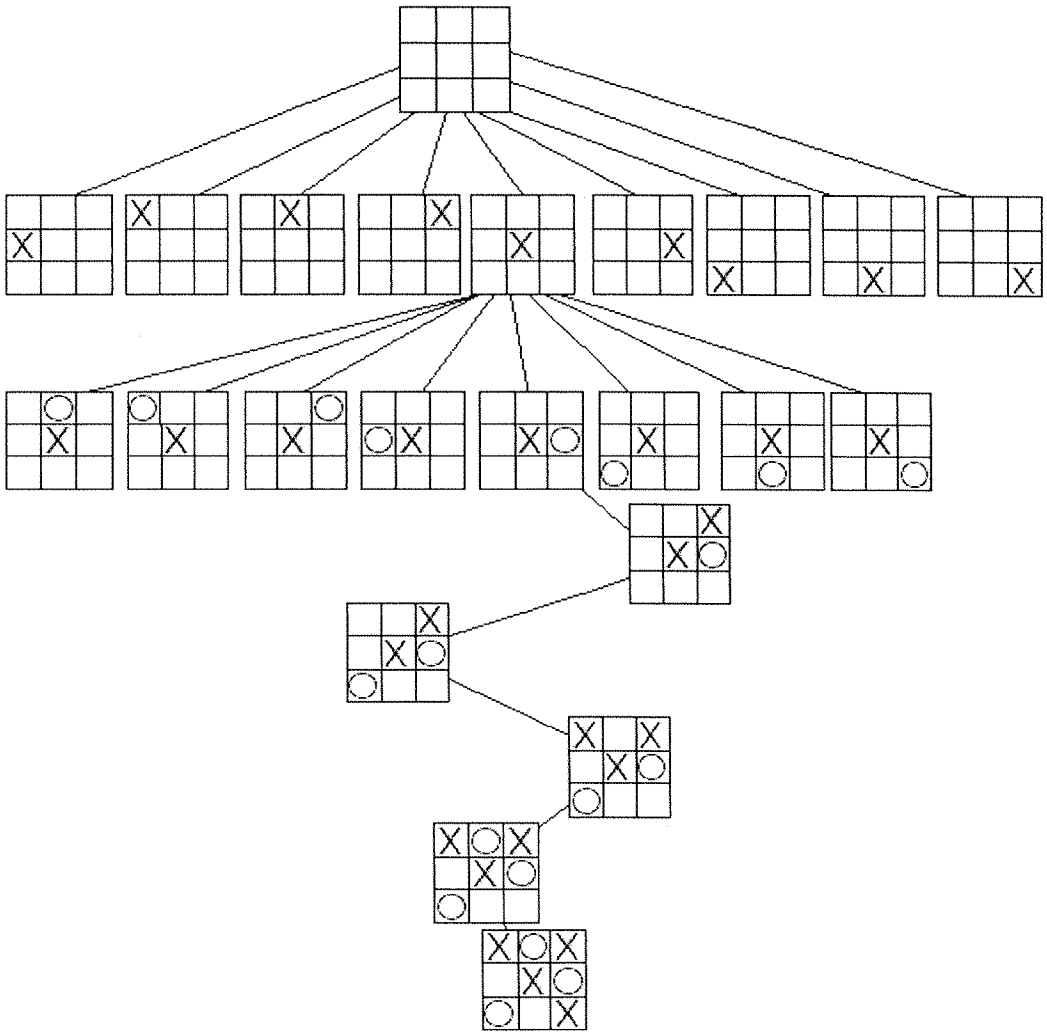


Figura 1 – Árvore de jogo do jogo da velha

Usando o mesmo exemplo do jogo da velha para a complexidade do espaço de estados chega-se ao número de 5.478 posições legais [3]. A definição dessa complexidade pode ser refinada se for considerado que posições simetricamente equivalentes são contadas como uma só.

Abaixo segue uma tabela com estimativas das complexidades do espaço de estados e da árvore de jogo de alguns jogos de tabuleiro ($\log_{10}$). Uma outra coluna foi acrescentada para caracterizar em qual classe de problema computacional esses jogos se encontram. Para poder comparar os jogos adequadamente, é importante lembrar que:

LOGSPACE $\subseteq$ PTIME $\subseteq$ PSPACE $\subseteq$ EXPTIME $\subseteq$ EXPSPACE

| **Jogo** | **Espaço de estados** | **Árvore de jogo** | **Classe** |
|---|---|---|---|
| Trilha | 10 | 50 | |
| Mancala | 12 | 32 | |
| Lig-4 | 14 | 21 | |
| Gamão | 20 | 144 | |
| Damas | 21 | 31 | EXPTIME-completo |
| Othello | 28 | 58 | PSPACE-completo |
| Xadrez | 46 | 123 | EXPTIME-completo |
| Shogi | 71 | 226 | EXPTIME-completo |
| Go | 172 | 360 | EXPTIME-completo |

Tabela 2 – Logs das complexidades dos jogos

Uma observação importante é a disparidade da complexidade da árvore de jogo e do espaço de estados em alguns jogos, como por exemplo, o jogo de trilha. Isso costuma significar uma maior dificuldade para se produzir algoritmos de meio de jogo com bons resultados. Isso será discutido em maiores detalhes no capítulo específico.

Os programas jogadores utilizam as idéias de espaço de estados e de árvore de jogo para implementar diferentes técnicas. A forma como essas informações são representadas costuma variar podendo ser utilizada a forma de grafos, árvores ou tabelas, dependendo de requisitos de armazenamento e desempenho. Em cada capítulo, são discutidas as formas de representação mais utilizadas para as técnicas em questão.

## 1.5 – Resolução de Jogos de tabuleiro

Resolver um jogo usualmente indica que uma propriedade, no que diz respeito ao resultado final do jogo, foi determinada. São consideradas ao menos três definições para a resolução de um jogo soma-zero de 2 jogadores com informação perfeita (os 2 primeiros termos foram sugeridos por Paul Colley, enquanto o terceiro termo foi sugerido por Donald Michie) [3]:

- Ultrafraca

Nesta definição, resolver o jogo significa provar se, considerando jogadas perfeitas de ambos os lados, o primeiro jogador ganha, perde ou empata o jogo a partir da posição inicial.

- Fraca

Neste caso, resolver um jogo significa prover um algoritmo que assegure a vitória para um jogador, ou o empate para ambos, contra qualquer seqüência de movimentos a partir da posição inicial.

- Forte

Nesta definição, resolver um jogo significa criar um algoritmo que possa produzir jogadas perfeitas a partir de qualquer posição, mesmo que erros já tenham sido cometidos por um ou por ambos os jogadores.

Resolver um jogo costuma ser o objetivo de longo prazo de uma parte dos programas jogadores. A idéia é que o programa evolua a tal nível que seja possível jogar perfeitamente a partir da posição inicial (resolução fraca).

Enumerar todo o espaço de estados de um jogo por força-bruta é a maneira mais comum de prover uma resolução forte para um jogo. Conhecendo todos os estados, é possível para o programa jogar perfeitamente a partir de qualquer estado até o final do jogo. O capítulo de fim de jogo discute técnicas que, se estendidas, permitem enumerar uma grande parte do espaço de estados no final do jogo. Nesse caso, um algoritmo de busca deveria cuidar do início de jogo para garantir a resolução do mesmo. Claro que nem todos os jogos foram resolvidos apenas por enumeração, alguns foram resolvidos por programas baseados no conhecimento matemático humano (é o caso do Lig-4 que foi resolvido por Victor Allis em 1988).

Abaixo segue uma lista de alguns jogos de tabuleiro já resolvidos e uma lista com alguns que ainda não foram resolvidos.

**Jogos resolvidos**

- Mancala

Qualquer jogador pode forçar um empate (Resolvido por Henry Bal e John Romein na Free University em Amsterdam, Holanda em 2002).

- Lig-4

O primeiro jogador pode forçar a vitória (Resolvido por Victor Allis em 1988 e por James Allen em 1989 independentemente).

- Go-moku

O primeiro jogador pode forçar a vitória (Resolvido por Victor Allis em 1992).

- Hex

O primeiro jogador pode forçar a vitória para qualquer tamanho de tabuleiro N x N (Resolvido por John Nash).

- Trilha

Qualquer jogador pode forçar o empate (Resolvido por Ralph Gasser em 1993).

**Jogos não resolvidos**

- Damas

Diferente da crença popular, o jogo de Damas ainda não foi resolvido. Apesar disso, quase todas as posições de meio de jogo já foram resolvidas.

- Xadrez

Apenas resolvido para configurações de fim de jogo até 5 peças.

- Go

Resolvido apenas para tabuleiros 4x4. O jogo é jogado usualmente por humanos em um tabuleiro 19x19.

- Othello

Resolvido apenas para tabuleiros 6x6. O jogo real é jogado em um tabuleiro 8x8.

Independente do fato de ainda não terem sido resolvidos, já existem programas jogadores de Damas, Xadrez e Othello capazes de superar qualquer jogador humano, mesmo os grandes campeões mundiais. Apenas o jogo de Go permanece em um nível inferior aos mestres do jogo.

## 1.6 – Objetivo

A proposta deste trabalho é estudar técnicas modernas de inteligência artificial para jogos de tabuleiro, descrever o histórico dessa área e apresentar algoritmos usados atualmente, como eles são aplicados à área de jogos e que problemas eles resolvem. Apresentar também as contribuições da área de Inteligência Artificial para a área de jogos e vice-versa. Produzir, no decorrer do estudo, uma estrutura adequada para organizar o conhecimento da área.

### 1.6.1 Escopo da dissertação

Visando restringir o estudo de jogos de tabuleiro, uma vez que se trata de um extenso tópico de inteligência artificial, foram escolhidas algumas propriedades a serem satisfeitas pelos jogos aqui estudados, garantindo dessa forma um melhor foco em cima do estudo.

Os jogos desta dissertação satisfazem as seguintes propriedades:

- Número de jogadores: 2

Ficam fora desta dissertação, portanto, jogos que são jogados isoladamente e jogos com mais de 2 jogadores. O objetivo aqui é excluir jogos que não necessitam de adversário, além de excluir questões relativas à cooperação.

- Soma-zero

O objetivo dessa restrição é completar a exclusão das investigações sobre questões relativas a cooperação entre jogadores, uma vez que, para dois jogadores, essa propriedade significa que a vitória de um corresponde à derrota do outro. Ou seja, os jogadores não podem cooperar entre si.

- Não-trivialidade

Exclui do escopo desta dissertação jogos que podem ser resolvidos por análise matemática ou enumeração simples. A maioria dos jogos desta dissertação está classificada em termos de classe de complexidade como, pelo menos, PSPACE-difícil (caso do jogo de Hex). A título de exemplo, jogos como o jogo da velha ou algumas variantes do jogo de Nim ficam de fora.

- Popularidade

Exclui do escopo desta dissertação jogos pouco conhecidos, jogos matemáticos inventados para desafiar a capacidade dos algoritmos e também variações de jogos conhecidos. Isso significa que estão incluídos jogos muito pesquisados no mundo inteiro. O Xadrez give-away é um dos exemplos de jogos que ficam de fora desta dissertação, por ser uma variação obscura do jogo de Xadrez.

- Habilidade

Estão excluídos aqui os jogos apenas de passatempo, em que jogadores experientes não têm nenhuma vantagem sobre os novatos. O jogo "Scotland Yard" da Grow é um exemplo de jogo que está fora desta dissertação.

Apesar da lista de propriedades criadas aqui, muitas das técnicas citadas nesta dissertação, podem ser utilizadas com sucesso em jogos que não satisfazem as propriedades acima. Mais do que isso, essas técnicas podem ser utilizadas em diversos jogos que não são

de tabuleiro, como, por exemplo, alguns jogos de cartas. Algumas das técnicas podem até mesmo ser aplicadas em outras áreas que não jogos.

### 1.6.2 Lista dos jogos que satisfazem essas propriedades

Considerando as propriedades listadas na seção anterior, foi elaborada uma lista de jogos de tabuleiro que satisfazem todas essas propriedades: Mancala, Xadrez, Xadrez chinês, Damas, Lig-4, Go, Go-moku, Trilha, Othello, Renju, Hex, Scrabble, Alquerque, Gamão, Ludo e Nim. A lista criada não tem a pretensão de ser completa, mas os jogos listados já oferecem um nível de desafio suficiente. Estes jogos estão descritos no Apêndice A de forma conceitual para garantir a compreensão dos textos nos momentos em que são citados.

Esses jogos correspondem a jogos de tabuleiro em que são pertinentes as discussões de busca e de conhecimento feitas nesta dissertação. A maioria desses jogos também foi alvo de inúmeras pesquisas e avanços no sentido da produção de programas jogadores que são capazes de derrotar os seus correspondentes humanos.

A tabela 3 exibe a lista dos jogos desta dissertação classificados segundo a informação e o fator sorte.

| **Jogo** | **Informação** | **Fator sorte** |
|---|---|---|
| Alquerque | perfeita | determinístico |
| Amazons | perfeita | determinístico |
| Damas | perfeita | determinístico |
| Gamão | perfeita | probabilístico |
| Go | perfeita | determinístico |
| Go-moku | perfeita | determinístico |
| Hex | perfeita | determinístico |
| Lig-4 | perfeita | determinístico |
| Ludo | perfeita | probabilístico |
| Mancala | perfeita | determinístico |
| Nim | perfeita | determinístico |
| Othello | perfeita | determinístico |
| Renju | perfeita | determinístico |
| Scrabble | imperfeita | probabilístico |
| Shogi | perfeita | determinístico |
| Trilha | perfeita | determinístico |
| Xadrez | perfeita | determinístico |
| Xadrez chinês | perfeita | determinístico |

Tabela 3 – Jogos classificados segundo a informação e o fator sorte

## 1.7 – Divisão da dissertação

Como as técnicas e algoritmos relativos a jogos de tabuleiro mudam consideravelmente dependendo do momento do jogo, a divisão em capítulos desta

dissertação é similar às etapas de um jogo de tabuleiro qualquer (para cada capítulo, será eleito um jogo diferente para ser avaliado de forma um pouco mais detalhada):

Aberturas (capítulo 2), onde são descritas técnicas que se fazem presentes nos primeiros momentos de um jogo. São discutidas as diferentes formas de construção e suas conseqüências para o nível de qualidade do jogo. Adicionalmente são comentadas as formas de representação e armazenamento.

Meio de jogo (capítulo 3), onde são descritas técnicas que realmente simulam a inteligência do programa, esta é a maior e principal parte desse estudo. São mostrados os algoritmos seqüenciais e paralelos responsáveis pelo sucesso dos jogos. Também são comentados os avanços recentes e o que é esperado para o futuro.

Fim de jogo (capítulo 4), onde são descritas técnicas que os programas utilizam para vencer na etapa final do jogo. É discutida a principal técnica, a construção de bancos de dados de fim de jogo. Adicionalmente são discutidos formas de representação, armazenamento, compactação e acesso a esses bancos de dados.

Após isso, são descritas técnicas de Aprendizado (capítulo 5), onde são apresentadas algumas promissoras técnicas e outras já bem-sucedidas da área. São discutidas questões desde aprendizado de funções de avaliação e seus parâmetros até questões como modelagem de oponente.

Por fim, são feitas as considerações finais (capítulo 6) e sugestões para trabalhos futuros nesta área.

Considerando que nem todos os jogos que fazem parte do escopo desta dissertação são do conhecimento de todos, foi incluído um apêndice descrevendo de maneira conceitual os jogos citados aqui (Apêndice A).

# Capítulo 2

# Aberturas

A maioria dos jogos de tabuleiro possui um espaço de estados grande o suficiente para que as primeiras jogadas de um jogo qualquer não indiquem um vencedor. A experiência humana, no entanto, produziu uma vasta literatura para os diferentes jogos mostrando que as primeiras jogadas de um jogo podem determinar uma pequena vantagem para um dos jogadores. Isso significa que um bom início de jogo pode favorecer o jogador no meio do jogo. Essa informação foi incorporada nos programas jogadores na forma de um livro de aberturas.

Um livro de aberturas corresponde a um banco de dados de informações pré-calculadas sobre as primeiras jogadas possíveis de um jogo. Através de valores, são indicadas situações que podem levar a alguma vantagem e situações que devem ser evitadas por correr o risco de levar a alguma situação de desvantagem. A cada jogada do adversário, a máquina consulta essas informações e decide a sua melhor jogada. Isso somente nas primeiras jogadas, pois as decisões após isso são geradas por estratégias de meio de jogo descritas no próximo capítulo.

Os livros de aberturas costumavam ser produzidos manualmente por mestres de cada jogo, armazenando bons movimentos sugeridos pela teoria, ou simplesmente listando todos os jogos já jogados por grandes jogadores. O interesse dos pesquisadores da área recentemente mudou para a construção automática de livros de abertura [51].

A maioria dos programas campeões de jogos de tabuleiro utiliza um livro de aberturas e atualiza este livro automaticamente após cada jogo. Isso porque as informações contidas em um livro de aberturas ou são resultado de heurísticas (construção automática) ou resultado de estudo da literatura humana do jogo (construção manual), onde ambas são passíveis de erro. A cada jogo, o programa deve reavaliar se a abertura utilizada foi um acerto ou um erro e modificar a avaliação da mesma.

Utilizar um livro de aberturas em um jogo costuma trazer outras vantagens, além de melhorar as possibilidades de vitória. Por consultar um banco de dados, o programa joga rapidamente e economiza tempo de jogo para jogadas posteriores.

Este capítulo descreve as técnicas de construção, representação e acesso a bancos de dados de aberturas. Apesar de se utilizarem técnicas similares em todos os jogos, a aplicação das mesmas costuma diferir de jogo para jogo devido às diferenças de propriedades e regras. Por exemplo, no jogo de Shogi, a utilização de um livro de aberturas não é tão efetivo como no jogo de Xadrez. Isso se deve principalmente ao fato de o jogo de Shogi ter um fator de ramificação e uma complexidade de árvore de jogo bem maior que o Xadrez. Ou seja, em um mesmo espaço de armazenamento, muito mais níveis de profundidade podem existir para o jogo de Xadrez tornando as aberturas mais efetivas.

Para esse capítulo, foi eleito o jogo de Othelo para ser melhor comentado. O critério de escolha em cada capítulo se deve a alguns fatores principais: as técnicas apresentadas são bastante efetivas para o jogo escolhido e existe uma grande quantidade de publicações

do tema utilizando o jogo escolhido. Se existe mais de um jogo nas mesmas condições, é escolhido o jogo mais popular segundo a impressão do autor desta dissertação (o jogo escolhido não é repetido em outro capítulo). A principal desvantagem dessa abordagem é uma pequena perda de generalidade, uma vez que as técnicas se aplicam de forma semelhante, mas a transcrição da técnica para um outro jogo nem sempre é trivial. A maior vantagem dessa abordagem é aumentar a clareza e a compreensão do texto, uma vez que é possível descrever as técnicas com exemplos práticos. De qualquer forma, outros jogos são citados e comentados no decorrer dos capítulos.

## 2.1 – Construção Manual de livros de aberturas

Inicialmente, muitos programas jogadores não utilizavam livros de aberturas para jogar. Costumavam utilizar as técnicas que hoje são utilizadas apenas no meio do jogo, também para o início. Isso, no entanto, resultava em uma grande desvantagem em relação a grandes mestres humanos do jogo uma vez que eles se valiam de um conhecimento desenvolvido ao longo de dezenas ou centenas de anos.

Apesar da ausência de valor computacional científico, os pesquisadores consideraram válido importar esse conhecimento para dentro de seus programas. Isso foi determinante para muitos programas atingirem o nível de campeões. Por um lado, os programas perdiam a característica de surpresa de utilizar aberturas não-convencionais evitando a previsibilidade de aberturas previamente conhecidas por humanos. Por outro lado, os programas deixavam de cometer erros grosseiros nas aberturas evitando uma derrota prematura.

A técnica de produção manual de livros de aberturas corresponde a laboriosamente digitar milhares de posições ditadas por um grande mestre ou retiradas da literatura específica do jogo [18]. As boas aberturas são consideradas pelo programa e as aberturas ruins são evitadas pelo programa.

A abertura pode ter uma importância maior em alguns jogos. Por exemplo, no jogo de Damas, a ausência de um livro de aberturas tem potencialmente conseqüências muito mais sérias que no jogo de Xadrez [70]. Isso se deve basicamente à natureza dos jogos, uma vez que no Xadrez é possível voltar atrás na jogada seguinte, enquanto no jogo de Damas isso não é possível. Uma situação real a demonstrar a importância das aberturas é a participação do programa Chinook no campeonato de Damas U.S Open. O programa entrou no campeonato sem um livro de aberturas. Nas únicas 2 partidas que o programa perdeu, os oponentes ou espectadores mostraram que ele caiu em uma situação de derrota muito rapidamente. Para que ele conseguisse não cair nessas armadilhas, ele precisaria ver 33 jogadas à frente, o que estava bem acima das suas capacidades. O Chinook voltou a perder outras partidas pela ausência das aberturas. Só conseguiu ser campeão mundial após incorporar um banco de dados de aberturas com 34.000 posições construídas manualmente para o programa Colossus produzido por Martin Bryant. O programa Colossus foi o único programa a derrotar o Chinook (apenas uma vez), mostrando assim a importância dos bancos de dados de aberturas para os programas jogadores.

A limitação de jogar uma abertura de forma similar a um humano foi extinta com a construção automática de bancos de dados de aberturas que já não se valiam mais do conhecimento humano, mas sim das heurísticas. A construção automática é comentada na seção a seguir.

## 2.2 – Construção Automática de livros de aberturas

Existem duas formas de construção automática de livros de aberturas, que são chamadas de construção passiva, onde jogos são coletados e, após análise, são utilizados para atualizar e estender o livro de aberturas; e construção ativa, onde são avaliadas as aberturas segundo uma função heurística e posteriormente seus valores são armazenados no banco de dados[51]. As subseções a seguir explicam com maiores detalhes essas duas formas de construção.

### 2.2.1 – Construção Automática Passiva

Na construção passiva, a idéia é coletar o máximo número de jogos de grandes mestres do jogo. A maioria dos campeonatos de jogos conhecidos é divulgada ou até mesmo colocada na internet na forma usual de representação do jogo. Isso representa um importante material para servir de entrada para os algoritmos de análise desses jogos. Alguns jogos, como o caso do jogo de Mancala, não possuem tanta divulgação. Isso significa que, havendo pouca informação para utilizar como base, a construção de aberturas para esses jogos precisa seguir um outro paradigma.

O princípio básico da construção passiva é que, se uma abertura já foi jogada anteriormente, então ela deve ser boa o suficiente para ser adicionada a um livro de aberturas. Caso uma determinada abertura tenha levado a uma derrota do jogador, esta é adicionada ao banco de dados como uma abertura que deve ser evitada.

A desvantagem apresentada na construção manual, também existe nesta forma de construção. As aberturas são plenamente conhecidas por bons jogadores não existindo o fator surpresa. Existe também a possibilidade de se considerar boa uma abertura ruim.

### 2.2.2 – Construção Automática Ativa

Na construção ativa, é dispensado o conhecimento estratégico do jogo e são utilizadas heurísticas para avaliar as aberturas. Apesar da desvantagem de poder avaliar alguma abertura de forma errada, existe a vantagem da descoberta de novas aberturas e do conseqüente fator surpresa. Um programa que consiga descobrir uma nova boa abertura acaba por colocar o oponente em uma situação que ele desconhece as melhores jogadas e, portanto corre um risco maior de ficar em desvantagem logo no início.

O processo funciona basicamente assim. É criada uma função heurística que recebe como entrada uma situação do tabuleiro e devolve como saída um valor de $-\infty$ a $+\infty$ (ou uma faixa finita). Quanto menor for o valor, significa que pior está a situação do programa e quanto maior, significa maior vantagem para o programa jogador. Essa função é mais conhecida como "Função de Avaliação". Um programa auxiliar realiza o processo de expandir a árvore de jogo a partir da posição inicial. As posições vão sendo avaliadas e seus valores vão sendo armazenados no banco de dados de aberturas, para posterior uso pelo programa jogador, que deverá a cada momento do jogo decidir pela próxima posição cuja avaliação traz vantagem para ele (nem sempre vale a pena usar apenas valores ótimos, pois utilizando valores sub-ótimos, o programa reduz a sua previsibilidade).

A Figura 2 mostra uma situação inicial do jogo de Othello onde as pretas começaram jogando em C5, as brancas jogaram em E6 e as pretas em F3. Supondo as brancas sendo controladas pelo programa jogador, ele precisa decidir entre uma das seguintes jogadas possíveis: E3, C3, B5 ou C4. Uma das grandes curiosidades do jogo de Othello é que, apesar de vencer quem tem mais peças ao final do jogo, ter mais peças no meio do jogo pode significar grande desvantagem. As principais características a serem consideradas por uma função de avaliação são, na verdade, posicionais: estabilidade (discos estáveis são aqueles que não podem ser convertidos em peças do oponente), mobilidade (possibilidades de movimento a cada jogada) e paridade (último movimento em uma dada região do tabuleiro). Mesmo tendo uma boa função de avaliação baseada nas características posicionais comentadas, para que o programa decida da melhor forma sem um livro de aberturas, ele precisaria expandir as possibilidades de jogo para ver algumas jogadas à frente, aplicar as funções de avaliação e escolher dentre as quatro jogadas citadas, qual a que leva a maiores chances de vitória. Por limitações de tempo, ele deverá sempre ver menos jogadas à frente do que realmente veria caso o processamento tivesse sido pré-calculado (sem as mesmas limitações de tempo de um jogo) e armazenado em um livro de aberturas.

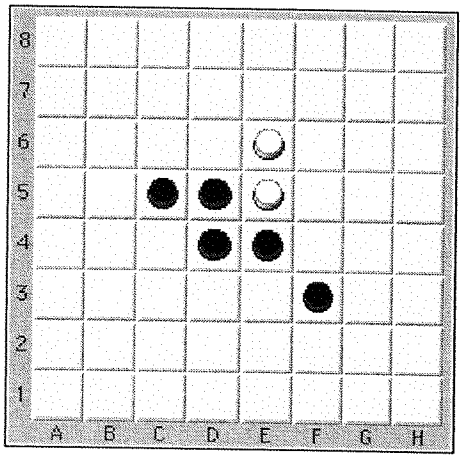


Figura 2 – Situação de abertura do jogo de Othello

É importante observar que o sentido de abertura humano é um sentido diferente da abertura computacional. Na abertura humana, a literatura ensina a decorar uma seqüência de movimentos que trazem vantagens para o jogador. Isso costuma significar que as primeiras X jogadas são decoradas e movimentadas automaticamente pelo jogador. No sentido computacional, não existe um limite X de jogadas. Um livro de aberturas pode ser tão grande quanto seja possível gerá-lo, armazená-lo e acessá-lo, ou seja, se os recursos assim permitirem o programa pode até jogar até próximo do final do jogo se baseando nesse banco de dados.

A forma mais usual de representar um livro de aberturas é através de um grafo dirigido, onde as posições são os nós e os movimentos válidos são os arcos. Um nó chamado de "nó inicial" representa a situação inicial de um jogo, e todos os outros nós tem que ser atingidos partindo dele. Se um nó possui uma extremidade para cada um dos seus movimentos, ele é chamado de "nó interior", de outra forma ele é chamado de "nó folha" [51].

Na figura 3, é exibida uma representação de um livro de aberturas. Cada nó i, tem 2 atributos: o valor heurístico $h_i$ (retornado pela função de avaliação aplicada ao nó) e o valor propagado $p_i$. Para nós interiores, $p_i$ é o valor "negamax" de $p_{sj}$ de todos os nós sucessores $s_j$. Para nós folha, $p_i$ é igual à $h_i$. Negamax é um algoritmo similar ao Minimax (explicado no capítulo seguinte) onde ambos os lados tentam maximizar o resultado, em vez de um dos lados tentando minimizar enquanto o outro tenta maximizar. Para nós interiores, não é mais preciso armazenar o valor $h_i$.

$$p_i = \begin{cases} \max_{sj} (- p_{sj}) & \text{para nós interiores} \\ h_i & \text{para nós folha} \end{cases}$$

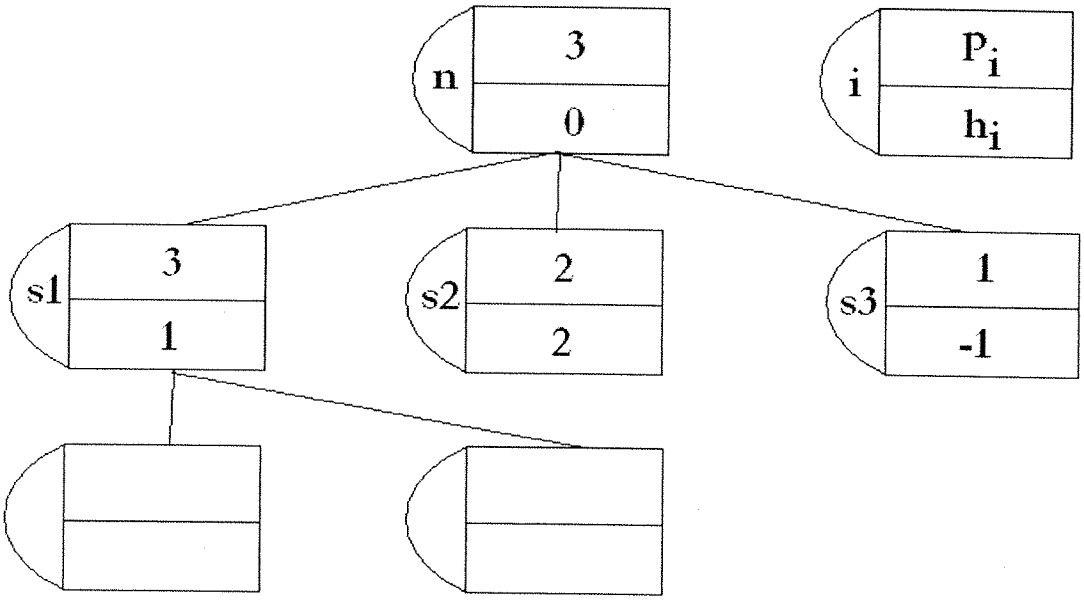



Figura 3 – Representação em grafo de uma abertura

Para se expandir um livro de aberturas partindo da posição inicial ou de um grafo já pré-calculado, é necessário reproduzir os seguintes passos:

(1) Escolher um nó folha e adicionar todos os sucessores ao livro
(2) Calcular os valores heurísticos dos novos sucessores
(3) Propagar os valores por todo o grafo

O princípio de propagação consiste em trocar o valor de $p_i$ do nó folha escolhido, calculado a partir dos valores $h_i$ dos seus sucessores. Encontrar todos os nós que possuem arcos que levam ao nó folha escolhido e recalcular os mesmos e assim por diante até que se encerrem os nós que necessitam que seus valores sejam atualizados com a nova informação.

O segredo de um bom livro de aberturas pode estar na estratégia de expansão, a forma como são escolhidos os próximos nós a serem expandidos. Existem três formas mais usuais de expansão: em profundidade, primeiro o melhor e "drop-out" (esta última proposta por Thomas R. Lincke em 2000).

- Expansão em profundidade

  Corresponde a forma mais ingênua de se garantir que em todo jogo, o programa vai jogar ao menos um determinado número de jogadas com informações do livro de aberturas, antes de ser obrigado a utilizar seus recursos de meio de jogo. O princípio consiste em expandir todos os nós com profundidade 1 a partir da posição inicial, calculando seus valores e armazenando. Após isso, fazer o mesmo para todos os nós com profundidade 2 e assim por diante. Isso, no entanto, é uma perda de tempo e espaço com posições que dificilmente ocorrerão em um jogo de campeonato, porque, para atingí-las, algum dos jogadores teria que ter feito algum erro. Esse tempo e espaço seriam utilizados na expansão de nós mais interessantes melhorando a qualidade da abertura. No entanto, pode ser uma boa estratégia para programas comerciais que enfrentam jogadores de diferentes níveis que podem seguir qualquer caminho a partir do início.

- Expansão primeiro o melhor

  A melhor idéia para expandir um livro de aberturas é escolher nós que possuem uma maior probabilidade de ocorrer em um jogo. Pensando em jogos de campeonato, podemos assumir que bons movimentos são mais prováveis de acontecer do que movimentos ruins. A regra da expansão primeiro o melhor é expandir o nó folha que é atingido pelo caminho de melhores movimentos a partir do nó inicial. Se um nó interior possui mais de um melhor movimento, então um deles é escolhido aleatoriamente.

  Essa estratégia é simples e ignora movimentos ruins, mas ela tem uma grave falha. Suponha que, para o problema de Othello comentado, a função de avaliação retorne 0.1 para a posição C3 e 0.0 para as posições E3, B5 e C4. A expansão irá continuar ao longo do caminho C3 que possivelmente será sempre maior do que 0.0, então todos os outros movimentos serão ignorados para sempre. Isto viola a regra do programa de tentar se manter o máximo de tempo dentro do livro de aberturas. Se o caminho seguido pelo jogo for diferente de C3, o programa terá que abandonar o seu livro de aberturas e partir para seus recursos de meio de jogo muito cedo.

- Expansão "drop-out"

Essa estratégia procura solucionar a falha criada pela estratégia de expansão "o melhor primeiro". Para a escolha do próximo nó, são considerados todos os movimentos e dados para cada um deles uma prioridade que é função da

profundidade das aberturas seguintes ao nó (sub-livro) e da diferença entre o melhor valor e o valor do nó. Um sucessor tem alta prioridade de expansão se ele é um bom movimento e/ou se ele tem um sub-livro raso. Ele tem baixa prioridade de expansão se ele é um movimento ruim e/ou se ele tem um sub-livro profundo.

Para calcular as prioridades de expansão, foram criados 2 novos atributos para cada nó, $epb_i$ e $epo_i$ .

$epb_i$ é a prioridade para quando é o movimento do jogador. É iniciado com zero nos nós folha e depende apenas da prioridade de expansão dos sucessores ótimos. Na fórmula a seguir, o +1 é a penalidade pela profundidade para garantir que nós mais rasos terão maiores prioridades.

$epo_i$ é a prioridade para quando é o movimento do oponente. É iniciado com zero nos nós folha e depende da prioridade de expansão de todos os seus sucessores. Além da penalidade por profundidade (+1), movimentos sub-ótimos têm uma penalidade adicional que depende da diferença de valor para o movimento ótimo.

$$epb_i = \begin{cases} 1+\min_{sj\ \text{ótimo}}(epo_{sj}) & \text{para nós interiores} \\ 0 & \text{para nós folha} \end{cases}$$

$$epo_i = \begin{cases} 1+\min_{sj}(epb_{sj}+\omega(p_i - p_{sj})) & \text{para nós interiores} \\ 0 & \text{para nós folha} \end{cases}$$

$\omega$ é o peso para a diferença $p_i - p_{sj}$, entre o valor ótimo e o valor sub-ótimo do sucessor $s_j$. Ele tem de ser maior que ou igual a zero, mas a escolha certa para $\omega$ depende do jogo e da resolução de valores da heurística. Um valor baixo para $\omega$ significa alta prioridade para movimentos sub-ótimos (se $\omega$=0, todos os sucessores serão expandidos para a mesma profundidade, independente dos seus valores). No entanto, se $\omega \rightarrow \infty$ então a expansão degenera em uma expansão "o melhor primeiro".

As figuras 4 e 5 mostram diagramas de profundidade por valores de um livro de aberturas de Othello para diferentes valores de $\omega$.

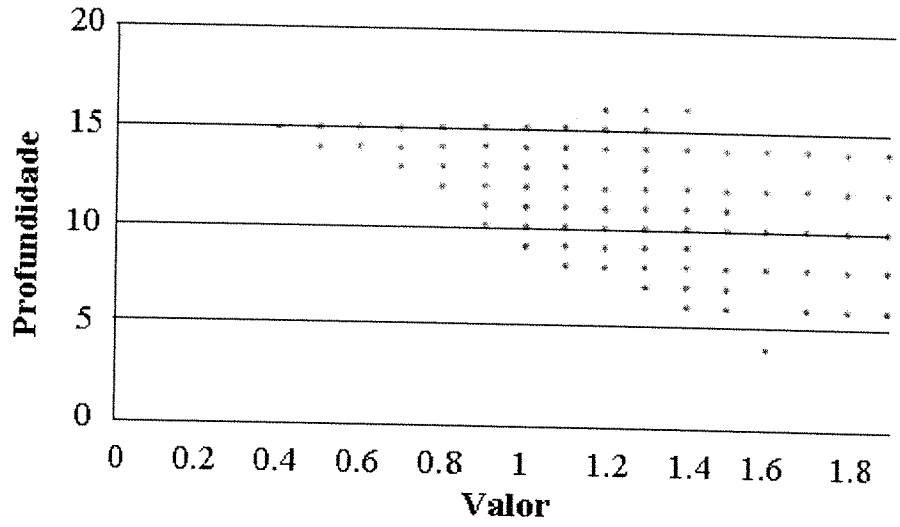



Figura 4 - Diagrama "Drop-out" de um livro de aberturas de Othello (ω=1.0) [51]

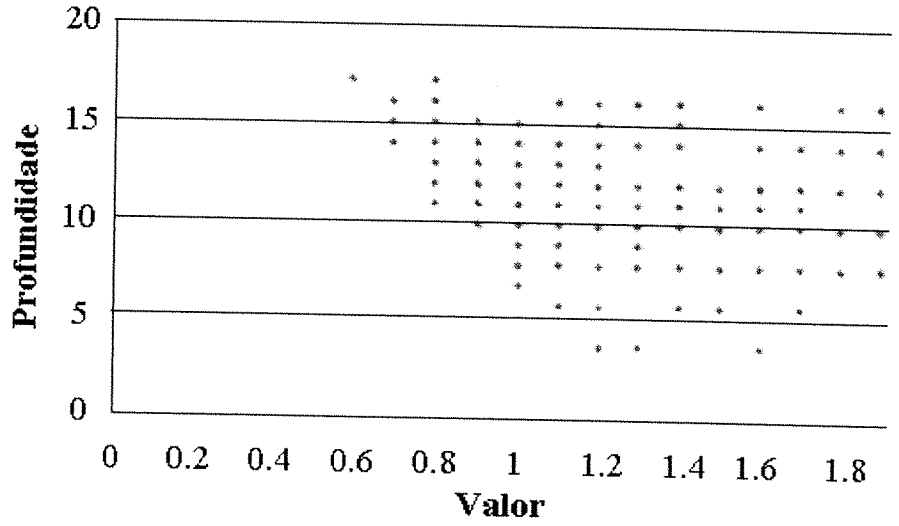



Figura 5 - Diagrama "Drop-out" de um livro de aberturas de Othello (ω=2.0) [51]

A grande vantagem da expansão "drop-out" é que um movimento que é apenas um pouco pior do que o movimento ótimo não será ignorado, pois, com o aumento da profundidade dos melhores movimentos, a prioridade dos movimentos sub-ótimos irá aumentar. A outra vantagem é o parâmetro ω que permite o ajuste mais próximo ou mais distante da expansão "o melhor primeiro".

## 2.3 – Atualização de livro de aberturas

Uma das motivações de atualizar um livro de aberturas é não perder dois jogos da mesma forma, ou seja, uma vez identificado que uma abertura foi parcialmente responsável

por uma derrota, o programa deveria atualizar as suas informações de forma a não utilizar a abertura novamente. Esse também pode ser considerado um tópico de aprendizado.

Em 1999, Buro [19] descreveu uma técnica que era utilizada em muitos programas fortes de Othello. O princípio sugere a construção incremental de um livro de aberturas adicionando toda posição de todo jogo encontrada pelo programa a uma árvore de movimentos. O programa avalia todas as posições à medida que joga e insere não apenas o movimento jogado, mas também o próximo melhor movimento de acordo com a sua avaliação. Então, a cada nó, ao menos duas opções estão disponíveis, o movimento jogado e uma sugestão alternativa. Cada nó folha da árvore do livro é avaliado pelo seu resultado (caso seja uma posição terminal) ou pela heurística. Durante o jogo, o programa encontra a posição atual no livro e escolhe o melhor movimento realizando uma busca na sub-árvore do livro de aberturas. Além de aprender com os próprios erros, o programa pode também aprender com os acertos do adversário. Muitos programas estão conectados em servidores de jogo na internet para, entre outras coisas, ampliar o seu livro de aberturas.

Uma estratégia para não repetir erros pode ser lembrar a posição em que o erro aconteceu. Isso funcionaria como um alerta. Em 2001, Epstein [31] implementou uma abordagem desse tipo em seu sistema HOYLE. Após cada jogo decisivo, o HOYLE olha para a última posição em que o derrotado poderia ter feito um movimento alternativo e tenta determinar o valor dessa posição através de uma busca exaustiva. Se a busca for considerada bem-sucedida, a movimento ótimo é gravado e a posição é marcada como "significante". A informação será utilizada em outras partidas caso a posição ocorra novamente.

## 2.4 – Biblioteca de aberturas

Algumas das estratégias descritas aqui estão implementadas na biblioteca de aberturas OPLIB. OPLIB é uma ferramenta de software independente para jogos de 2 jogadores baseada em uma implementação de grafos cíclicos dirigidos. Existe uma interface bem definida onde a biblioteca acessa funções específicas de um jogo e pode, a partir daí, gerar aberturas para o mesmo.

Desde que foram implementadas as regras de Othello, já existem mais de 500.000 nós construídos no livro de aberturas do mesmo. Para guardar uma posição de abertura para o jogo de Othello são necessários aproximadamente 120 bytes em média (caso não fosse necessária a estrutura de grafo, uma posição de Othello poderia ser armazenada em apenas 16 bytes). De qualquer modo, não existe necessidade de se otimizar o uso de espaço em disco, uma vez que seria possível armazenar 80 milhões de posições em 10Gbytes (dentro dos limites da tecnologia atual de discos rígidos). O grande gargalo é o tempo de processamento. Para se construir uma posição com uma boa qualidade para os valores de nós folha, uma sugestão é utilizar pelo menos o tempo médio de uma jogada de campeonato (~3 min). Nessas condições, para se construir 80 milhões de posições, seriam necessários mais de 400 anos. A OPLIB pode executar em modo distribuído para minimizar esse problema.

Para jogos onde as aberturas têm um importante papel, um programa que não implemente as técnicas aqui comentadas ou outras que porventura venham a surgir, pode ter uma grande desvantagem em relação a jogadores humanos ou mesmo outros programas.

# Capítulo 3

# Meio de jogo

Os jogos de tabuleiro considerados nesta dissertação podem parecer uma área menos interessante de estudo por lidar com o entretenimento ou, por vezes, até ser uma brincadeira de criança. O fato, no entanto, é que esses inocentes jogos são mais difíceis em termos de classes de complexidade que os problemas NP. A maioria deles pertence a classes de complexidade tais como Pspace, Exptime e Expspace. A área de estudo também apresenta aplicações ou conexões com diversas outras áreas, tais como complexidade, lógica, grafos, teoria dos matróides, redes, teoria dos números surreais e biologia.

Problemas de decisão, como o problema do circuito hamiltoniano ou do caixeiro viajante são problemas NP-completos. Aparentemente, a melhor solução parece ser percorrer a maior parte da árvore de decisão (ou toda), cujo tamanho é exponencial ao tamanho de entrada do problema. Já um problema de decisão em um jogo como Xadrez onde é a vez das brancas jogarem e quer se saber se elas podem vencer, tem uma outra forma: Existe um movimento das brancas para o qual qualquer movimento das pretas existe um movimento das brancas para o qual qualquer movimento das pretas existe um movimento das brancas ... de forma que as brancas possam vencer? Neste caso, estamos buscando toda uma sub-árvore e não um único caminho como nos problemas anteriores. É por isso que qualquer jogo de tabuleiro não-polinomial é, no mínimo, Pspace-difícil [34].

Alguns jogos possuem variantes que têm solução matemática simples. É o caso do jogo de Nim, que possui uma variante onde os jogadores tiram quantas peças quiserem de uma única linha na sua vez. O jogador que tirar a última peça vence o jogo. A solução conhecida para este jogo é deixar sempre uma configuração de peças, de tal forma que a soma binária do número de peças de cada coluna é 0 (zero). A única ação que um programa jogador teria que realizar é efetuar essa verificação em sua vez de jogar. Um jogo de tabuleiro que não possua solução matemática simples como essa variação do jogo de Nim, precisa utilizar outros recursos como buscas e heurísticas, por exemplo.

O sucesso das técnicas criadas para jogos é incontestável. Um marco desse sucesso foi o ano de 1997, onde o "Deep Blue", uma máquina que se valia de intenso paralelismo, venceu o campeão mundial de Xadrez Garry Kasparov em uma partida com 6 jogos. O fato é que, naquele mesmo ano, um programa jogador de Xadrez executando em computador pessoal conseguiria derrotar 99.9% dos humanos utilizando apenas um algoritmo de busca em árvore e algumas heurísticas (técnicas usuais de meio de jogo) [15].

Diferente das técnicas de abertura e de fim de jogo, onde é possível, por vezes, utilizar bancos de dados pré-calculados com uma pequena porção do espaço de estados do jogo, a quantidade de possibilidades no meio de jogo é muito grande e essas técnicas se apresentam inviáveis para utilização.

Muitos esforços foram realizados por pesquisadores no que diz respeito às técnicas de meio de jogo. Diversos algoritmos de busca seqüenciais e paralelos foram criados com o passar dos anos. Outros muitos avanços também foram descobertos para os já existentes. No entanto, alguns algoritmos antigos ainda se apresentam como grandes responsáveis pelo

sucesso dos programas jogadores atuais, tais como o Minimax e o Alpha-beta. Eles representam técnicas-base que serão descritas aqui para que os avanços mais recentes possam ser compreendidos.

As técnicas de meio de jogo, criadas com o passar do tempo, são de um número tão grande que restrições de tempo impedem a descrição de todas elas. Dessa forma, este capítulo tenta focar nas técnicas de maior sucesso e nas mais recentes e promissoras. Como em outros capítulos, um jogo foi eleito para ser melhor comentado neste, o jogo de Xadrez. A grande maioria das técnicas pode ser generalizada para outros jogos, porém existe um pequeno risco de perda de generalidade, mas que se faz necessário pelo bem da clareza e da compreensão das técnicas citadas. A primeira seção discute o paralelo busca versus conhecimento, onde são mostradas as implicações de ambos nos programas jogadores. Nas seções seguintes, são descritas as técnicas de meio de jogo propriamente ditas.

## 3.1 – Paralelo: Busca versus Conhecimento

Muitos experimentos foram realizados em programas jogadores para mensurar os benefícios de um conhecimento melhorado e/ou de uma busca mais profunda. Em particular, o Xadrez se tornou um jogo muito popular para esses experimentos. A mensagem implícita ou explícita desses trabalhos é que os resultados para o Xadrez podem ser generalizados para outros jogos. Existem ainda poucos estudos que examinaram o impacto do conhecimento melhorado sobre o desempenho do programa. Em contraste, os benefícios de uma busca adicional foram muito bem documentados: buscas mais profundas produzem ganhos de desempenho imediatos [46].

A figura 6 foi conjeturada para representar o relacionamento entre a qualidade do conhecimento e o esforço de busca. As curvas representam as várias combinações de busca e conhecimento com desempenho equivalente. A figura mostra que aumentando o esforço de busca, menos conhecimento é necessário para se atingir o mesmo desempenho, e vice-versa. A figura não foi ainda comprovada e pode não ser correta. No entanto, experimentos realizados em alguns jogos parecem confirmar essa hipótese.

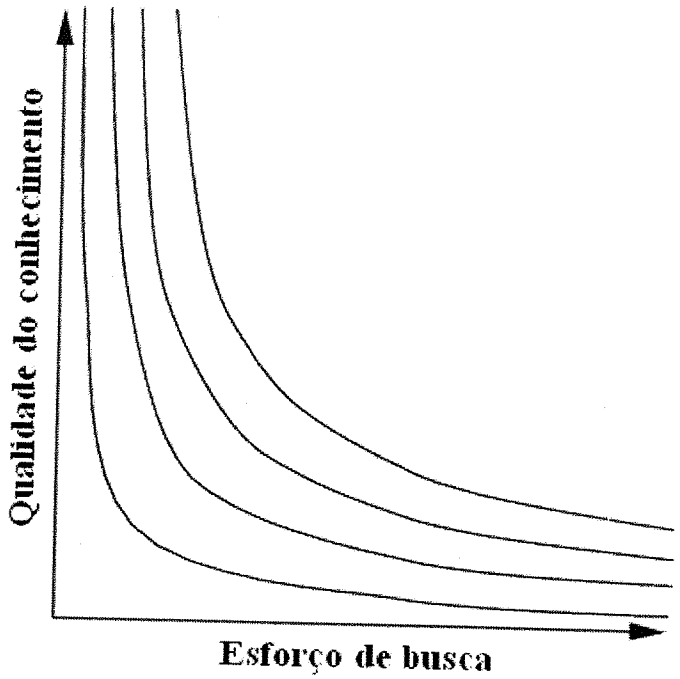


Figura 6 – Proposta de relacionamento da busca e do conhecimento [46]

Das duas dimensões, melhoras na busca são mais fáceis de lidar. Ganhos podem ser alcançados com pouco esforço. Alguém pode simplesmente reprojetar um algoritmo ou reescrevê-lo de forma a executar mais rápido, ou, melhor ainda, pode não fazer nada e aguardar para que computadores mais velozes se tornem disponíveis.

A dimensão do conhecimento já é mais nebulosa. Considerando que já existem métricas bem definidas para mensurar o esforço de busca (tais como profundidade, tempo de execução e nós examinados), não existe nada comparável para o conhecimento.

É interessante observar que as curvas são provenientes de dois pontos conhecidos no gráfico: o ponto em que não há busca e o conhecimento é perfeito e o ponto em que não há conhecimento, mas a busca é exaustiva. Ambos os pontos correspondem a um programa jogador perfeito. É claro que sempre existe algum conhecimento, pois um programa que realize uma busca exaustiva precisa, ao menos, conseguir identificar situações de vitória, empate e derrota para poder decidir entre os movimentos. Um programa sem conhecimento algum executaria o movimento correto com uma chance de 1/n (onde n é o número de possíveis movimentos). Isso independente da profundidade da busca, uma vez que sem conhecimento, ele produz apenas movimentos aleatórios. Ainda assim, ele seria melhor do que um programa com qualidade de conhecimento negativa, que no pior caso deveria escolher sempre o pior movimento.

Considerando um programa que não realiza buscas e decide somente baseado no conhecimento e se considerarmos como medida de desempenho a taxa de movimentos corretos (considerando apenas um por jogada), então temos:

| Qualidade do conhecimento | Desempenho |
|---|---|
| Perfeita | 100% |
| 0 (zero) | 100/n % |
| Antiperfeita (pior caso negativo) | 0% |

Tabela 4 – Tabela da qualidade do conhecimento e o desempenho

Diversos esforços foram realizados para analisar aspectos relativos a busca e ao conhecimento. Por exemplo, Beal [8], percebendo os grandes resultados obtidos pelos programas baseados em busca, criou um modelo matemático para analisar se e porquê os valores retornados por um algoritmo de busca eram mais confiáveis que os valores retornados pela heurística propriamente dita. Para sua surpresa, os valores retornados pela busca eram menos confiáveis que os valores das heurísticas. Ele então escreveu: "Este resultado é desapontador. Era esperado que a análise mostrasse que a probabilidade de erro reduziria com o aumento da busca".

Alguns anos depois, foi mostrado que o modelo matemático utilizado não considerava uma característica principal em jogos de tabuleiro populares: os valores verdadeiros de nós irmãos em uma árvore de jogo não são independentes um do outro [12]. Esse problema ficou conhecido como "patologia de busca". Apesar de concordar que fortes dependências entre nós irmãos eliminavam a patologia, Pearl [59] argumentou que jogos práticos como Xadrez não tinham dependências dessa força entre nós irmãos. A conclusão dele é que "O sucesso dos programas de busca se baseia no fato de que jogos comuns não possuem uma estrutura uniforme, mas são decifrados por posições terminais precoces, coloquialmente chamadas de armadilhas. Ancestrais desses nós terminais carregam informações mais confiáveis do que o resto dos nós, e quanto mais desses ancestrais existem na busca, mais o resultado se torna válido". Ainda sobre o assunto da patologia, Schrüfer [72] e Althöfer [4], observaram que para evitar a patologia, a heurística, entre outras coisas, deveria ter probabilidade desprezível de subestimar posições da perspectiva do jogador.

O porquê do sucesso dos programas baseados em busca parece ainda um assunto distante de uma conclusão. Em 2003, Sadikov [66] realizou experiências em um final de jogo de Xadrez RTR (KRK) onde as brancas possuem um Rei e uma Torre e as pretas possuem apenas um Rei. A abordagem utilizada foi experimental e não matemática como nos outros experimentos citados. Apesar de ser uma situação muito simples e só ter 2 resultados possíveis: vitória para as brancas ou empate, ela ainda possui todos as características importantes para o experimento: posições de diferentes dificuldades (considerando como dificuldade, a quantidade mínima de movimentos necessários até o xeque-mate), dependências entre valores de nós irmãos na árvore de jogo e a existência da possibilidade de um fim precoce. A vantagem desse experimento era poder contar com o valor correto da posição sem necessitar de heurísticas (um banco de dados de 28.056 posições foi utilizado). Para o experimento, Sadikov corrompeu o valor correto da posição, de uma maneira controlada, utilizando um ruído gaussiano sobre os valores. Os valores foram corrompidos em diversos níveis para comparação. Para os valores resultantes da busca, era calculada uma tendência que mostrava se os resultados eram otimistas/positivos (acima dos valores corretos) ou pessimistas/negativos (abaixo dos valores corretos). Como os valores foram corrompidos simetricamente, Sadikov esperava que o valor da tendência fosse 0 (zero). No entanto, o experimento mostrou que, quanto maior era o nível de

corrupção dos valores, maior era a tendência. O fato de todos os valores serem positivos se deve ao último nível de profundidade avaliado, porém o fato de eles serem diferentes de zero se devem provavelmente ao nível de corrupção (eles apresentam forte correlação). Como conclusão, ele escreveu: "Nossos resultados mostram que dependências entre avaliações de nós irmãos em árvore de jogo e a abundância de possibilidades de armadilhas presentes no final de jogo RTR não são suficientes para explicar o sucesso das buscas em jogos práticos como se acreditava previamente. O artigo mostra que a busca em combinação com a avaliação com ruídos introduz uma tendência nos valores propagados e sugere que essa tendência foi quem mascarou a eficiência das buscas em estudos anteriores."

O paralelo da busca versus conhecimento, aponta para o fato que, neste capítulo de meio de jogo, deveriam ser discutidas técnicas relativas ao conhecimento e às buscas. No entanto, o conhecimento utilizado nos programas jogadores ou é proveniente do aprendizado humano do jogo em questão (literatura, conselhos de especialistas, partidas realizadas, etc) e, portanto não é computacionalmente relevante para ser discutido aqui; ou é proveniente de aprendizado de máquina, que será discutido no capítulo 5. Dessa forma, este capítulo versa apenas sobre as técnicas de busca em jogos de tabuleiro.

## 3.2 – Técnicas de busca

A presença de um oponente torna o problema de decisão mais complicado do que problemas de busca tradicionais. O oponente introduz a incerteza, porque a máquina nunca sabe o que o jogador irá fazer. Todos os programas de jogos precisam lidar com esse "problema de contingência".

O que torna realmente os jogos difíceis de resolver é o fator de ramificação (a quantidade máxima de opções que um jogador possui na sua vez de jogar). Isso porque o programa de busca deve construir uma árvore de possibilidades de jogo a partir daquela situação e, quanto maior o fator de ramificação, maior a quantidade de nós a cada nível da árvore. O jogo de xadrez possui, na média, 35 filhos para cada nó e os jogos têm cerca de 50 jogadas por jogador. Então a árvore de busca tem cerca de $35^{100}$ nós.

Um dos jogos que tem se mostrado bastante complexo é o jogo de Go, jogo de tabuleiro mais popular do Japão. Por possuir um fator de ramificação de aproximadamente 360, os métodos regulares de busca não têm a menor chance. Sistemas baseados em grandes bases de conhecimento de regras parecem ter esperança de se tornarem desafiadores, mas ainda jogam muito mal.

Para se entender melhor os algoritmos de busca, é preciso entender a porção do conhecimento que eles utilizam. Dessa forma, a primeira subseção desta seção discute a "função de avaliação", que corresponde à porção necessária do conhecimento que as buscas utilizam. Nas subseções seguintes, são discutidos os algoritmos de busca seqüenciais e os algoritmos de busca paralelos em árvore de jogo.

### 3.2.1 - Função de avaliação

Se não pensarmos em termos de busca, teremos apenas a atual situação do tabuleiro para realizar a tomada de decisão. Uma função que avalia a situação do tabuleiro e retorna um valor de $-\infty$ a $+\infty$ (ou uma outra faixa escolhida) indicando vantagem para um dos lados

ou 0 para uma situação de empate, é usualmente conhecida na literatura por "função de avaliação". Quando um programa jogador possuir $x$ movimentos possíveis em uma determinada situação de tabuleiro, ele pode gerar $x$ situações de tabuleiro (uma para cada movimento), aplicar a função de avaliação em cada uma delas e escolher o movimento cujo valor é mais vantajoso para o programa. Dessa mesma forma, a função de avaliação pode ser generalizada para utilização nas buscas.

O Algoritmo de busca usual assume que o programa tem tempo de percorrer toda a árvore até os estados terminais, o que, no caso de jogos de tabuleiro, na maioria das vezes, não é verdade (isso pode funcionar para um jogo da velha, mas não para a árvore do jogo de Xadrez, por exemplo). Se fosse dessa forma, o algoritmo precisaria apenas de uma função que reconhecesse um estado terminal e retornasse um valor baseado no resultado do estado terminal (função utilidade). Propagando os valores para a situação atual, ele poderia fazer a escolha perfeita.

A única opção para jogos complexos é parar a busca no meio da árvore (possivelmente fixando um limite de profundidade). É claro que, para folhas não terminais, não é possível aplicar a função utilidade. Nesse caso, deve-se utilizar uma função de avaliação para cada folha.

Um programa jogará tanto melhor quanto melhor for sua função de avaliação. Uma possível heurística para o jogo de Xadrez seria atribuir valor às peças do jogo (torre – 5, rainha – 9, etc) e ainda considerar fatores como segurança do rei, por exemplo.

Dois requisitos para a função de avaliação são: concordar com a função utilidade nos estados terminais e não demorar muito (alguns programas precisam jogar campeonatos que possuem limite de tempo, outros enfrentam jogadores humanos que não gostariam de ficar esperando muito tempo por uma resposta da máquina). Isso implica em balancear o desempenho com a precisão para se obter um bom resultado.

Dependendo da função avaliação, ela pode ter resultados desastrosos para certos casos. Considerando uma função de avaliação de um jogo de Damas que conta o número de peças de cada lado. O grande problema é que a busca pode ter parado em uma posição em que o jogador do outro lado irá ganhar várias peças, então o que parecia ser uma boa posição, torna-se uma derrota iminente (conhecido como "efeito horizonte"). Essa posição é chamada de não-quiescente. Para esses casos, é preciso fazer uma busca extra (busca quiescente) até encontrar uma posição quiescente e só então utilizar a função de avaliação.

### 3.2.2 – Algoritmos de busca seqüencial

Por mais de 40 anos, o algoritmo alpha-beta tem sido o escolhido para busca em árvores de jogo. Utilizando-se uma simples esquerdo-direita busca em profundidade ele é capaz de fazer uma busca eficiente em árvores [47]. Muitos avanços foram realizados sobre o algoritmo. Alguns estudos mostraram que a eficiência do algoritmo está próxima da ótima e existe pouco espaço para novos avanços [68]. Para se entender o algoritmo alpha-beta, antes é preciso entender a idéia do Minimax.

Em 1979, Stockman [75] introduziu o algoritmo SSS* que era capaz de expandir menos nós que o Alpha-beta adotando uma estratégia de busca primeiro o melhor. Mesmo sendo uma idéia interessante para teóricos e provando que árvores de busca menores podiam ser construídas, o algoritmo não se tornou muito utilizado pelos práticos. Isso porque o algoritmo tinha alguns grandes problemas de desempenho.

Esta subseção se limita a descrever apenas alguns algoritmos mais bem-sucedidos no sentido teórico e prático, que por sua vez se tornaram mais conhecidos ou mais utilizados. Entretanto, diversos outros algoritmos de busca seqüencial existem e possuem os seus méritos. A título de exemplo, em 1980, Pearl [58] introduziu o algoritmo SCOUT. A idéia baseia-se nos denominados "cortes no nível" (é possível se efetuar cortes sem a informação de valores dos irmãos, avôs, tios, etc) [60]. O algoritmo calcula o nó mais à esquerda e, para cada nó direito, primeiro testa, e somente se necessário calcula o nó. Mesmo esse algoritmo necessitando de uma quantidade de memória que independe da árvore de jogo (enquanto o Minimax e o Alpha-beta precisam de memória proporcional à profundidade), ele acaba por avaliar mais nós que os avaliados pelo alpha-beta.

Uma questão central em qualquer análise de algoritmos é o método pelo qual se compara o desempenho dos algoritmos. Nos casos de busca em árvores de jogo, quatro métodos são normalmente utilizados [58] :

- *Número de nós terminais (NBP)*: Consiste em se contar o número de nós terminais avaliados pelo algoritmo. O fundamento dessa comparação é a consideração de que a maior parte do tempo de processamento, no caso de algoritmo de busca em jogos reais, é gasto no cálculo da função de avaliação. Neste sentido, o tempo gasto com os nós terminais, onde as funções de avaliação são aplicadas, deveria responder pela maior parte do tempo gasto pelo algoritmo (alguns novos avanços consideram a função de avaliação também em nós não-terminais). É um dos métodos mais utilizados.
- *Fator de ramificação*: Consiste na versão assintótica do NBP, definida da seguinte maneira: Seja um algoritmo A e uma classe $\phi$ de árvores de jogo; seja ainda $T_{A,d,h}$ o número médio de nós terminais examinados pelo algoritmo A em árvores uniformes de grau d e profundidade $p$, pertencentes à $\phi$; então o fator de ramificação do algoritmo A sobre a classe $\phi$, $\tau_A(d) = \lim_{p\to\infty} (T_{A,d,h})^{1/h}$. Intuitivamente, corresponde ao poder de redução do algoritmo. Embora seja um dos métodos favoritos da análise teórica, possui reduzida aplicação na prática na medida em que a profundidade $p$ necessária para que $\tau_A$ comece a convergir para o seu valor teórico pode, facilmente ser impraticável computacionalmente.
- *Números de nós visitados*: é uma variação de NBP na qual são considerados todos os nós pelos quais o algoritmo passa. Era pouco utilizado devido à semelhança com o NBP, e também por misturar nós com tempos de processamento drasticamente diferentes (isso já não é uma verdade uma vez que técnicas modernas sugerem a aplicação da função de avaliação em todos os nós).
- *Tempo de CPU*: Método que compara algoritmos diretamente pelo tempo gasto no processamento. Assim, os resultados são dependentes de diversos fatores indesejáveis, como o tempo de processamento de um nó terminal, a codificação do algoritmo, as características da máquina, entre outros. Deste modo, este método é normalmente empregado somente para teste de um mesmo algoritmo sobre árvores de diferentes modelos e tamanhos.

A tabela 5 baseada em [21], compara os algoritmos comentados (assintoticamente ótimos) sobre árvores aleatórias com o algoritmo Minimax (que avalia todos os nós) pelo método NBP.

| Algoritmo | Profundidade $p$=2 | Profundidade $p$=4 |
|---|---|---|
| Minimax | 567 | 331.776 |
| Scout | 191 | 11.313 |
| Alpha-beta | 161 | 10.857 |
| SSS* | 100 | 3.252 |

Tabela 5 - Simulação dos algoritmos ótimos sobre árvores aleatórias de grau 24 [21]

É importante observar que apesar de avaliar uma quantidade de nós superior ao SSS*, o algoritmo alpha-beta possui um desempenho melhor na prática e, por isso, é mais largamente utilizado.

O número de posições avaliadas pelo algoritmo minmax considerando um fator de ramificação *fr* e uma profundidade *p* é "$fr^p$". Existe, no entanto, um valor teórico mínimo de nós que um algoritmo deveria percorrer para determinar o valor minimax, $NBP_{min}$ posições terminais.

$$NBP_{min} = fr^{\lceil p/2 \rceil} + fr^{\lfloor p/2 \rfloor} - 1$$

Alguns algoritmos chegam a esse valor no melhor caso para árvores de jogos uniformes, entre eles o alpha-beta. O melhor caso do alpha-beta ocorre quando, a cada nó, o movimento expandido primeiro é o que possui o maior valor minimax. Uma árvore desta forma é chamada de "árvore de jogo perfeitamente ordenada" ou "árvore crítica". A árvore crítica, no entanto, não é necessariamente a mínima. Outros métodos podem gerar o valor correto percorrendo uma árvore ainda menor.

O grafo "mínimo primeiro-esquerda" tem sido utilizado por muitos autores para ilustrar o tamanho mínimo da menor árvore que pode ser procurada. Contudo, este grafo não tenta minimizar o número de nós duplicados dentro da árvore, nem tenta sempre pegar o corte mais barato para determinar o grafo mínimo. Pesquisas recentes mostraram que o "verdadeiro grafo mínimo", a menor árvore possível que qualquer algoritmo pode buscar para determinar o valor minimax, é significantemente menor que o "grafo mínimo primeiro-esquerda" [15]. Contudo, encontrar o "verdadeiro grafo mínimo" é um problema computacionalmente intratável [61].

As subseções, a seguir, descrevem os algoritmos de busca seqüencial.

### 3.2.2.1 – MiniMax

Um jogo pode ser formalmente definido como um tipo de problema de busca com os seguintes componentes:

- Estado inicial: inclui a posição do tabuleiro e a indicação de quem é a vez.
- Conjunto de operadores: definem os movimentos legais do jogador.
- Teste terminal: determina quando o jogo terminou.
- Função utilidade: retorna valor numérico como resultado do jogo.

No caso do Xadrez, os resultados podem ser vitória, derrota ou empate, o que podemos representar pelos valores +1,-1 ou 0. Alguns jogos podem ter uma variedade de resultados como é o caso do Gamão que pode variar de +192 a –192.

Considerando uma partida entre 2 jogadores, J (jogador) e C (computador), C precisa encontrar uma estratégia de jogo que o leve a um estado terminal de vitória (+1). Esse é o objetivo do algoritmo Minimax, determinar uma estratégia ótima e decidir qual o próximo movimento de C. O algoritmo parte do pressuposto que a cada jogada, J fará sempre a melhor escolha, portanto a idéia é minimizar as chances de vitória de J e maximizar as chances de vitória de C.

O algoritmo consiste em 5 etapas:

- Gerar toda a árvore do jogo até todos os estados terminais.
- Aplicar a função utilidade a cada estado terminal para pegar seu valor.
- Utilizar esses resultados para gerar o valor numérico dos nós acima considerando que na vez de J, ele sempre escolherá o nó de menor valor (MIN) e na vez de C, ele sempre escolherá o de maior valor (MAX).
- Continuar esse processo das folhas até a raiz, um nível a cada vez.
- Ao chegar à raiz, C escolhe o nó filho que tiver o maior valor numérico como sendo a melhor jogada.

A figura 1 exibe os primeiros níveis de uma árvore de um jogo qualquer. É a vez de C jogar e é preciso escolher um dos filhos da raiz. A jogada de C é do nível 1 para o 2 e a de J é do nível 2 para o 3. O nível 3 está inicialmente preenchido com seus valores numéricos. O nível 2 é preenchido com o menor valor dentre os filhos pois é a vez de J jogar e a raiz é preenchida com o maior valor dentre os filhos pois é a vez de C jogar. Finalizando, a melhor escolha para C é fazer a jogada correspondente ao nó do meio do nível 2.

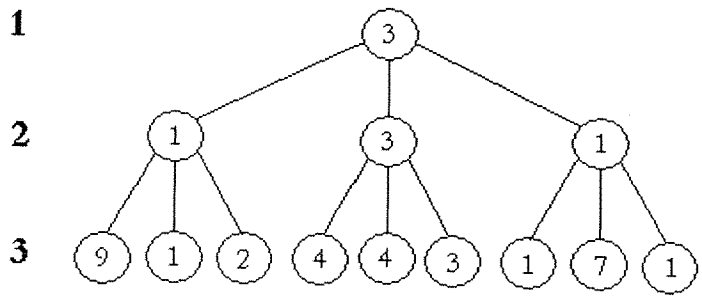



Figura 7 – Primeiros níveis de uma árvore de um jogo qualquer

Se a profundidade máxima da árvore é p e existem m movimentos legais, então a complexidade do algoritmo minimax é $O(m^p)$. Para jogos reais, o custo de tempo é impraticável, mas este algoritmo serve de base para técnicas mais realistas e para análise matemática de jogos.

A figura 8 mostra uma situação exemplo no jogo de Xadrez da árvore gerada pelo Minimax (considerando que a função de avaliação retorna valores de –1 a 1). Alguns ramos foram omitidos por limitação de espaço. É importante observar que nenhum número existe inicialmente atribuído a cada tabuleiro e os primeiros números são gerados nas folhas e

depois propagados até a raiz. Os movimentos do computador são calculados e os do oponente são estimados com a mesma função de avaliação. O algoritmo considera que o movimento do oponente será sempre o melhor que o computador escolheria. Isso usualmente não é verdade e impede que o programa realize buscas mais profundas cortando sub-árvores desnecessárias, mas, ao menos, garante que o jogador não será subestimado. Observando na última linha onde a árvore foi cortada que a decisão do oponente é muito melhor do que as outras opções que ele tem (-0.5). No entanto, na decisão do computador na linha acima, essa decisão é anulada, pois é escolhida uma outra posição que gera maior vantagem para o computador (0.5). Na terceira linha, no entanto, a decisão do oponente influencia diretamente a decisão final. O resultado é que o programa acredita que ele está com uma leve vantagem (0.1) e decide por um desses caminhos evitando o último movimento que leva a uma situação de empate.

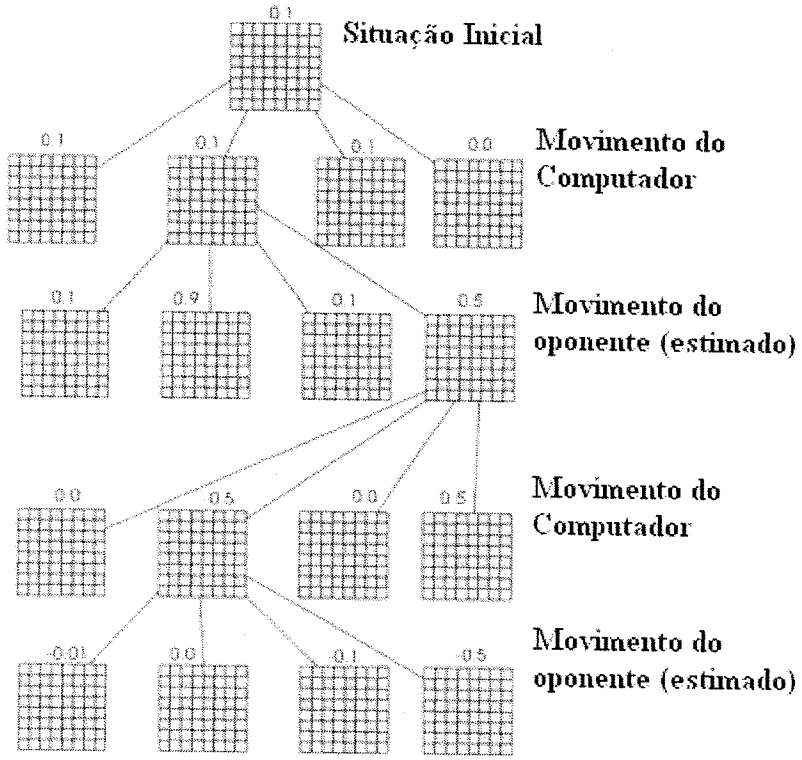



Figura 8 – Árvore de um jogo de Xadrez

No caso de jogos que contém o fator sorte, é preciso tomar muito cuidado com os valores retornados, pois não estará mais sendo escolhido o mínimo e o máximo e sim se fazendo uma média ponderada do valor numérico com a probabilidade de cada folha.

### 3.2.2.2 - Alpha-Beta

Quanto maior o fator de ramificação, menos níveis à frente o computador consegue ver a cada jogada. Para o caso de um programa bem escrito de Xadrez rodando num PC padrão, seria possível olhar 3 ou 4 jogadas à frente com o Minimax. Mesmo um jogador

médio de Xadrez pode olhar 6 a 8 jogadas à frente, isso significa que o programa seria facilmente derrotado.

É possível, no entanto, utilizar o Minimax sem necessariamente percorrer todos os nós de uma árvore de busca. O processo de eliminação de um galho de uma árvore de busca é chamado "Corte" (pruning).

O corte alpha-beta consiste basicamente em se avaliar se a um dado momento já não existe informação suficiente para se escrever os valores dos nós de cima. A primeira versão publicada do algoritmo alpha-beta data de 1963 e pode ser encontrada em [16]. Contudo, existem numerosas reivindicações por pesquisadores sobre quem desenvolveu o método primeiro.

O algoritmo alpha-beta ($\alpha\beta$) é uma modificação do algoritmo minimax. Dois limites são utilizados em cada nó da árvore, $\alpha$ e $\beta$, e estes limites são passados à medida que a árvore é atravessada em profundidade. Em qualquer nó, $\alpha$ representa o menor valor do nó que pode afetar o valor minimax acima daquele ponto da árvore, enquanto $\beta$ representa o maior valor do nó que pode afetar o valor minimax. Por isso, $\alpha$ e $\beta$ são usualmente referidos como janela de busca, que é normalmente escrita como $(\alpha,\beta)$.

O parâmetro $\alpha$ representa o maior valor minimax dos nós MAX das ramificações ao longo do caminho até o nó, incluindo o próprio nó. A medida que são explorados novos sucessores de um nó, $\alpha$ vai crescendo monotonicamente, pois se o valor de $\alpha$ é menor que o valor do nó, então $\alpha$ passar a ser igual a esse valor. De forma similar, $\beta$ representa o menor valor minimax de nós MIN avaliados ao longo do caminho até o nó. Por isso, $\beta$ decresce monotonicamente à medida que mais nós são explorados.

Quando a busca atinge um ponto em que $\beta \leq \alpha$, sabe-se que existe um caminho melhor para um dos jogadores mais próximo da raiz da árvore. Portanto, não existe necessidade de continuar a busca e ela deve voltar ao nó pai imediatamente. Em termos de efeito, isso corta a parte da árvore que não pode contribuir para o valor minimax. Foi mostrado que o algoritmo alpha-beta retornará o valor minimax correto se para a raiz for utilizada uma janela de busca $(-\infty,+\infty)$ [47].

Na figura 9, no nível 3 da árvore, o algoritmo está buscando o menor valor, é possível assumir que no nó do meio, o resultado deverá ser menor ou igual a 3 uma vez que o resultado da primeira folha é 3.

Nesse caso, o algoritmo já calculou o filho esquerdo e o direito do nó raiz (4 e 1 respectivamente). Já é possível então afirmar com certeza que a função no nível 2 da árvore (procura pelo maior valor) retornará o valor 4 para o nó raiz. Nesse caso, a busca não precisa ser continuada (os galhos podem ser cortados).

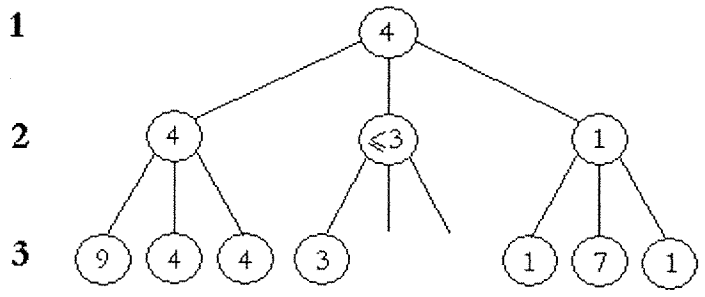


Figura 9 – Galhos desconsiderados pelo corte alpha-beta

A técnica pode ser generalizada para níveis mais distantes um do outro, pois se um dado nó de nível ímpar (por exemplo) possui um valor maior do que um outro nó de nível superior também ímpar, o algoritmo não precisa continuar percorrendo o galho irmão do nó de nível inferior, pois o mesmo nunca jogará aquela opção (pois o valor sempre será menor).

Com o tempo, foram surgindo numerosas melhoras no algoritmo. Alguns desses avanços são descritos a seguir:

**- Busca seletiva**

Nem todos os movimentos são iguais. Alguns movimentos são fracos e deveriam ter menos recursos alocados para eles. Outros movimentos têm potencial e devem ser mais explorados. Em vez da busca com profundidade fixa, muitos programas jogadores reduzem a busca para movimentos mais fracos e aumentam a busca para movimentos mais fortes. Alguns processos de seleção removem apenas movimentos implausíveis e significam apenas um pequeno ganho de desempenho. O sucesso da busca seletiva depende principalmente da qualidade do processo de seleção.

**- Busca por aspiração (com sistema de falha)**

Corresponde a uma tentativa de adivinhar o valor minimax da árvore de jogo antes de fazer a busca. Nessa busca, o algoritmo alpha-beta é iniciado com uma janela menor (a,b) que $(-\infty,+\infty)$ através de uma estimativa feita pelo programa jogador. Se o valor minimax está dentro da janela escolhida, isso causará cortes na árvore em nós anteriores aos nós caso a busca fosse feita com a outra janela. Se o valor minimax $f$ não estiver na janela de busca estimada, o programa deverá reiniciar a busca com a janela $(-\infty,f)$ para $f<a$ e $(f,+\infty)$ para $f>b$.

**- Aprofundamento iterativo**

A idéia é que o algoritmo alpha-beta se limite a explorar uma pequena busca com profundidade $k$ forçando a avaliação dos nós uma vez que essa profundidade for atingida. Quando essa profundidade for atingida, a busca pode continuar por uma distância $s$ e a busca pode ser repetida até $k+s$, e assim por diante. Tipicamente, para programas jogadores de Xadrez, $k$ e $s$ são iguais a 1. Este método é utilizado uma vez que não é possível saber quanto tempo o algoritmo vai demorar, mas sabendo-se quanto tempo demorou para a profundidade $k$, é possível estimar valores para a profundidade $k+1$. Se o tempo estourar no meio da execução do algoritmo, sem esse recurso, o programa poderia realizar um movimento catastrófico, mas com o recurso, basta utilizar o melhor movimento da busca anterior. Esse método se mostrou muito valioso para programas jogadores de campeonatos, por estes serem baseados em tempo.

**- Tabela de transposição**

Informação específica sobre uma busca pode ser salva em uma "tabela de transposição"[39] que pode incluir, com relação a uma posição do tabuleiro (nó), o melhor

valor, o melhor movimento, a profundidade considerada, entre outros. Quando uma posição se repete na busca, todas as informações já estão armazenadas e disponíveis para uso.

**- Ordenação de movimentos**

Visando reduzir a quantidade de nós avaliados pelo algoritmo alpha-beta, a primeira idéia de ordenação que surgiu foi ordenar os nós segundo os resultados da função de avaliação. Apesar de funcionar bem assim para o jogo de Othello, não funcionava bem para o jogo de Xadrez, por exemplo. Um outro modo de ordenar movimentos é utilizando a "tabela de transposição". Nos jogos aqui estudados, o melhor movimento de uma busca anterior tem boas chances de ser o melhor movimento da busca atual. Dessa forma, utilizando valores da busca anterior, é possível ordenar os movimentos dessa busca. Da mesma forma, pode-se utilizar a "tabela assassina" onde são apenas guardados movimentos que causaram um corte na árvore. Esses movimentos devem ser considerados primeiros uma vez que é maior a chance de eles resultarem em novos cortes. Como as tabelas citadas, só oferecem ordenação para alguns movimentos, a "heurística histórica" pode completar a ordenação. Para o jogo de Xadrez, uma matriz 64 por 64 é utilizada. Cada vez que um movimento de uma posição $a$ para uma posição $b$ é escolhido como melhor movimento é armazenado um bônus na posição $[a,b]$. O tamanho desse bônus depende da profundidade em que o movimento foi bem-sucedido. Foi observado na prática que um bônus exponencial com a profundidade funciona bem. Movimentos com valores históricos maiores são mais provavelmente movimentos melhores.

**- Janelas Nulas**

Foi observado que utilizar uma janela de busca reduzida, reduzia o tamanho da árvore percorrida pelo algoritmo alpha-beta. Como conseqüência da ordenação de movimentos, é provável que o primeiro movimento seja o melhor. Supondo que o valor desse movimento seja $\gamma$, então, em vez de buscar os outros movimentos com a janela $(\gamma,\beta)$, poderia ser utilizada a janela $(\gamma, \gamma+1)$. Se os valores dos outros nós não forem inferiores como imaginado, então a busca deve ser refeita com uma janela maior. As janelas nulas só devem ser utilizadas em ambientes com boa ordenação de movimentos.

Alguns experimentos realizados por Schaeffer em [68] tentaram determinar quais desses avanços utilizados de forma combinada, mais contribuíam para o desempenho dos programas jogadores. Seus resultados indicaram que a "heurística histórica" em conjunto com a "tabela de transposição" resultavam em desempenhos bem superiores que outros avanços. Para árvores de jogo até a profundidade 8, quando essas técnicas foram consideradas juntas, elas respondiam por 99% das reduções de tamanho árvore, com os outros avanços representando reduções desprezíveis. De uma forma ou de outra, os limites que podem ser alcançados com uma busca seqüencial através do alpha-beta estão muito próximos do estágio atual de desenvolvimento.

### 3.2.2.3 – SSS* e Pn-search

O Algoritmo "State Space Search" ou SSS* é uma variante do alpha-beta que tenta realizar buscas em regiões mais promissoras da árvore, em vez do tradicional esquerda para

direita do alpha-beta. Este algoritmo possui grandes similaridades com o algoritmo pn-search ("proof number search") que, além de ser um algoritmo primeiro o melhor, ele também não usa avaliações heurísticas para nós internos.

A principal diferença entre esses dois algoritmos está na diferença do critério de qual o melhor nó a ser expandido. O SSS* escolhe o nó puramente baseado no limite superior ainda atingível. Em qualquer momento, durante a busca, o nó que tem a maior possibilidade de ter um limite superior é selecionado. Pn-search não utiliza uma faixa de valores de nós terminais. Em vez disso, o conjunto de valores possíveis de nós terminais é dividido em duas partes. Resolver a árvore significa determinar em qual das duas partes o valor verdadeiro se encontra. Se o valor exato de uma grande faixa precisa ser determinado, pn-search deve ser chamado repetidamente. Como esse algoritmo baseia sua seleção nos números provados ou não, isso significa que um nó é experimentado se pode ser parte de uma solução com esforço mínimo [3].

Inicialmente, acreditava-se que o algoritmo SSS* dominava o alpha-beta no sentido em que o SSS* não procuraria um nó que o alpha-beta não tivesse procurado. O algoritmo original, no entanto, não possui essa propriedade. Uma mudança realizada posteriormente tornou correta a prova de dominância [20].

Um problema percebido com o algoritmo é que a estrutura de lista (a lista aberta) precisava ser mantida. Essa lista poderia crescer para $fr^{p/2}$ elementos, onde $fr$ é o fator de ramificação e $p$ é a profundidade da árvore. Naquele momento, este requisito de memória foi considerado muito grande para programas jogadores de Xadrez. Mesmo hoje ainda existe um problema, o aumento da lista aberta diminui a velocidade do algoritmo na prática. Com os avanços do alpha-beta como a ordenação de movimentos, o SSS* perdeu a sua vantagem e não é mais necessariamente melhor.

Com relação ao pn-search, alguns experimentos foram realizados com intuito de provar valores teóricos de jogos como Lig-4, Mancala e Go-moku. O algoritmo também foi comparado a algumas variantes do alpha-beta. Segundo Allis [3], o pn-search superou em desempenho o alpha-beta. Sua conclusão foi que o sucesso do algoritmo no jogo de Mancala se deveu principalmente à não-uniformidade da árvore. Ele percebeu que existe um grau mínimo necessário de não-uniformidade da árvore para que esse algoritmo possa superar outros.

#### 3.2.2.4 – Números conspiratórios (cn-search)

É de alguma forma um ancestral direto do pn-search. McAllester introduziu este algoritmo na década de 80 como uma alternativa para se encontrar um valor minimax. A idéia básica é determinar quantas folhas dentro da árvore de jogo devem trocar de valor para que o valor minimax de uma posição mude para aquele valor. É de se supor que para alterar o valor da raiz seja necessária uma grande conspiração de nós. A árvore pode crescer um nó por vez de uma maneira a tentar maximizar os números conspiratórios da raiz da árvore de jogo. A idéia parece razoável na teoria e funciona bem na prática para algumas posições táticas do jogo de Xadrez. Contudo, freqüentemente é difícil provar que a raiz da árvore se apóia em 2 ou mais folhas para posições não-táticas.

A grande diferença desse algoritmo para o pn-search é que, enquanto o pn-search foca no menor número de nós que precisam conspirar para provar o valor da posição, o cn-search determina o menor número de nós necessários para mudar o valor da posição. Dessa forma, o pn-search não utiliza funções de avaliação heurística para avaliar nós não-

terminais enquanto o cn-search usa. Mesmo sendo um descendente, o pn-search não tem os mesmos objetivos do cn-search, uma vez que o pn-search é focado em provar valores e, portanto provavelmente não seria mais bem-sucedido que o cn-search aplicado a um programa jogador.

Na década de 90, Lorenz and Rottman [52] desenvolveram a busca por números conspiratórios de forma controlada, cuja tentativa era repartir o trabalho provando um número conspiratório a partir de um determinado nó sobre todos os sucessores desse nó. Esta abordagem permite que a busca seja subdividida e examinada em profundidade (alpha-beta). Além do mais, informações sobre o que o nó está tentando provar permitem ao programa utilizar janelas alpha-beta de buscas mais justas. O programa jogador de Xadrez, Paderborn, utilizou essa busca controlada em um campeonato de Xadrez e obteve um resultado de 3.5 em 7 jogos, o que é um resultado admirável para um algoritmo relativamente novo.

### 3.2.2.5 – B* e BPIP

O Algoritmo B* foi introduzido por Berliner [9]. O objetivo do algoritmo era tentar provar que um determinado movimento é melhor que todos os outros na raiz da árvore. Isto pode ser feito através de limites otimistas e pessimistas definidos heuristicamente em um valor minimax de um nó. O limite pessimista de melhor movimento na raiz da árvore deveria ser melhor do que os valores otimistas de todos os outros movimentos da raiz. O B* tenta de uma maneira primeiro o melhor focar na busca por nós que possam trazer o máximo benefício para completar a prova. Uma vez que a prova seja completada, a busca pode ser terminada. Isto faz com que o algoritmo faça movimentos óbvios rapidamente, uma característica que o alpha-beta não tem.

Vamos assumir, por exemplo, que o nó raiz é um nó MAX e que a raiz tem dois filhos A e B, com valores respectivamente nos intervalos [0,2] e [1,3]. O objetivo do B* é determinar qual o melhor movimento sem necessariamente saber o seu valor exato. Considerando que o melhor movimento seja o B, antes de terminar a busca é preciso provar que o limite superior do intervalo de A (2) pode ser reduzido para um valor abaixo do limite inferior do intervalo de B (atualmente igual a 1) ou provar que o limite inferior do intervalo B (1) pode ser aumentado até, pelo menos, qualquer valor acima do limite superior de A (atualmente igual a 2).

Um importante pré-requisito do algoritmo B* é a confiável função de avaliação que determina o limite superior e inferior para cada nó. Essa avaliação depende fortemente do conhecimento específico de domínio o que pode ser um grande obstáculo em alguns casos. Uma maneira alternativa de obter esses limites utilizando uma pequena busca foi introduzida por Palay [57]. Palay também introduziu regras para propagar a distribuição de probabilidade determinada a cada nó. Outros avanços nas estimativas que guiam a busca foram introduzidos por Berliner e McConnel em [10].

O algoritmo BPIP (Best Play for Imperfect Players) foi introduzido por Baum e Smith em 1995 [6]. O trabalho realizado é muito similar ao B* baseado em probabilidade. A idéia básica do BPIP é retornar a distribuição de probabilidade em vez de um simples valor minimax, e escolher o movimento da raiz que tiver a distribuição de probabilidade mais vantajosa. O princípio de abandonar os valores minimax está relacionado à suposição de jogadores que jogam perfeitamente. Experimentos realizados nos jogos de Mancala e Othello apresentaram resultados melhores que o alpha-beta.

Tanto o B* quanto o BPIP pode significar o próximo nível de desempenho dos métodos baseados em busca de árvores de jogo. Ambos parecem ser boas idéias na teoria, mas ainda são muito complicados para serem postos em prática. Testes independentes devem ser completados para verificar as reivindicações experimentais dos algoritmos [15].

### 3.2.2.6 – Abstract Proof Search

Inúmeros outros algoritmos foram criados recentemente. Cada pesquisador escolhe um determinado domínio/jogo e mostra para aquela situação específica as vantagens da busca criada sobre outras buscas. É muito difícil avaliar algoritmos recentemente criados ou aplicados a árvores de jogos, uma vez que usualmente poucos pesquisadores fazem testes sobre eles e os testes realizados são feitos em poucos domínios.

De qualquer forma, para não deixar essas novas técnicas ainda não suficientemente experimentadas de fora desta dissertação, foi escolhido um algoritmo mais recente que, na visão do autor desta dissertação, pode se tornar uma promissora técnica no futuro, o "Abstract Proof Search". O "Abstract Proof Search" foi introduzido em 2000 por Tristan Cazenave [25]. Em 2002, Cazenave modificou a técnica, criando o "Gradual Abstract Proof Search" [26].

A proposta do algoritmo é ser um método seguro de escolher movimentos interessantes utilizando as funções de definição do jogo. Ele tem muitas vantagens em relação ao alpha-beta básico: ele resolve mais problemas, as respostas que ele encontra são sempre as corretas, ele resolve os problemas de forma mais rápida e com menos nós e é mais simples de programar do que métodos heurísticos usuais. O único e grande problema é que o algoritmo requer uma análise abstrata do jogo.

Este algoritmo pode ser considerado como um algoritmo que desenvolve pequenas árvores especializadas a cada nó da busca na árvore de jogo. Experimentos foram realizados no jogo da captura, que é o principal sub-jogo do Go, e tiveram sucesso. As funções que selecionavam seguramente um movimento usavam, na prática, muito pouco conhecimento.

O algoritmo consiste em desenvolver pequenas árvores de busca nos nós MIN da busca principal de forma a selecionar os movimentos interessantes ou decidir parar a busca. Dado que o jogador 1 joga com os nós MAX e o 2 com os nós MIN, uma busca "Abstract Proof Search" de ordem um consiste em verificar a cada nó MIN se o jogador 1 pode vencer com um movimento. Se não for esse o caso, a busca é parada e o nó MIN é marcado com "perdido para o jogador 1". De outra forma, se o jogador 1 pode vencer em um movimento, apenas os movimentos do jogador 2 que podem evitar essa vitória em um movimento são considerados e tentados neste nó. Uma posição que é vitória em um movimento para o jogador 1 é marcada com "Win-1" para o jogador 1, e "Forced-1" para o jogador 2. Uma busca de ordem dois consiste em desenvolver árvores com dois movimentos do jogador 1 a cada nó MIN (profundidade 3). Uma busca de ordem três consiste em desenvolver pequenas árvores nos nós MIN com no máximo três movimentos do jogador 1 a partir da raiz (profundidade 5).

Um outro algoritmo que faz parte dos algoritmos "proof search" e que possui conexões fortes com o algoritmo descrito aqui é o Lambda-search [83]. A principal diferença entre esses algoritmos reside no fato que o "Abstract Proof Search" impõe limites na profundidade e na ordem das árvores desenvolvidas a cada nó, enquanto o Lambda search impõem limites na profundidade e na ordem global da árvore.

### 3.2.3 – Algoritmos de busca paralelos

O advento de máquinas de múltiplos processadores gerou muito interesse em todos os campos da computação, nos modos pelos quais as tarefas realizadas nos computadores convencionais podem ser transportadas, com ganho significativo de tempo, para esses equipamentos [60].

Na área de jogos, os primeiros trabalhos datam de 1978, com um grande volume de artigos publicados em 1982 e 1983. É interessante notar que estas pesquisas foram feitas antes que seus autores tivessem acesso a máquinas paralelas concretas: quase sempre foram utilizados simuladores de paralelismo, o que, de certa forma, idealiza o problema. Assim, tanto o desenvolvimento como a análise dos algoritmos propostos não se preocupou com problemas típicos de multiprocessadores, tais como, comunicação, acesso concorrente à memória, deadlock, entre outros [60].

Ao citar algoritmos de busca paralelos em árvores de jogos, é natural que a lembrança do "Deep Blue" apareça. O "Deep Blue", apesar de não ter sido o único nem necessariamente o melhor, tornou-se o programa jogador de Xadrez mais conhecido no mundo inteiro. O projeto começou em 1985, com o estudante de PhD da Universidade Carnegie Mellon, Feng-hsiung Hsu, que construiu o "ChipTest", um simples programa jogador de Xadrez de 1 processador que avaliava 50.000 posições de Xadrez por segundo. Em 1988, O projeto "Deep Thought" foi iniciado baseado em uma versão do "ChipTest". Nesse momento, o programa rodava em 2 processadores e avaliava 720.000 posições de Xadrez por segundo. Em 1989, o projeto "Deep Thought" entrou para a área de pesquisas da IBM (com Hsu). O objetivo era investigar o processamento paralelo. Nesse mesmo ano, o programa enfrentou o campeão mundial Garry Kasparov e foi derrotado (6 processadores que avaliavam juntos 2 milhões de posições de Xadrez por segundo). Em 1991, o "Deep Thought II" ganhou o campeonato internacional de Xadrez da ACM. Em 1993, o projeto foi renomeado para "Deep Blue". Em 1996, o então "Deep Blue" enfrentou Kasparov e alcançou um empate. Finalmente em 1997, com 256 processadores e avaliando 200 milhões de posições de Xadrez por segundo, o "Deep Blue" derrotou Kasparov [43]. A figura 10 mostra a evolução do "Deep Blue" com o passar dos anos.

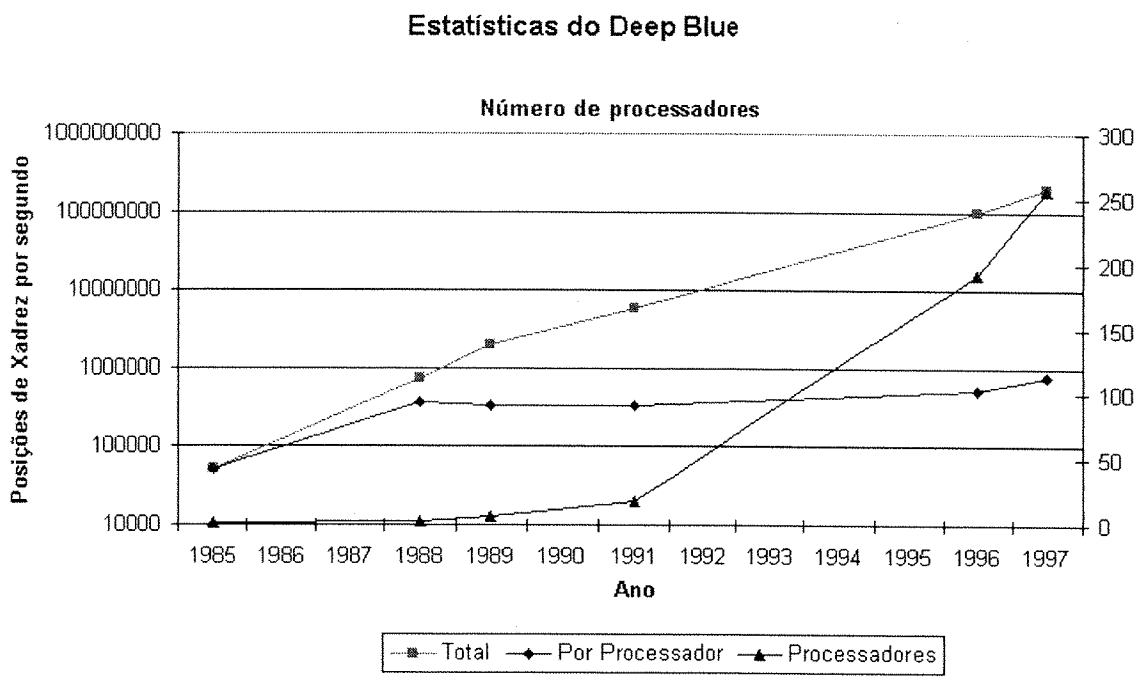



Figura 10 – Gráfico da evolução do "Deep Blue" [41]

Uma das conclusões que é possível tirar do gráfico da evolução do "Deep Blue" é que sem o avanço da tecnologia de hardware, até hoje o "Deep Blue" estaria ainda amargando derrotas consecutivas. No entanto, a evolução dos algoritmos paralelos com o passar dos anos é incontestável.

Esta subseção lida com os algoritmos paralelos de busca em árvore de jogo. Um assunto que, como outros nesta dissertação, poderiam ser assunto de uma dissertação completa e ainda assim não ser possível descrevê-lo em detalhes. A título de exemplo, em 1989, Cláudio Pinhanez [60] descreveu em sua dissertação de mestrado alguns algoritmos paralelos de busca em árvore de jogo já melhor experimentados naquele momento, tais como o Tree-Splitting, o PV-Splitting, o Mandatory Work First (MWF) e o Key Node (KNM).

Em 1996, Mark Brockington [14] sugeriu uma taxonomia para se classificar os inúmeros algoritmos paralelos existentes. Ele dividiu os algoritmos em duas categorias principais: os algoritmos alpha-beta e os outros paradigmas de busca (SSS*, ER e métodos teóricos). A tabela 6 apresenta os algoritmos paralelos baseados no alpha-beta listados por Brockington, agrupados de acordo com a hierarquia do processador e a distribuição do controle. A hierarquia do processador pode ser considerada estática, se um ou mais processadores são designados como Mestres e comandam os outros escravos; ou dinâmica, se a hierarquia não é fixa e pode ser alterada de acordo com o nível de ocupação dos processadores. A distribuição do controle pode ser considerada centralizada, se o controle do algoritmo está em um pequeno número de mestres; ou distribuída, se o controle do algoritmo está distribuído por todos os processadores.

| | Distribuição de controle **Centralizada** | Distribuição de controle **Distribuída** |
|---|---|---|
| Hierarquia do processador **Estática** | Parallel Aspiration Search (1978)<br>Mandatory Work First (1979)<br>Tree Splitting (1980)<br>PV-Split (1981)<br>Key Node (1983)<br>UIDPABS (1986)<br>Delayed Branch Tree Expansion (1990)<br>CABP (1994)<br>APHID (1996) | |
| Hierarquia do processador **Dinâmica** | DPVS (1987)<br>EPVS (1987) | Waycool (1987)<br>Young Brothers Wait (1987)<br>Dynamic Tree Splitting (1988)<br>Branch-and-Bound (1988)<br>Frontier Splitting (1993)<br>αβ* (1993)<br>Jamboree (1994)<br>ABDADA (1995)<br>Dynamic Multiple PV-Split (1995) |

Tabela 6 – Algoritmos paralelos de busca em árvore de jogo

Todos os algoritmos listados na tabela 6 são síncronos, isto é, possuem nós de sincronização onde o paralelismo fica restrito, exceto pelo APHID (Asynchronous Parallel Hierarchical Iterative Deepening) que é assíncrono. Em 1998, Brockington realizou experimentos com APHID mostrando que algoritmos assíncronos podiam se igualar ou até serem melhores que os algoritmos síncronos [15].

Os algoritmos síncronos citados possuem quatro problemas principais. Primeiro, o número de pontos de sincronização global ao longo da variação principal resulta em tempo de processamento desperdiçado. Segundo, esses algoritmos necessitam utilizar uma tabela de transposição compartilhada e/ou recursos de aprofundamento iterativo para obter uma boa ordenação de movimentos e um desempenho razoável. Terceiro, o programa poderia iniciar paralelismo em nós que seriam melhor avaliados seqüencialmente (se algum galho fosse cortado, outros processadores poderiam estar desperdiçando tempo avaliando o resto da sub-arvore). Quarto, os algoritmos de busca paralelos necessitam de um grande esforço de engenharia de software para serem instalados em uma aplicação já existente [15].

Por sua vez, o algoritmo APHID é assíncrono em sua natureza. Todos os pontos de sincronização global, utilizados em outros algoritmos, foram retirados da arquitetura deste. Ele também não necessita de uma tabela de transposições compartilhada para ordenar movimentos, embora uma possa ser utilizada. O APHID também só aplica paralelismo a nós que tenham alta probabilidade de precisar de paralelismo.

Para se comparar o desempenho de algoritmos paralelos em árvores de jogos, algumas medidas podem ser utilizadas: aceleração (tempo tomado pelo algoritmo seqüencial mais rápido dividido pelo tempo tomado pelo algoritmo paralelo), eficiência (aceleração dividida pelo número de processadores utilizados) ou overhead (número de nós

avaliados pelo algoritmo paralelo dividido pelo número de nós avaliados pelo algoritmo seqüencial). Porém, determinar que um dado algoritmo paralelo é melhor que outro é uma tarefa um tanto quanto problemática, uma vez que pode levar a resultados enganadores. Isso porque as acelerações simuladas são calculadas em cima de um modelo simplificado e é difícil determinar o comportamento em uma aplicação real com inúmeros processadores. As árvores de jogo artificiais utilizadas raramente refletem as propriedades de uma árvore de jogo real. Mesmo que fossem utilizadas árvores de jogo reais, os resultados acabariam dependendo do algoritmo seqüencial ou da ordenação de movimentos. Os diferentes fatores de ramificação também podem resultar em acelerações bem diferentes para um mesmo algoritmo. Até mesmo a velocidade do processador ou da rede podem afetar os resultados finais. Por esses motivos, valores de aceleração calculados em outros trabalhos, não serão comentados aqui.

Apesar dos algoritmos paralelos baseados em alpha-beta serem usualmente enfatizados, existem diversos algoritmos baseados em outras abordagens. É o caso dos algoritmos HYBRID [50], MDSSS [73] e SDSSS [29] que são baseados no SSS*. Caso também do algoritmo "CCNS paralelo" baseado no cn-search. Steinberg e Solomon introduziram o algoritmo ER (Evaluate-Refute) [74]. Em experimentos realizados, o ER se mostrou menos eficiente que o alpha-beta.

Existem apenas algumas poucas bibliotecas famosas de busca paralela. Algumas delas se concentram apenas em implementar soluções "branch-and-bound" paralelas de forma eficiente, como é o caso da PPBB-lib (Portable Parallel Branch-and-Bound Library) [85]. Um usuário que precise usar a PPBB-lib não precisará de conhecimento nem da arquitetura de hardware nem dos mecanismos de paralelização. Algumas pesquisas foram realizadas com a ajuda dessa biblioteca. A biblioteca ZRAM [17] já corresponde a um projeto mais ambicioso. Trata-se de uma biblioteca portável de estruturas de dados e de algoritmos de busca paralela. Entre os algoritmos de busca incluídos estão o branch-and-bound, o "reverse search" e o "backtracking". Esta biblioteca também auxiliou a produção de diversos projetos.

# Capítulo 4

# Fim de jogo

Os jogos de tabuleiro podem ser divididos em 3 categorias segundo a propriedade de convergência [3] definida na introdução. A primeira categoria consiste nos jogos divergentes onde as peças do tabuleiro aumentam no decorrer do jogo. É o caso de jogos como jogo da velha, Lig-4, Go moku, Go, Othello, Hex e Amazons. A segunda categoria consiste nos jogos imutáveis onde o número de peças não necessariamente aumenta ou diminui no decorrer do jogo. É o caso do jogo de Shogi onde as peças retiradas podem voltar ao jogo evitando que a quantidade de peças na mesa diminua. A terceira categoria consiste nos jogos convergentes onde o número de peças no tabuleiro diminui no decorrer do jogo, pois um jogador pode realizar movimentos de captura de peças. Jogos que pertencem a essa categoria são: Xadrez, Damas, Gamão, Mancala e "Lines of Action" [38].

Em qualquer das categorias acima, algoritmos de busca podem provar o resultado do jogo para posições próximas do final do jogo. Entretanto, para jogos das duas primeiras categorias, o número de posições de fim de jogo é tão grande que é praticamente impossível enumerar todas elas (exceto para jogos triviais como "Jogo da velha"). Para jogos convergentes, o número de posições próximo do final do jogo é usualmente pequeno o suficiente para enumerar todas as posições e obter os valores de resultado de jogo de cada uma delas. Esses valores são armazenados em um banco de dados chamado de banco de dados de fim de jogo.

A estratégia de utilizar bancos de dados de fim de jogo é muito poderosa uma vez que deixa à disposição do programa jogador não o resultado de uma avaliação heurística de uma posição de jogo e sim o seu valor real. Isso significa que ao encontrar a posição atual do jogo em seu banco de dados de fim de jogo, o programa jogador já conhece o resultado do jogo caso ambos os jogadores joguem perfeitamente. Caso o adversário não jogue perfeitamente, o resultado ali gravado significa a melhor possibilidade de resultado que o adversário irá conseguir. Mais do que isso, utilizando as técnicas de busca relatadas no capítulo anterior, o programa pode ter conhecimento do resultado do jogo muitas jogadas antes de se atingir o banco de dados em questão.

Infelizmente, essa estratégia não pode ser utilizada com o mesmo nível de resultado em todos os jogos convergentes. Para o jogo de Damas, por exemplo, o banco de dados de fim de jogo tem um papel importantíssimo na qualidade de jogo de um programa enquanto no jogo de Xadrez isso não ocorre da mesma forma. A maioria dos jogos de Xadrez nunca dura tempo suficiente para atingir o seu banco de dados [70]. Isso se deve principalmente a 2 fatores: a complexidade do espaço de estados do jogo de Xadrez é muito maior (devido aos diferentes tipos de peças); e o jogo de Xadrez termina ao encurralar o Rei (xeque-mate) e não ao capturar todas as peças do adversário como no jogo de Damas (ou seja, no final ainda podem existir muitas peças de ambos os jogadores no tabuleiro no caso do Xadrez).

A abordagem de final de jogo é produzir um banco de dados que possua todas as situações de final de jogo interligadas de forma que seja possível ao programa saber precisamente como jogar até o final e já saber o resultado do jogo naquele momento. O

Chinook, por exemplo, possui um banco de dados de fim de jogo de Damas com 8 peças ou menos quaisquer que elas sejam (exemplo de configuração: 3 peças pretas, 4 peças vermelhas e 1 dama vermelha). Isso corresponde a mais de 400 bilhões de posições. O arquivo contendo essas informações possui cerca de 6 Gigabytes e pode ser encontrado na internet em [82].

Este capítulo descreve as técnicas de construção, representação, compactação e acesso a bancos de dados de fim de jogo. Apesar de se utilizarem as mesmas técnicas para todos os jogos, a aplicação das mesmas costuma diferir de jogo para jogo devido às diferenças de propriedades e regras. Sendo assim, foi eleito o jogo de Damas para a demonstração dessas técnicas nas seções a seguir. A aplicação para outros jogos é semelhante, mas, como comentado anteriormente, a transposição dessa técnica para outro jogo não é necessariamente trivial.

## 4.1 – Construção de banco de dados de fim de jogo

A construção desses bancos de dados não costuma ser trivial e a técnica utilizada com mais sucesso recebe o nome de análise retrocessiva.

A Análise Retrocessiva ("Retrograde Analysis"-RA) calcula bancos de dados de fim de jogo inserindo valores das posições finais até as iniciais. Segue o caminho contrário do alpha-beta que busca a partir de posições iniciais até as finais. Inicialmente, são identificadas todas posições finais onde o resultado do jogo é conhecido (usualmente vitória, empate e derrota). Realizando movimentos reversos a partir dessas posições finais, os resultados de algumas posições não-finais são encontrados. São feitos movimentos reversos a partir das novas posições encontradas e assim por diante até o infinito [38]. Uma das grandes vantagens dessa abordagem é que o resultado de uma posição é baseado nas posições decorrentes dos movimentos desta, não necessitando de uma busca mais profunda.

Segundo Goot [38], Ströhlein foi o primeiro pesquisador a surgir com a idéia de criar bancos de dados de fim de jogo. Ele aplicou sua idéia ao Xadrez. Ken Thompson calculou muitos bancos de dados de 4, 5 e 6 peças de Xadrez. No jogo de Damas, Jonathan Schaeffer e outros construíram o banco de dados do Chinook, o que possibilitou a vitória do campeonato mundial homem-máquina de Damas. Ralph Gasser provou que o jogo de Trilha é um empate criando um banco de dados e depois realizando uma busca do início até o banco de dados.

Para o caso do jogo de Damas, a construção de um banco de dados de N peças consiste em enumerar todas as posições com N peças ou menos no tabuleiro e calcular se o valor é vitória, empate ou derrota. A idéia básica é simples, o que torna o problema difícil é o aumento de tamanho do problema a medida que N aumenta [69]. A Tabela 7 mostra a quantidade de posições válidas de fim de jogo para 6 peças ou menos no tabuleiro do jogo de Damas.

| Brancas/pretas | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 3.488 | 98.016 | 1.773.192 | 23.204.660 | 233.999.928 |
| 2 | 98.016 | 2.662.932 | 46.520.744 | 587.139.846 | |
| 3 | 1.773.192 | 46.520.744 | 783.806.128 | | |
| 4 | 23.204.660 | 587.139.846 | | | |
| 5 | 233.999.928 | | | | |

Tabela 7 – Número de posições válidas de fim de jogo de Damas

Para o caso de uma peça preta e uma branca, teremos 4 casos: Dama Branca (DB) e Dama Preta(DP); Dama Branca e Peça Preta (PP); Peça Branca (PB) e Dama Preta; e Peça Branca e Peça Preta. Sabe-se que existem 32 posições válidas para Damas e 28 para Peças (exclui-se a última linha de cada lado, pois a peça vira Dama ao atingir essa linha). Sendo assim, para o primeiro caso (DP-DB), teremos 32*31=992 posições. Para o segundo caso, temos 2 situações: situação em que a DB está sobre a última linha (4*28=112 posições) e a situação em que DB não está sobre a última linha (28*27=756 posições), somando assim 868 posições. O terceiro caso tem o mesmo número de posições do segundo (868), pois basta que se invertam as cores. O último caso também tem 2 situações segundo PP: Se PP estiver sobre a última linha das brancas (4*28=112) e se não estiver (24*27=648), somando assim 760 posições possíveis para PP-PB. O total 992+868+868+760=3.488 corresponde ao número de posições válidas de 1 peça para cada lado. Assim por diante para se calcular o resto da tabela.

O Banco de dados de N peças é calculado usando um algoritmo iterativo simples, após o banco de dados de 1,2,..., (N-1) peças estarem completos. Inicialmente, todas as posições com N peças são vistas como tendo o valor "desconhecido". Cada iteração examina todas as posições tentando determinar se já existe informação suficiente para trocar o valor de uma posição de "desconhecido" para vitória, empate ou derrota. Dado que as pretas jogam, seu valor é determinado pelas seguintes regras, em ordem de precedência (se as brancas jogam, basta reverter as cores):

(1) Pretas ganham se existe um movimento para a derrota das brancas

(2) O valor continua "desconhecido" se existe um movimento para uma posição das brancas cujo valor é "desconhecido".

(3) Pretas empatam caso exista um movimento para empate das brancas e as pretas não conseguem vencer.

(4) Pretas perdem caso todos os movimentos levem a uma vitória das brancas.

O programa deverá realizar iterações até que não seja possível calcular mais posições de N peças. Neste ponto, todas as posições com valores "desconhecido" são empates [69].

No início, metade das posições é de captura. Seus valores são determinados na primeira iteração, pois todas as capturas levam a posições com (N-1) ou menos peças cujos valores já são conhecidos.

A figura 11 mostra uma situação clássica do jogo de Damas (regras americanas onde as damas andam apenas uma casa por jogada) em que as pretas jogam e ganham. A solução é trivial para qualquer jogador humano: Pretas (P): E3-F2, Brancas(B): H2-G1, P: F4-E3, B: G1-H2, P: F2-G1, B: H2-G3, P: G1-H2, B: G3-H4, P: E3-D4, B: H4-G5, P: D4-E5, B: G5-H4 ou G5-H6, P: E5-F6, B: (qualquer jogada é capturada). Para um humano, a

estratégia é simples e possui 2 passos: Tirar a dama adversária do canto duplo; e encurralar a dama adversária em um canto simples. Para um algoritmo de busca, acertar essa mesma estratégia pode significar 15 níveis de uma árvore de fator de ramificação médio de 5.

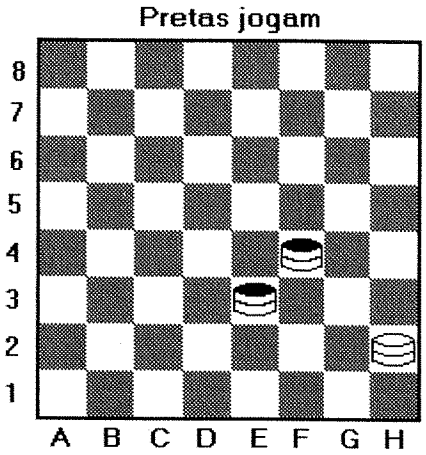



Figura 11 – Duas damas pretas contra uma dama branca

Imaginando na mesma figura 11 que as damas pretas estão na parte superior esquerda do tabuleiro. A estratégia do ponto de vista humano ganha um passo adicional: encurralar a dama adversária no canto duplo, depois se seguem os 2 passos normalmente. Para um algoritmo de busca, isso significaria muitos níveis a mais na árvore e a chance de talvez não encontrar a estratégia correta e acabar empatando o jogo (devido ao número de lances) ou até perdendo (devido ao tempo). Essa é uma das grandes vantagens de se criar um banco de dados de fim de jogo que contém armazenado o caminho até a vitória mais curta (nem todos são assim, como será apresentado na próxima seção).

A análise retrocessiva aqui explicada nem sempre é utilizada da mesma forma. Algumas variações utilizam geração de movimentos normais, em vez de movimentos reversos e propagam valores desdobrados, em vez de valores fixos. Para citar exemplos de diferenças, a proposta de [38] para o jogo de Mancala é, em vez dos resultados serem na forma de valores de vitória, empate e derrota, os mesmos serem na forma de número de peças que podem ser ganhas (-N a +N). Isso, na realidade, é uma otimização para reduzir o número de posições que podem ser atingidas ao excluir as peças das posições. Outras diferenças dizem respeito a aspectos como economia de memória e custos computacionais.

Além do já comentado, a adoção de técnicas para construção de bancos de dados de fim de jogo tem um efeito colateral importantíssimo, mas que não tem relevância computacional. É possível através dessas informações, surgirem novas estratégias a serem inseridas na literatura de cada jogo para aprimorar a técnica dos humanos. Alguns pesquisadores escreveram artigos inteiros visando a comunidade daquele jogo específico. É o caso de [49], um artigo que discute a situação de final de jogo de Damas onde existem 3 damas pretas e 1 peça preta contra 3 damas brancas. Se essa situação de jogo fosse iniciada aleatoriamente, a peça preta estivesse na quarta linha, fosse a vez das pretas jogarem e não fosse uma situação de captura, a chance de vitória das pretas seria de 91.74%, 8.19% para empate e apenas 0.07% de vitória para as brancas. Mesmo com essas estatísticas, essa situação é bastante complexa para um jogador humano que, para vencer com as pretas, teria talvez que fazer mais de 70 jogadas precisas para vencer as brancas.

## 4.2 – Representação de banco de dados de fim de jogo

Tanto a questão de representação quanto de armazenamento das informações em um banco de dados de fim de jogo estão diretamente ligadas com a qualidade de jogo do programa jogador.

Os fatores principais relativos à representação e ao armazenamento são:

- Valores de resultado da posição de jogo
- Caminho até a vitória (ou a melhor situação possível)
- Menor caminho até a vitória

As decisões na forma de representar e do que armazenar podem depender do tipo de jogo e até do algoritmo utilizado para o meio de jogo, mas dependem também de aspectos como desempenho e tamanho de arquivo de banco de dados que precisam ser balanceados com a qualidade do resultado final.

Essas decisões podem gerar diferentes níveis de estratégia no fim de jogo. Uma estratégia para um jogador é um conjunto de regras que o mesmo segue para fazer o próximo movimento utilizando a informação disponível. Uma estratégia infalível é uma que pode ser utilizada para vencer um jogo iniciando de uma situação de vitória garantida, ou não perder um jogo a partir de uma situação de empate garantido. Uma estratégia perfeita não é apenas infalível como significa jogadas ótimas no sentido da vitória mais rápida ou da derrota mais lenta possível. A estratégia pode depender da quantidade de informação disponível [33].

A maioria dos programas jogadores de Damas, por influência do Chinook, não armazenam nenhum grafo nem informações adicionais no banco de dados de fim de jogo, mas apenas os valores de resultado de cada posição de jogo: vitória, empate ou derrota. A informação é armazenada em apenas 2 bits por posição de jogo visando manter o tamanho de arquivo o menor possível garantindo assim mais posições de jogo armazenadas. Para o caso do banco de dados de 6 peças (ou menos), existem 2.571.945.320 posições possíveis. Isso significa um tamanho de arquivo não compactado de 613MB (o arquivo do Chinook disponível na internet tem menos de 30MB e será assunto da próxima seção). O objetivo de manter pequeno o arquivo era que o mesmo coubesse na RAM do computador uma vez que seria freqüentemente acessado pelos algoritmos de meio de jogo.

A estratégia adotada implica em alguns riscos para o programa jogador. Mesmo estando em uma posição marcada como vitória, não existe informação suficiente para saber como jogar para vencer, uma vez que mesmo escolhendo entre um dos movimentos que leve a uma posição marcada como vitória, não há como garantir que o programa vai convergir para uma vitória mesmo ou simplesmente ficar circulando indefinidamente por posições marcadas como vitória. Para tanto, são utilizados os algoritmos de busca de meio de jogo para tentar encontrar um caminho vitorioso. Mesmo existindo a possibilidade de falha, uma vez que algumas posições podem exigir cerca de 100 movimentos até a vitória, a experiência relatada pelos produtores do Chinook é que ele sempre conseguiu encontrar os caminhos nos jogos em campeonatos.

O outro risco é a escolha entre caminhos para empates e derrotas. Uma vez que o adversário pode não conhecer perfeitamente todos os caminhos até a vitória (ou empate),

uma boa estratégia pode ser jogar o caminho mais longo até a derrota podendo assim contar com um erro do adversário e reverter a situação do jogo. Claro que essa estratégia pode não ser a melhor dependendo do adversário, o que dá margem a uma discussão mais profunda conhecida como "modelagem de oponente", comentada no capítulo de Aprendizado.

Os programas jogadores de Xadrez usualmente evitam essa estratégia e armazenam também a informação da distância até a vitória. Uma vez em uma posição no banco de dados, os programas convergem para a vitória escolhendo um movimento vencedor que minimize o número de movimentos até a vitória.

Outras abordagens utilizam a representação na forma de Grafos finitos, dirigidos e cíclicos. Seja qual for o jogo de tabuleiro finito de 2 jogadores de soma zero e informação perfeita, as seguintes propriedades são observadas para os estados filhos de um estado não-final em um grafo de estados[33]:

(1) Um estado vencedor tem que ter ao menos um estado filho perdedor.
(2) Um estado perdedor não tem nada além de estados filhos vencedores.
(3) Um estado de empate tem pelo menos um estado filho de empate e nenhum estado filho de derrota.

No caso do jogo de Xadrez Chinês [33], a abordagem segue na forma acima e utiliza uma idéia adicional ao xadrez que garante jogadas perfeitas, porém com aumento no tamanho de arquivo. É armazenada a informação do máximo número de jogadas que o jogador precisa para vencer o jogo caso seja uma posição de vitória ou o mínimo número de jogadas que um jogador pode se defender caso seja uma posição de derrota. Esses cálculos adicionais aumentam os custos computacionais no momento da construção do banco de dados.

## 4.3 – Compactação de banco de dados de fim de jogo

A idéia de reduzir o tamanho dos bancos de dados de fim de jogo é importante para conseguir manter o mesmo na memória RAM da máquina evitando ou apenas reduzindo os lentos acessos ao disco rígido. Os programas comerciais necessitam desse tipo de redução também por outros fatores como a capacidade de um CD-ROM (mídia usualmente utilizada para distribuir esses jogos) ou mesmo a limitação de espaço no disco rígido do usuário final.

Uma estratégia utilizada em alguns jogos é, depois de realizados todos os esforços de compactação específicos, utilizar o algoritmo RLE (run length encoding). O RLE é conceitualmente simples consistindo de busca por repetidas aparições de um único símbolo em um arquivo, substituindo o mesmo por uma única instância do símbolo e uma contagem de aparições. O mesmo já foi utilizado em jogos como Trilha [36] e Damas, por exemplo. Outros algoritmos de compactação também produzem bons resultados.

Novamente, nem todas as técnicas utilizadas são genéricas o suficiente para serem aplicadas a todos os jogos. Os melhores resultados são alcançados por técnicas específicas de cada jogo, que podem ser aproveitadas em alguns casos para outros jogos com pequenas modificações.

Um exemplo de compactação inteligente é a utilização de heurísticas que tenham uma boa exatidão para pequenas porções do banco de dados. Esta função deveria retornar

valores de resultado de jogo para as diferentes posições de um pedaço do banco. Neste caso, só haveria necessidade de se armazenar as exceções (posições incorretamente classificadas pela função heurística). Por exemplo, uma posição no jogo de Damas com 3 Damas pretas contra 3 peças brancas é usualmente uma vitória para as pretas. Posições de derrota ou empate que sejam classificadas como vitória pela função criada seriam armazenadas no banco de dados.

Apesar dessa compactação ser promissora, o problema acima está em criar funções com alta fidelidade. Os criadores do Chinook, por exemplo, relataram ter tentado diversas abordagens, incluindo o uso de redes neurais visando o aprendizado dos pesos das características da função heurística, não conseguindo afinal construir um banco de dados de exceções que fosse menor do que o formato RLE.

Uma das idéias de melhor resultado na compactação do banco de dados do jogo de Damas foi definir as posições como ímpares ou pares dependendo do último bit da soma dos números das casas ocupadas por peças (numeradas de 1 a 32). Apenas as posições marcadas como pares são armazenadas. Este método tem a propriedade que um movimento sem captura sempre resulta na passagem de uma posição ímpar para uma par (e vice-versa), significando que uma pequena busca pode descobrir os valores das posições ímpares ausentes [70].

## 4.4 – Acesso à banco de dados de fim de jogo

Uma vez construído o banco de dados, escolhida a representação, armazenada e compactada a informação, nenhuma alteração a mais é necessária. O arquivo já está pronto para ser levado para o ambiente de execução.

Resta, no entanto, o problema de como o programa irá interagir com esse banco de dados de forma a obter informações precisas e rápidas.

A abordagem mais utilizada é a criação de funções de indexação e desindexação. Essas funções mapeiam cada posição no banco de dados para um único número. O ideal é que não existam espaços entre os números garantindo um menor tamanho de banco de dados.

Funções de indexação podem ser adaptadas para jogos com disposições similares como Damas e Xadrez. No caso do esquema de indexação do Chinook, o banco de dados de fim de jogo foi dividido em pedaços. Cada pedaço contém todas as posições possíveis para uma determinada tupla (ndp,ndb,npp,npb,nlp,nlb), sendo:

ndp = número de damas pretas
ndb = número de damas brancas
npp = número de pedras pretas
npb = número de pedras brancas
nlp = número da linha da pedra preta mais à frente (de 0 a 7)
nlb = número da linha da pedra branca mais à frente (de 7 a 0)

Através de uma fórmula [48], é calculada a quantidade de posições para cada tupla. Os pedaços são então seqüenciados no arquivo de forma a não haver espaços entre eles. O tabuleiro é então numerado de 0 a 31 para todas as casas válidas do jogo de Damas, a partir do lado esquerdo da linha 0 (zero) do lado das pretas. O algoritmo de enumeração

inicialmente coloca as pedras pretas nas casas (0,1,...,npp). As outras peças vão sendo colocadas e os conflitos de posição vão sendo evitados. Cada posição é representada por quatro tuplas: ($PP_0$, $PP1_1$, ..., $PP_{npp}$), ($PB_0$, $PB_1$, ..., $PB_{npb}$), ($DP_0$, $DP_1$, ..., $DP_{ndp}$) e ($DB_0$, $DB_1$, ..., $DB_{ndb}$). Cada entrada, em cada tupla, corresponde aos números das casas ocupadas pelas peças, ordenadas da esquerda para a direita em ordem crescente. Para cada tupla, é calculado um índice único associado.

A função de desindexação recebe como entrada uma posição do tabuleiro. Verifica na matriz armazenada em qual dos pedaços do banco de dados a posição se encontra. Soma a esse pedaço o número do índice calculado a partir das tuplas da posição atual. Maiores detalhes sobre as fórmulas utilizadas podem ser encontrados em [48].

# Capítulo 5

# Aprendizado

A área de aprendizado se tornou recentemente uma das principais áreas de pesquisa da Inteligência Artificial com inúmeras revistas e conferências voltadas para a publicação de resultados. Entre as suas inúmeras aplicações estão as médicas, em telecomunicações, nos esportes, na música, na internet e não menos importante, nos jogos de tabuleiro para computador.

Mesmo sendo uma difícil tarefa, a idéia deste capítulo é lidar com toda a literatura relacionada ao aprendizado em jogos de tabuleiro. A divisão foi feita considerando os diferentes aspectos de um programa jogador, tais como a busca e a função de avaliação. Infelizmente, não é possível detalhar todos os aspectos uma vez que alguns deles, como o caso de aprendizado por diferença temporal (que corresponde a apenas um pequeno trecho do item "aprendizado por reforço" da sub-seção "aprendizado de ajuste de função de avaliação" deste capítulo), poderiam ser assunto de uma dissertação inteira e ainda assim não serem completamente descritos.

Um dos grandes sonhos da IA era fazer um computador aprender pela prática por si só. Este sonho foi atingido por um jogo de Gamão utilizando aprendizado por reforço como paradigma de aprendizado e uma rede neural como representação. Trata-se de uma das mais impressionantes aplicações que utilizam aprendizado por reforço.

Este programa, TD-Gammon, produzido por Gerald Tesauro [79], em 1995, utilizava o mínimo de conhecimento do jogo de Gamão e aprendeu a jogar extremamente bem. O programa campeão antes do TD-Gammon, era o NeuroGammon, construído a partir de uma rede neural e de aprendizado supervisionado com a ajuda de especialistas do jogo. O TD-Gammon, após ser treinado jogando 300.000 jogos contra ele próprio, já conseguia derrotar qualquer outro programa jogador e só perdia para os grandes mestres do jogo [77].

Em um caso clássico de aprendizado de função de avaliação, o TD-Gammon inicialmente sequer realizava buscas na árvore de jogo, apenas avaliava a situação atual do tabuleiro e decidia segundo o que já tinha aprendido. Logo em seguida, surgiram iniciativas de pesquisas na mesma linha para outros jogos diferentes do Gamão.

Inicialmente, neste capítulo, serão descritas as técnicas de aprendizado relativas ao controle de busca em árvore de jogo. Posteriormente, serão mostradas as diferentes formas de aprendizado de funções de avaliação. Também serão apresentados grandes problemas onde não houve muitos avanços até então, tal qual a modelagem de oponente, a descoberta automática de características de um jogo, entre outras. O jogo eleito para ser melhor comentado neste capítulo é o jogo de Gamão.

## 5.1 – Aprendizado de controle de busca

Uma implementação típica de um programa jogador baseado em busca tem uma grande quantidade de parâmetros que podem ser utilizados para tornar a busca mais eficiente. Entre eles, estão a profundidade de busca, as extensões de busca, parâmetros da busca quiescente, heurísticas de ordenação de movimentos, etc [35]. Mesmo sendo intratável o problema de descrever todas as técnicas de aprendizado de controle de busca em jogos de tabuleiro, aqui serão descritas técnicas mais conhecidas por terem obtido sucesso em algum jogo específico.

A otimização automática de parâmetros de busca, por vezes, pode ser mais valiosa do que o ajuste de parâmetros de uma função de avaliação. Essa constatação foi feita por alguns pesquisadores, entre eles Donninger [30] em seu programa jogador de Xadrez, o Nimzo. A otimização de parâmetros tinha mais conseqüências na qualidade de jogo do programa que o ajuste da função de avaliação. Essa mesma idéia também é compartilhada por outros pesquisadores como Baxter, Tridgell e Weaver [7] que consideram o tópico de aprendizado de busca seletiva como um tópico compensador para estudos futuros.

Para os programas jogadores baseados em busca, é imediato perceber que quanto mais profunda for a busca na árvore de jogo, melhores serão as chances do programa de vitória (considerando uma boa função de avaliação). O problema é que, mesmo com as otimizações trazidas pelo algoritmo alpha-beta e seus avanços (tabelas de transposição, janelas nulas, ordenação de movimentos, profundidade variável), as buscas não conseguem atingir profundidades muito grandes em tempo real (dados os recursos computacionais atuais) ficando por vezes limitadas a um ponto onde seres humanos conseguem prever bem mais adiante. Uma otimização viável é o estudo das heurísticas para buscas seletivas que conseguiriam, para um determinado jogo, cortar galhos da árvore de jogo que não deveriam ser avaliados por algum motivo.

O programa jogador de Go, Honte, utiliza uma abordagem para melhor selecionar suas jogadas. Como seu objetivo é enfrentar grandes mestres do jogo, ele tenta expandir apenas jogadas do lado do humano na hora da busca que ele acredita serem as mais prováveis. A partir de uma coletânea de jogos de especialistas, uma rede neural foi treinada para, baseado nas formas das peças no tabuleiro, estimar a probabilidade de um especialista realizar aquele tipo de jogada. A qualidade da "forma" é um conceito muito importante para os humanos jogadores de Go. É comum até que justifiquem suas jogadas como "uma jogada para criar uma boa forma" [28].

Outro exemplo clássico da utilização da busca seletiva é no caso do programa jogador de Othello, o Logistello. A técnica que recebeu o nome de ProbCut foi introduzida por Michael Buro em 1995 com o objetivo de cortar prováveis sub-árvores irrelevantes. A idéia base é estimar os valores de uma busca mais profunda através de uma busca mais rasa e relaxar as condições do algoritmo alpha-beta introduzindo um novo parâmetro, o nível de confiança de corte T. A partir daí, deveriam ser realizadas inúmeras partidas de um jogador com ProbCut contra um jogador sem ProbCut, sempre alterando o valor de T para descobrir qual o melhor valor para utilizar com aquele programa. Em uma experiência realizada pelo próprio Buro no jogo de Othello, ele chegou a um valor de T~1.5 que ganhou 74% dos pontos contra o jogador sem o ProbCut. Em 1999, Buro generalizou a técnica para cortes em múltiplos níveis e com múltiplos valores de T para cada fase do jogo. Esta técnica recebeu o nome de Multi-ProbCut. Experimentos realizados no jogo de Othello

apresentaram vitória de 72% dos pontos para o algoritmo que utilizava MultiProbCut contra o que utilizava apenas o ProbCut. Ainda é uma incógnita se essa técnica pode ser generalizada para outros jogos de forma bem-sucedida como foi utilizada no jogo de Othello, isso porque é fato conhecido que este jogo apresenta forte correlação entre valores de buscas rasas e de buscas profundas.

Alguns pesquisadores investiram mais esforços em tentar aprender as heurísticas de ordenação de movimentos para aumentar a eficiência do alpha-beta. Uma ordenação mais eficiente não altera o resultado final do algoritmo, mas diminui a quantidade de nós avaliados e resulta conseqüentemente em buscas mais rápidas. Isso possibilita buscas mais profundas que podem levar a uma melhor qualidade ao final. Em 1999, Greer, Ojha e Bell [40] treinaram uma rede neural para aprender a identificar regiões importantes de uma posição de Xadrez e ordenar os movimentos de acordo com a influência exibida sobre essas regiões. Também em 1999, Inuzuka sugeriu o uso de programação de lógica indutiva para aprender uma relação de preferência binária entre movimentos. Essas iniciativas, mesmo estando na direção certa, ainda não fazem frente a técnicas convencionais de ordenação de movimentos tais como a heurística histórica [67] e a heurística assassina [2].

Em 2002, Yngvi Björnsson e Tony Marsland demonstraram uma arquitetura que permite aprender a decidir quanto à profundidade que um determinado nó deve ser expandido (esse problema é também conhecido como problema da alocação de tempo, uma vez que em campeonatos existe uma restrição de tempo e o programa precisa decidir quais nós avaliar ou não dentro dessas restrições). O estudo realizado se aplica tanto a treinamento sobre posições de jogo quanto treinamento online enfrentando adversários reais. O treinamento online tem sido muito utilizado para testar o desempenho de programas jogadores e para garantir o aprendizado completo jogando contra seres humanos de diferentes estilos e níveis. A arquitetura também se aplica a qualquer tipo de jogo. Experiências foram realizadas com o programa jogador de Xadrez Crafty (o melhor programa jogador de Xadrez disponível publicamente, ele se encontra em alguns servidores online e a sua classificação costuma ser bem superior a de todos os jogadores) e se comprovaram bem-sucedidas neste caso.

Aparentemente, o problema de aprendizado dos diversos parâmetros relativos ao controle de busca é mais difícil que o problema de ajuste de função de avaliação que será visto na seção a seguir. Isso porque é difícil separar esses parâmetros do algoritmo de busca que é controlado por eles [35].

## 5.2 – Ajuste de função de avaliação

O problema de aprendizado mais estudado em jogos de tabuleiro é o de ajuste automático de pesos de uma função de avaliação. Tipicamente, são calculadas diversas propriedades de um jogo em uma determinada situação do tabuleiro: quantidade de peças, estabilidade, mobilidade, tamanho do território controlado, entre outras. O problema do ajuste consiste em como combinar essas propriedades e quantificar a sua importância relativa.

Muitos programas campeões como, por exemplo, o Chinook, utilizam funções que correspondem a uma combinação linear de componentes cujos pesos de cada um são ajustados manualmente. No caso do Chinook, são 24 propriedades diferentes, cada uma com peso produzido por ajuste manual. Os pesquisadores chegaram a tentar diversas

abordagens para ajustar os pesos por processos automáticos tais como equações lineares, redes neurais e algoritmos genéticos. Não chegaram a nenhum resultado melhor do que o ajuste manual. Isso já aconteceu de uma forma diferente a jogos como Gamão e Othello, onde o aprendizado gerou resultados surpreendentes.

Apesar dos progressos, existem alguns grandes problemas relativos ao aprendizado de funções de avaliação, tais como a utilização de funções de avaliação não-lineares, as estratégias de treinamento, o aprendizado em buscas e a descoberta automática de características.

Muitos programas jogadores convencionais dependem de algoritmos de busca velozes e, portanto requerem uma função de avaliação que seja rapidamente calculada. Uma combinação linear de propriedades é uma solução óbvia para isso, mas nem sempre representa a melhor forma de avaliar situações de um jogo. Em 1967, Samuel sugeriu a utilização de tabelas de assinatura, uma estrutura não-linear em camadas de tabelas "look-up". Em 1988, Lee e Mahajan interpretaram os seus resultados obtidos por aprendizado bayesiano em comparação com uma função linear ajustada à mão como uma evidência de que uma função não-linear é melhor do que uma função linear (eles mostraram que as matrizes de covariância que eram a base da função bayesiana exibiam correlação positiva para todos os termos). Em 1995, Buro mostrou que as correlações eram uma condição necessária, mas não suficiente para se utilizar uma função não-linear. Em 1998, Tesauro mostrou que eventualmente uma estrutura não-linear pode ser crucial para desempenho. Uma técnica simples para alcançar a não-linearidade é usar funções de avaliação diferentes nas diversas fases do jogo.

O problema relativo às estratégias de treinamento é o de como prover ao programa aprendiz uma informação de treinamento que seja ao mesmo tempo focada o suficiente para garantir convergência rápida para uma boa função de avaliação, e também suficientemente variável para permitir o aprendizado de funções aplicáveis geralmente. Em particular, no aprendizado realizado por autoconfronto, é preciso garantir uma boa variedade de situações durante o treinamento para evitar que ele se limite a um mínimo local. Em 1994, Epstein apresentou algumas evidências experimentais que o autoconfronto não gera um bom aprendizado em jogos determinísticos. Ele sugeriu a utilização do "aprenda e pratique", que significava alternar lições entre autoconfronto e jogos com especialistas. Acabou por produzir o sistema de aprendizado de jogo Hoyle [31]. Alguns pesquisadores optaram por sortear os primeiros movimentos de um jogo para assegurar a variação. Em 1994, Angeline e Pollack evoluíram várias instâncias de seu programa jogador através de algoritmos genéticos. A aptidão dos jogadores de cada geração era determinada fazendo eles jogarem campeonatos entre si. Uma outra forma de treinamento que tem sido freqüentemente utilizada é o treinamento em servidores de internet. Jogando contra humanos de diferentes níveis, o programa garante uma grande variação para melhorar seu nível.

Em alguns jogos como o de Gamão, buscas profundas são impraticáveis devido ao grande fator de ramificação da árvore de jogo. No caso do Gamão, esse grande fator de ramificação se deve ao fator sorte, introduzido pelo uso do dado (existem 21 resultados de dados diferentes com uma média de 20 movimentos possíveis para cada um). Outros jogos como o de Damas possuem fator de ramificação médio inferior a 3 (graças aos movimentos de captura que limitam as jogadas por serem obrigatórios) em contraste com o fator 400 do Gamão. Curiosamente, alguns jogos como Damas e Xadrez necessitam de buscas profundas para se atingir o nível de um especialista. O problema está em como integrar o aprendizado com essas técnicas de busca. A solução para esse problema está em basear a avaliação na

posição dominante, em vez da posição atual do jogo. A posição dominante é uma posição folha da árvore de jogo expandida a partir da posição atual. Dessa forma, os valores são propagados para a raiz da árvore. Aparentemente, esse problema foi resolvido por Samuel em 1959. Em 1993, no entanto, Gherrity publicou sua tese em uma arquitetura de sistema que integra o aprendizado e a busca em uma variedade de jogos, tais como Xadrez e Lig-4. Em 1998, Beal e Smith aplicaram técnica similar para o aprendizado de valores de peça no jogo de Shogi.

O ponto crucial de todos as abordagens que ajustam funções de avaliação é a presença de características cuidadosamente selecionadas que capturam informações importantes sobre o atual estado do jogo, e que vão além da posição das peças no tabuleiro. No Xadrez, conceitos como segurança do rei, controle do centro ou mobilidade, são comumente usados para avaliar posições e abstrações similares são utilizadas em outros jogos [35]. Em 1989, Tesauro e Sejnowski observaram um melhora de 15% a 20% nos resultados do programa após adicionar características conhecidas, tipicamente consideradas por especialistas, à rede neural do programa jogador de Gamão. A idéia do aprendizado aqui não é ajustar os pesos, mas descobrir automaticamente características de um jogo. Um dos grandes passos realizados na direção dessa descoberta automática foi o desenvolvimento de procedimentos de treinamento automatizados para redes neurais de múltiplas camadas. Diferentemente da perceptron (rede neural de uma camada), essas redes possuem ao menos uma camada escondida. Os pesos para e dessa camada são normalmente iniciados com valores aleatórios. Durante o treinamento, esses pesos vão sendo alterados para valores que facilitam o aprendizado do sinal de treinamento e conceitos vão aparecendo nessas camadas. No seu famoso programa TD-Gammon, Tesauro interpretou suas camadas escondidas como sendo detectores de características orientados ao ataque e à concorrência. A desvantagem dessas camadas é que elas não são imediatamente interpretadas. Em 1992, Fawcett e Utgoff discutiram o sistema Zenith, que automaticamente produzia características para uma função de avaliação linear do jogo de Othello. Cada característica era representada com uma fórmula em cálculo de predicado de primeira ordem. Em 2001, Utgoff descreveu detalhadamente o problema da construção automática de características em [87].

As abordagens conhecidas que resolvem problemas de aprendizado para ajuste de função de avaliação podem ser categorizadas de várias formas. Aqui, essas abordagens serão divididas quanto ao tipo de informação de treinamento recebido por elas. As subseções a seguir descrevem em detalhes essas formas de aprendizado.

### 5.2.1 – Aprendizado supervisionado

Uma abordagem para aprendizado de pesos de uma função de avaliação mais direta é fornecer ao programa posições exemplos para a qual o valor exato da função de avaliação é conhecido. O programa então tenta ajustar os pesos de uma forma a minimizar o erro da função de avaliação nestas posições. A função resultante, aprendida por otimização linear ou alguma técnica de otimização não-linear como treinamento por propagação reversa para redes neurais, pode então ser utilizada para avaliar novas posições [35].

Essa é a idéia básica do aprendizado supervisionado de uma função de avaliação. Nem sempre os valores exatos são considerados importantes para este tipo de treinamento. Alguns treinamentos simplesmente indicam a direção em que os pesos devem ser ajustados.

Esse tipo de treinamento é normalmente realizado com o auxílio de especialistas no jogo. Estes usualmente não são capazes de avaliar uma posição de uma forma quantitativa, mas sim qualitativa. Para o especialista, é fácil dizer se existe uma vantagem para um dos lados ou se a partida está mais ou menos empatada, porém é incomum conseguir avaliar que existe 25% de vantagem para um dos jogadores, por exemplo.

O Aprendizado supervisionado é um dos grandes culpados pelos programas jogadores realizarem jogos semelhantes a jogos de humanos. Como eles são treinados por humanos e por partidas jogadas por humanos, acabam imitando esse tipo de comportamento no momento do jogo. Essa deve ser a provável explicação para a dúvida de Garry Kasparov ao perder do "Deep Blue". Ele acreditava que uma das jogadas parecia ter sido feita por um humano. Existem até alguns programas comerciais jogadores de Xadrez que simulam um determinado estilo de jogo baseado em algum grande mestre. É o caso do Chessmaster que simula o estilo do próprio Kasparov e muitos outros jogadores.

Em 1984, Mitchell utilizou a técnica de aprendizado supervisionado a partir de valores exatos de uma função de avaliação no jogo de Othello. Ele selecionou 180 posições, ocorridas em campeonatos, a partir do 44° movimento e calculou seus valores exatos através de uma busca até o fim do jogo. Ele utilizou esses valores então para calcular os pesos de uma função de avaliação linear de 28 características por meio de regressão linear.

Em 1988, Lee e Mahajan utilizaram o programa jogador de Othello, Bill, para gerar material de treinamento através do autoconfronto. A variedade de situações foi garantida realizando as primeiras 20 jogadas aleatoriamente. As posições eram marcadas como vitória, empate e derrota dependendo do resultado do jogo. O treinamento servia para que o programa aprendiz pudesse estimar as chances de vitória de uma determinada posição. O experimento foi bem-sucedido uma vez que o programa aprendiz mostrou um bom desempenho contra o próprio Bill.

Em 1989, Tesauro e Sejnowski treinaram a primeira rede neural para função de avaliação (Gamão). As posições de treinamento foram retiradas de diversas fontes tais como livros-texto e jogos da internet. Enfatizaram a importância de ter exemplos ruins, além de bons exemplos (de outra forma o programa tenderia a considerar todos os movimentos como sendo bons). Também acrescentaram exemplos aleatórios de situações incomuns para garantir a exposição da rede a um vasto material de treinamento.

Como nas diversas áreas do aprendizado de máquina, os programas aprendizes de jogos de tabuleiro precisam gerenciar a super especialização. Um programa que classifica demasiadamente bem o conjunto de dados de treinamento acaba tendo seu desempenho ferido no conjunto de dados de teste. A idéia é que os conceitos possam ser aprendidos, mas o programa não deve simplesmente reproduzir os dados de treinamento. Por exemplo, em 2001, Dahl treinou uma rede neural para avaliar partes das posições de Go (os chamados campos receptivos). Para cada exemplo, foi utilizada uma jogada sugerida por um especialista e uma jogada criada aleatoriamente a partir de todos os movimentos legais.

O principal problema relacionado ao aprendizado supervisionado é que ele precisa de material de qualidade pré-existente para treinamento. Isso nem sempre é possível para todos os jogos. Jogos como o de Mancala, por exemplo, possuem poucas partidas publicamente disponíveis. A dependência de um especialista do jogo também pode ser um problema, uma vez que nem todos os pesquisadores têm fácil acesso a esse tipo de profissional, além de que o trabalho de criação de material torna-se manual e laborioso.

### 5.2.2 – Treinamento comparativo

Introduzido por Tesauro em 1989, o treinamento comparativo corresponde a uma das formas de produzir uma função de avaliação. Mesmo sendo uma forma de treinamento para aprendizado supervisionado, o princípio desse treinamento é fornecer menos informação do que o aprendizado supervisionado usual e mais informação do que o aprendizado não-supervisionado. O programa aprendiz, baseado em uma rede neural, não recebe valores exatos de avaliação para os possíveis movimentos, mas é informado de uma ordem relativa entre eles. Tipicamente, ele recebe exemplos de treinamento na forma de pares de movimentos com um sinal de treinamento indicando qual deles é preferível.

O programa aprendiz não deverá aprender uma relação de preferência entre movimentos, como no experimento de Utgoff e Heitman em 1988, mas sim tentar usar essa informação para ajustar parâmetros de uma função de avaliação. Este tipo de treinamento foi utilizado para ajustar parte dos pesos da função de avaliação do Deep Blue, mais especificamente com a parte de avaliação de segurança do rei. Isso acabou por fazer uma grande diferença em um dos jogos contra Garry Kasparov em 1997.

Esta arquitetura tem dois principais problemas: eficiência e consistência. Para calcular o melhor de N possíveis movimentos, o programa tem que comparar $(N^2-1)$ pares de movimentos. Além do mais, as preferências previstas precisam não ser transitivas: a rede poderia preferir o movimento *a* em relação ao *b*, *b* em relação ao *c* e *c* em relação ao *a*. Elas também podem não ser simétricas: a rede poderia preferir o movimento *a* em relação ao *b* e o *b* em relação ao *a*, dependendo da ordem em que os movimentos são fornecidos à rede[35].

Os problemas acima foram resolvidos forçando a arquitetura simétrica (os pesos utilizados para processar o primeiro tinham que ser equivalentes aos pesos utilizados para processar o segundo movimento) e a arquitetura separável (as duas sub-redes apenas compartilhavam a mesma unidade de saída). Isso significa que a mesma sub-rede é utilizada em ambas as metades da rede inteira, com a diferença que os pesos que conectam uma camada escondida de uma sub-rede à unidade de saída são multiplicados por –1 na outra sub-rede. Na prática, isso significa que apenas uma sub-rede precisa ser armazenada e que esta prevê uma avaliação absoluta para o movimento que está codificado como entrada. Conseqüentemente, apenas N avaliações precisam ser realizadas para determinar o melhor movimento [35].

Este tipo de treinamento se mostrou bem sucedido em diversos casos inclusive na sua contribuição para o programa jogador de Gamão, Neurogammon, que se tornou o primeiro programa jogador, baseado principalmente em tecnologia de aprendizado de máquina, a ganhar um campeonato (na Primeira Olimpíada de Computadores).

Conceitualmente, a informação de treinamento, na rede utilizada para treinar o Deep Blue, consistia em pares de movimentos. No entanto, o aprendizado era na verdade dado apenas pelo melhor movimento em coletânea de posições. Essa informação era então utilizada para minimizar a função de erro sobre todos os movimentos legais em cada posição. Maiores detalhes podem ser conhecidos em [80].

Diversos outros pesquisadores utilizaram técnicas similares. Por exemplo, em 1992, Schaeffer relatou a utilização dessas técnicas para ajuste de uma função de avaliação de jogo de Damas em 800 jogos de grandes mestres. Apesar de ter atingido bons resultados, os valores desse ajuste automático não se comparavam ao ajuste manual.

Apesar de ter apresentado bons resultados em algumas abordagens utilizadas, o treinamento comparativo tem alguns defeitos fundamentais. Um deles é que, em sendo uma forma de aprendizado supervisionado, ele tende a imitar o estilo humano de jogo não sendo capaz de surpreender seus adversários. Isso era considerado pelos pesquisadores do Chinook como algo de grande importância para vencer um grande mestre, uma vez que se o jogo tentasse seguir uma linha conhecida, acabaria, na melhor das hipóteses, conseguindo um empate. Em 1998, Buro atentou para um outro problema da técnica, as posições de treinamento não devem ser apenas representativas para posições encontradas durante o jogo, mas representativas para posições encontradas durante uma busca. Isso significa que mesmo posições que tendam para um dos lados e posições antiintuitivas deveriam constar no conjunto de treinamento. Outro defeito é considerar que movimentos de grandes mestres são os melhores, afinal podem existir outros movimentos melhores ou, ao menos, tão bons quanto o movimento realizado.

### 5.2.3 – Aprendizado por reforço

Imaginando uma máquina que recebe entradas $x_1$, $x_2$, $x_3$, ..., $x_n$. No aprendizado supervisionado, a máquina também recebe as saídas desejadas $y_1$, $y_2$, $y_3$, ..., $y_n$ e o seu objetivo é aprender a produzir as saídas corretas dada uma nova entrada $x_{n+1}$. No caso do aprendizado por reforço, a máquina não recebe as saídas como entrada. Na verdade, a máquina produz ações $a_1$, $a_2$, $a_3$, ..., $a_m$ que afetam o estado do mundo em questão (jogo), e recebe recompensas (ou punições) $r_1$, $r_2$, $r_3$, ...,$r_m$. O objetivo então é aprender a realizar ações de uma forma que maximize a soma das recompensas no longo prazo.

Imaginando um exemplo em que será necessário explicar a um cachorro o que ele deve fazer com relação à segurança de uma casa, pode ser criada uma série de situações. A cada uma dessas situações, o cachorro irá reagir de uma forma. Caso a forma tenha sido correta, ele recebe comida. Em caso contrário, ele pode receber algum tipo de punição. Aos poucos, o cachorro tende a executar as ações corretas para conseguir ganhar mais comida. Esse é o princípio do aprendizado por reforço.

Mesmo não sendo uma técnica de aprendizado não-supervisionado, esta forma de aprendizado em jogos não necessita intervenção humana para aplicar as recompensas e punições. Isso porque o reconhecimento do resultado ao final do jogo pode ser programado e esse próprio resultado pode ser utilizado como recompensa/punição. No caso do programa ter vencido o jogo, ele recebe a recompensa (sinal positivo) e no caso de ter perdido, a punição (sinal negativo). Esse resultado pode ser gradual de acordo com o nível de sucesso ou de fracasso. Em alguns jogos, como o Gamão, além da vitória, é considerado o resultado em termos de pontos obtidos.

O problema aqui é que, ao retornar um sinal positivo ou negativo no caso do jogo de Gamão, ele está recompensando ou punindo uma série de ações e não apenas uma (reforço atrasado). Seria preciso saber quais as ações mais responsáveis por uma derrota, uma vez que todas as jogadas poderiam ter sido perfeitas, exceto uma que levou à derrota. Esse problema é conhecido na literatura como "problema da atribuição de crédito temporal".

Quase todas as técnicas de aprendizado por reforço em uso são baseadas no método de diferenças temporais (TD) introduzido por Sutton [76] em 1988. Isso porque, em contraste com tentativas anteriores de implementar a idéia de reforço, TD fornece uma estrutura matemática consistente [63]. Trata-se de um algoritmo para aproximar o custo de

longo prazo esperado de um sistema dinâmico estocástico como uma função do estado atual.

Talvez o sucesso mais marcante do TD seja o programa jogador de Gamão, TD-Gammon [79]. Neste caso, o mapeamento dos estados para os custos futuros é implementado por um estimador de função parametrizada como uma rede neural. O TD-Gammon é uma rede neural que treina a si mesma, para ser uma função de avaliação para o jogo de Gamão, através de autoconfronto e do resultado final de cada jogo. Desenvolvido apenas com o propósito de explorar novas abordagens para problemas tradicionais em aprendizado por reforço, o programa acabou superando todos os programas jogadores anteriores.

O TD-Gammon foi projetado de forma a explorar as capacidades de redes neurais de múltiplas camadas treinadas pelo algoritmo TD(λ) [76] para aprender complexas funções não-lineares. Foi também projetado para fornecer uma detalhada comparação entre a abordagem TD e a abordagem supervisionada sobre um corpus de exemplos de especialistas.

Enquanto as buscas apresentavam bons resultados para jogos como Othello, Damas e Xadrez, para o jogo de Gamão, no entanto, o fator de ramificação da árvore de jogo é muito grande devido à questão probabilística dos dados. Mesmo em supercomputadores, não seria possível atingir grandes profundidades. Sem a opção de usar tabelas nem buscas, o programa tem que confiar em seu julgamento posicional heurístico.

As conexões da rede do TD-Gammon eram parametrizadas por pesos. Cada um dos nós na saída da rede correspondia a uma soma linear ponderada das entradas que alimentavam a rede, seguido por uma operação sigmoidal não-linear que mapeia a entrada somada total no intervalo da unidade. A não-linearidade da operação habilitava a rede neural a computar valores de funções não-lineares da entrada. Conforme teóricos [42] provaram, dadas suficientes unidades escondidas, a arquitetura MLP (Perceptron em múltiplas camadas) é capaz de aproximar qualquer função não-linear com exatidão arbitrária. A figura 12 mostra uma ilustração da MLP usada no TD-Gammon.

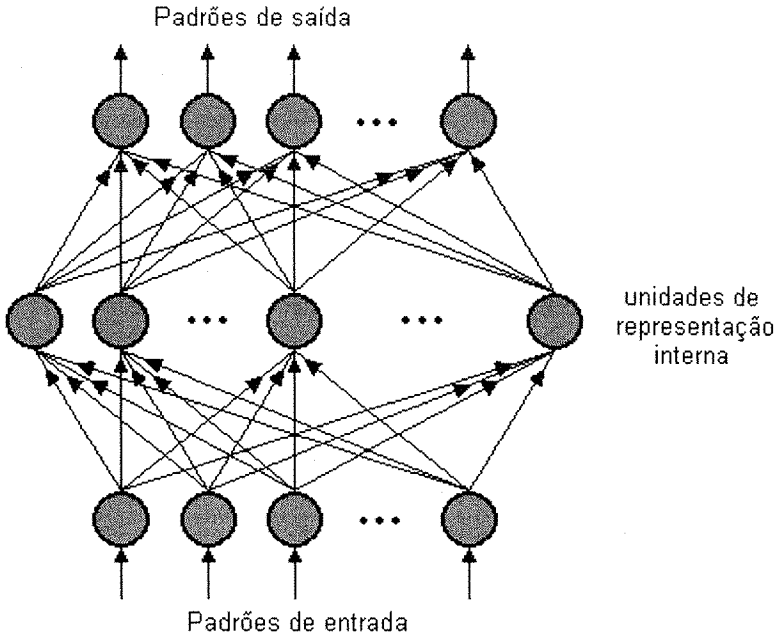



Figura 12 – Uma ilustração da arquitetura multicamadas usada no TD-Gammon [79]

Durante os primeiros milhares de jogos, as redes aprenderam táticas e estratégias elementares, tais como acertar o oponente, jogar de forma segura e produzir novos pontos. Conceitos mais sofisticados foram surgindo depois. Apesar disso, a descoberta mais encorajadora foi o bom comportamento escalar. À medida que a rede crescia e o treinamento aumentava, melhoras substanciais eram observadas no desempenho do programa. Após, alguns poucos milhões de jogos, o TD-Gammon não só estava ficando imbatível como estava alterando a maneira dos jogadores humanos jogarem mesmo as aberturas. A figura 13 mostra um exemplo de uma situação de tabuleiro em que o julgamento do programa é aparentemente superior ao de um especialista.

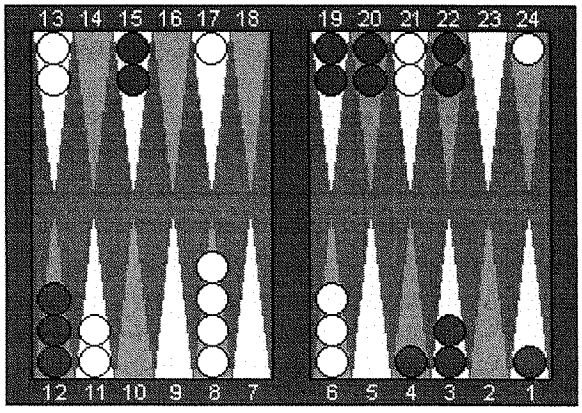



Figura 13 – Situação complexa em que o julgamento do TD-Gammon é superior

Na situação acima, as brancas irão jogar e tiram 4 e 4 nos dados (por ser número repetido, jogam duas vezes cada). O movimento óbvio de um jogador humano é 8-4*, 8-4, 11-7, 11-7 (o asterisco denota que uma peça do oponente foi atingida). O movimento do

TD-Gammon, no entanto, é 8-4*, 8-4, 21-17, 21-17. Pela análise realizada pela função de avaliação do programa, a jogada realizada pelo humano é boa, porém inferior em termos de resultado à jogada alternativa.

Outros pesquisadores tentaram reproduzir a técnica para outros jogos sem o mesmo sucesso. Em um artigo escrito em 1997, Pollack [62] comenta as razões pelo qual o TD-Gammon é tão bem sucedido. Eles chegaram à conclusão que a causa do sucesso estaria relacionada à dinâmica do Gamão combinada com o poder do aprendizado co-evolucionário. De fato, no jogo de Gamão, as transições de um estado para o outro são markovianas, isto é, dependem apenas do estado atual. Para processos não-markovianos, o método TD sugerido se torna pouco prático devido ao tamanho e à complexidade do espaço de entrada. A outra vantagem é que o jogo de Gamão é altamente estocástico garantindo assim uma larga variedade de jogos diferentes (se não fosse assim, o autoconfronto começaria a repetir caminhos iguais).

## 5.3 – Aprendizado de padrões

O confronto de Kasparov contra o Deep Blue exibe um intrigante fato. O Deep Blue era capaz de avaliar bilhões de posições a partir de cada posição de tabuleiro, enquanto Kasparov não conseguia avaliar mais do que algumas dúzias delas. Apesar disso, Kasparov estava praticamente no mesmo nível do Deep Blue em termos de qualidade de jogo. Este fato indica que um entendimento mais profundo pode diminuir drasticamente a quantidade de cálculos necessária. A experiência adquirida com o passar dos anos de estudo e prática fez o campeão ver pistas, em padrões do tabuleiro de Xadrez, que informavam a ele quais movimentos deveriam ser examinados.

A análise acima aponta para o próximo passo lógico: um programa com habilidades para reconhecimentos de padrões importantes e para generalização, significaria juntar a visão posicional de Kasparov ao poder computacional do DeepBlue e criar um programa tão forte que faria os programas atuais de força bruta parecerem iniciantes.

O grande problema parece estar em formalizar conceitos que para os humanos são simples, mas para as máquinas representam padrões não-triviais. Por exemplo, a forquilha (ataque a duas ou mais peças adversárias simultaneamente) no Xadrez é um conceito extremamente simples para o entendimento humano, mas de difícil formalização. Obviamente, se você não consegue descrever um conceito em uma dada linguagem de hipóteses, então você não consegue aprendê-lo. Apesar disso, várias tentativas foram realizadas de se criar programas jogadores baseados principalmente no conhecimento de padrões.

Uma das técnicas utilizadas que requer a menor iniciativa pelo aprendiz é o aprendizado por tomada de conselho. Nesta arquitetura, o usuário é capaz de comunicar conceitos abstratos e objetivos ao programa. Um exemplo clássico é o de Carlson em 1973 no qual um professor de Xadrez fornecia ao programa uma biblioteca de padrões utilizando uma linguagem de programação de Xadrez. A idéia de ensinar, nesse tipo de caso, é como programar em uma linguagem de alto nível. Para que o professor possa se comunicar com o programa de uma forma intuitiva é preciso que o programa dedique um esforço considerável para compilar os conselhos na sua própria linguagem de padrões. Uma das grandes dificuldades dessa técnica é combinar os pedaços de conselhos que podem ser aplicáveis a uma determinada situação de jogo. Em 2000, Fürnkranz, Pfahringer, Kaindl e

Kramer deram um primeiro passo identificando dois diferentes tipos de conselho, conselho de abstração de estado (conselho que mostra ao sistema características importantes do jogo) e conselho de seleção de movimentos (conselho que sugere potenciais bons movimentos).

Uma outra abordagem sugerida, entre muitas existentes para esta técnica, é a "indução estruturada" cuja idéia básica é induzir padrões que não são apenas confiáveis, mas compreensíveis o suficiente para permitirem a um especialista em determinado jogo interagir com o algoritmo aprendiz, focando o mesmo em porções relevantes do espaço de busca, definindo novos padrões que quebram o problema em sub-problemas menores e utilizar indução automática para cada um deles. Essa não é a única forma de realizar a indução de padrões. Na verdade, o princípio de indução de padrões reside no fato de avaliar posições muito conhecidas como posições de fim de jogo. Em já se sabendo o resultado daquela posição, ela se torna uma boa forma de avaliar um algoritmo de aprendizado de máquina. Muitas pesquisas foram realizadas no sentido de converter alguns bancos de dados de fim de jogo em um conjunto de regras que fosse capaz de prever os seus valores.

Outro estudo que tem influenciado os experimentos de aprendizado de padrões em jogos é o estudo de modelos cognitivos. Alguns estudos psicológicos, por exemplo, têm mostrado que as diferenças entre um novato em Xadrez e um especialista não são muitas com relação a calcular uma seqüência de movimentos completos, mas a maior diferença está em quais movimentos são escolhidos para iniciar o cálculo. Para essa pré-escolha, os especialistas fazem uso dos padrões. Em 1973, Simon e Gilmartin estimaram o número de padrões considerados por um especialista em Xadrez na ordem de 10.000 a 100.000. Do outro lado, a maior parte dos programas campeões de Xadrez não considera mais do que algumas centenas deles. Algumas áreas de estudo são devotadas exclusivamente ao aprendizado humano de habilidades de jogo. As pesquisas atuais na área de aprendizado humano em jogos podem ser classificadas em três grandes categorias: estudo de percepção, memória ou solução de problemas; estudos acompanhando o desenvolvimento de um novato a especialista; e modelagem de máquina [37]. Em 2001, Gobet e Simon discutiram a influência da psicologia sobre o aprendizado de máquina e do aprendizado de máquina sobre a psicologia. Muitos pesquisadores têm desenvolvido programas que utilizam bancos de dados de padrões e simulam aspectos relativos ao processo de resolução de problemas, tais como percepção e recuperação. Por exemplo, em 1995, Kerner descreveu um programa baseado nos padrões explicativos [71] introduzidos por Schank. O conhecimento básico consiste em uma hierarquia de conceitos estratégicos de Xadrez, cada um indexado por um padrão explicativo. Ao encontrar uma nova posição, o sistema recupera o padrão explicativo e reage através de operações de inserção, remoção, substituição, especialização e generalização. O padrão resultante será armazenado caso seja aprovado.

Existem inúmeros outros programas jogadores baseados em padrões que não são explicitamente baseados em resultados da ciência cognitiva. Existem alguns programas, por exemplo, cuja entidade de representação básica são pares de pesos de padrões. Os padrões são representados por grafos conceituais, onde os arcos são relacionamentos de ataque e os nós são as peças e espaços importantes do tabuleiro. O aprendizado de padrões, nesse caso, pode ser generalizado extraindo a sub-árvore comum de dois padrões com pesos similares ou especializando adicionando nós e arcos ao grafo. Diferentes padrões podem ser combinados usando engenharia reversa. Os pesos dos padrões são ajustados utilizando uma variante do aprendizado por diferença temporal, onde cada padrão possui uma taxa de aprendizado própria, configurada pelo "recozimento simulado" (quanto mais freqüente um padrão é atualizado, menor se torna a sua taxa de aprendizado). Um movimento é avaliado

combinando os pesos de todos os padrões encontrados em um único valor. O movimento escolhido é aquele cujo valor é maior. O MORPH, programa jogador de Xadrez, utiliza o sistema descrito e apresentou resultados razoáveis em testes contra outros programas conhecidos baseados em busca.

Outro grande tópico de estudo em jogos é o de aprendizado baseado em explicação (EBL). O princípio básico consiste no fato de a teoria de domínio poder ser utilizada para encontrar explicação para um determinado movimento. Generalizar esta explicação significa encontrar um padrão que será associado ao movimento jogado. O programa irá então jogar este tipo de movimento nas situações em que o padrão for encontrado. Diversos programas foram construídos utilizando a abordagem EBL. É o caso do programa jogador de Go, Gogol, que é um programa baseado em regras. Ele utiliza milhares de regras para saber se um determinado objetivo tático pode ser atingido. Ele se vale de uma teoria de jogo simplificada para representar o grau de obtenção de objetivos. A aquisição automática de regras táticas segue três passos: geração de padrões, avaliação de jogo e generalização. Maiores informações sobre a aquisição automática de regras táticas de Go podem ser encontradas em [24].

## 5.4 - Aprendizado de estratégias

Definindo uma estratégia como sendo um procedimento simples e interpretável para jogar um jogo, o aprendizado de tal estratégia parece ser consideravelmente mais difícil que o aprendizado de classificação de posições. Para uma posição final de um jogo de Xadrez pode ser fácil classificar o jogo como uma situação de vitória se houver alguma vantagem para o programa. No entanto, trocar essa posição por outra posição classificada como vitória, não significa evoluir para dar o xeque-mate. O problema de ter que escolher entre posições que parecem igualmente boas foi denominado efeito escarpa [54].

Em 1995, Bain [5] tentou resolver esse problema com um jogador ótimo aprendendo a partir de um banco de dados de fim de jogo de Xadrez. Ele tentou aprender regras que previssem o número de passos para a vitória, considerando jogo ótimo para ambos os lados. A idéia era prever o número de passos para o final em todos os movimentos possíveis. Seria então escolhido o movimento com o menor número de passos. Mesmo com os conceitos geométricos utilizados como suporte de conhecimento, o aprendiz tinha dificuldade para prever os valores corretos do número de passos com precisão.

Em 1997, Morales [55] utilizou técnicas de programação de lógica indutiva para induzir padrões complexos de lógica de primeira-ordem. Era necessário um professor humano para prover alguns exemplos de treinamento. Este algoritmo utilizava a descrição atual do conceito e gerava novas posições alterando certos aspectos como a localização das peças e o lado a movimentar. Foi realizada uma demonstração da flexibilidade do programa aprendendo com uma coletânea de fim de jogo de Xadrez.

Outros trabalhos foram realizados no aprendizado de estratégias e vários deles no domínio do Xadrez como os exemplos acima. Isso porque o Xadrez possui uma grande quantidade de bancos de dados de fim de jogo disponíveis e outros exemplos de treinamento. E acima de tudo, porque os humanos não conseguem aplicar uma estratégia ótima para alguns desses finais de jogo, por não serem triviais.

## 5.5 – Modelagem de oponente

A modelagem de oponente é uma importante área do estudo de aprendizado de jogos de tabuleiro. Surpreendentemente, contudo, esta área não recebeu muita atenção na comunidade de pesquisadores de jogos [35]. Billings [11] acredita que a razão para tal seja o fato que, em jogos como Xadrez, a modelagem de oponente não é um fator crítico para se atingir um alto desempenho (para um jogo de Poker, isso já seria essencial para o sucesso).

Muitos autores, no entanto, já comentaram que a busca Minimax considera sempre o jogo ótimo para ambos os jogadores, enquanto poderia ser mais recompensador tentar maximizar o resultado para um jogador imperfeito. Por exemplo, no jogo de Gamão, uma vitória pode ser melhor que a outra em pontos. Isso significa que modelando corretamente o oponente, é possível seguir um caminho que provavelmente levará a uma vitória com maior pontuação. O mesmo serve para o jogo de Damas, mesmo não existindo diferença de uma vitória para outra, pois o programa pode acreditar que se encontra em uma situação de empate e não tentar explorar caminhos que possam levar o seu adversário ao erro.

Alguns trabalhos foram realizados considerando também a utilização de valores sub-ótimos do Minimax para forçar o programa a criar situações diferentes de jogo evitando seguir uma linha de jogadas já previamente conhecidas pelo seu adversário. Jansen [88,89] investigou heurísticas que ocasionalmente sugeriam movimentos sub-ótimos com potencial de criar uma armadilha para o lado mais fraco no caso do fim de jogo de Xadrez RrRT (Rei e rainha contra Rei e Torre). Outros pesquisadores sugeriram incorporar esse modelo de oponente na busca do programa [23].

O provável primeiro estudo a tentar determinar automaticamente o modelo de um oponente através do progresso da seleção de seus movimentos foi realizado por Carmel e Markovitch [22] em 1993. Inicialmente, eles propuseram um algoritmo simples para tentar estimar a profundidade da busca do adversário. Para este fim, eles empregaram uma técnica de treinamento comparativo. Para cada posição jogada pelo adversário, o programa realizava busca em diferentes profundidades e atribuía uma pontuação para cada profundidade através da comparação dos resultados da busca com o movimento jogado. A profundidade que atingisse o maior número de pontos era considerada como sendo a profundidade de busca do adversário. Logo após, eles propuseram uma maneira de aprender a função de avaliação do oponente através de aprendizado supervisionado. Utilizando sugestões de movimentos, o programa tentava encontrar os pesos da função de avaliação utilizada. Também eram considerados vários níveis de profundidade de busca. Os autores testaram essa abordagem em um jogador de Damas e alcançaram exatidão em duas a cada três posições. Conhecer a função de avaliação do adversário poderia significar avaliar os nós MIN do Minimax com essa função, enquanto os nós MAX seriam avaliados com a própria função.

Walczak e Dankel II [88] introduziram o "modelador de adversário indutivo" (IAM – Inductive adversary modeler), que assumia que humanos realizavam esforços por posições que eram cognitivamente simples. O modelador inicialmente aprendia padrões a partir de uma coleção de jogos do adversário. Para prever os movimentos do jogador, ele considerava quais padrões poderiam ser completados e optava pelo movimento que completava o maior número de padrões. No jogo de Xadrez, considerando a existência de uma média de 35 movimentos por jogada, um algoritmo aleatório conseguiria prever

apenas 2,8% dos movimentos de um determinado jogador. O IAM conseguiu prever 10% dos movimentos realizados por grandes mestres de Xadrez incluindo Karpov e Kasparov.

Conforme comentado, poucos trabalhos foram produzidos para se criar modeladores de oponente em jogos. A maioria deles foi realizada para jogos de informação imperfeita, onde os ganhos de desempenho são mais significativos que nos jogos de informação perfeita. Dentre os jogos listados aqui, somente o jogo de Scrabble é de informação imperfeita. Mesmo assim, o programa jogador de Scrabble, Marven, derrotou o campeão mundial do jogo em 1998 sem a utilização de nenhuma técnica de modelagem de oponente. Mais material sobre modelagem de oponente pode ser encontrado em estudos de jogos de cartas.

# Capítulo 6

# Considerações finais

Este capítulo não poderia começar sem antes pedir sinceras desculpas a todos os pesquisadores de inteligência artificial para jogos de tabuleiro cujo trabalho não foi descrito nessa dissertação devido às restrições de escopo, objetividade, tempo e/ou espaço. Naturalmente, esse texto acaba por privilegiar alguns pesquisadores responsáveis por programas campeões ou que tiveram uma maior visibilidade por outras razões, mesmo sabendo que seus projetos não são nem os únicos nem necessariamente os melhores.

Por um lado, a grande quantidade de projetos existentes impediu a presença de todos eles neste texto. Por outro, essa área de jogos possui muitos projetos interessantes, mas nem sempre acessíveis ao autor. Isso porque existe, como na área financeira e em outras, muitos interesses comerciais envolvidos. Dessa forma, nem sempre as técnicas mais recentemente criadas já estão publicadas. E quando estão publicadas, nem sempre estão acessíveis sem maiores custos. Essa acabou por ser uma das dificuldades desse trabalho, mas que acabou sendo contornada através da leitura de publicações semelhantes ou que faziam referências a essas outras.

No entender do autor desta dissertação, este trabalho poderá servir para outros pesquisadores que queiram desenvolver trabalhos mais específicos relacionados à inteligência artificial de jogos de tabuleiro, tendo neste trabalho um ponto de apoio para conhecer um pouco do histórico e dos avanços já realizados e compreender as técnicas utilizadas em cada momento de um jogo. A partir deste texto, é possível até mesmo partir na direção de se criar um programa jogador, mesmo que o jogo em questão não tenha sido relacionado aqui, uma vez que algumas das técnicas comentadas também podem ser aplicadas a outros tipos de jogos como os de cartas, por exemplo.

Ainda existe bastante espaço para novos estudos e avanços na área de inteligência artificial para jogos de tabuleiro. A conferência bienal "Computer and Games" a ser realizada de 5 a 7 de julho de 2004 possui diversos tópicos de interesse relacionados à área aqui estudada. Entre esses tópicos estão: o estado atual dos programas jogadores; novos desenvolvimentos teóricos em áreas correlatas; técnicas de IA aplicadas a jogos incluindo aprendizado, busca heurística e representação do conhecimento; e pesquisas cognitivas sobre como os humanos jogam.

Para futuros trabalhos, na visão deste autor, a área de aberturas e de fim de jogo parecem razoavelmente estagnadas. Apesar de que muitos trabalhos de fim de jogo ainda estão sendo publicados nos últimos anos relacionados a implementações em jogos não tão populares, como Xadrez chinês, por exemplo.

A área de meio de jogo, apesar de já estar em um estágio bastante avançado como no caso dos algoritmos de busca que já se encontram bastante otimizados, ainda apresenta um considerável espaço para pesquisas. Um dos pontos em que há espaço é na busca realizada para jogadores imperfeitos. Outro ponto é na realização de experimentos com algoritmos que possam fazer comparações nos inúmeros jogos existentes e comprovar o sucesso de técnicas publicadas.

A área de aprendizado, por sua vez, apesar de alguns avanços interessantes, parece estar ainda no seu estágio inicial. Mesmo sendo um assunto largamente estudado atualmente, ainda existem muitas sub-áreas pouco desenvolvidas. Um programa jogador de Go campeão talvez tenha que esperar o amadurecimento da área, uma vez que as técnicas de busca junto à capacidade de processamento atual pouco podem fazer para vencer os campeões humanos.

Outras áreas de estudo, fora do escopo desta dissertação, também podem utilizar parcialmente as técnicas aqui relacionadas como o estudo de jogos de mais de 2 jogadores, que precisaria estar associado a técnicas de cooperação.

Apesar de não ter encontrado nenhuma publicação neste sentido, a construção de programas que possam ensinar jogadores humanos a jogar um determinado jogo pode ser uma área interessante. O programa teria que traduzir o seu conhecimento de buscas e aprendizado próprio de uma forma compreensível ao intelecto humano. Esse ainda parece um estudo consideravelmente distante da realidade atual.

# Bibliografia


[1] Akl, S.G. e Doran, R.J. (1983). A comparison of parallel implementations of the alpha-beta and Scout tree search algorithms using the game of checkers. *Computer Game-Playing – Theory and Practice*, p.290-303, Ellis Horwood.

[2] Akl, S.G. e Newborn M. (1977). The principal continuation and the killer heuristic. Em *proceedings of the 5th annual ACM Computer Science Conference*, Seattle, WA, p.466-473. ACM Press.

[3] Allis, L.V. (1994). Searching for solutions in games and Artificial intelligence. Tese de PhD, Universidade de Limburg, Holanda.

[4] Althöfer, I. (1990). Generalized minimax algorithms are no better error correctors than minimax itself. Em *Advances in Computer Chess 5*, p.265-282

[5] Bain, M. e Srinivasan, A. (1995). Inductive logic programming with large-scale unstructured data. Em *Machine Intelligence 14*, p.233-267, Oxford University Press.

[6] Baum, E.B. e Smith W.D. (1995). Best Play for Imperfect Players and Game Tree Search; Part I – Theory. Artigo técnico, NEC Research Institute, Princeton, NJ.

[7] Baxter, J. e Tridgell, A. e Weaver, L. (2000). Learning to play chess using temporal differences. *Machine Learning 40*(3), 243-263.

[8] Beal, D.F. (1980). An analysis of minimax. Em *Advances in Computer Chess 2*, p.103-109.

[9] Berliner, H.J. (1979). The B* Tree Search Algorithm: A best-first proof procedure. *Artificial Intelligence 12*, p.23-40.

[10] Berliner, H.J. e McConnell C. (1996). B* probability based search. *Artificial Intelligence* 86, p.97-156.

[11] Billings, D. e Papp D. e Schaeffer, J. e Szafron, D. (1998). Opponent modeling in poker. Em *proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, pp.493-498. AAAI Press.

[12] Bratko, I. e Gams, M. (1982). Error analysis of the minimax principle. Em *Advances in Computer Chess 3*, p.1-15.

[13] Brautaset, S. (2003). Generalised Game-Tree Search,. Trabalho de graduação, University of Westminster, Londres, Inglaterra.

[14] Brockington, M.G. (1996). A Taxonomy of parallel game-tree search algorithms. *ICCA Journal*, 19(3),p.162-174.

[15] Brockington, M. G. (1998). Asynchronous Parallel Game-Tree Search. Tese de PhD, Universidade de Alberta, Edmonton, Canadá.

[16] Brudno, A.L. (1963). Bounds and valuations for abridging the search for estimates. *Problems of Cybernetics*, vol.10, p.225-241. Tradução do original russo *Problemy Kibernetiki 10*, p.141-150.

[17] Brünger, A. e Marzetta, A. e Fukuda, K. e Nievergelt, J. (1997). The ZRAM parallel search bench and its applications. Annals of Operation Research.

[18] Bryant, M. (1990). Personal communication to Paul Lu, London.

[19] Buro, M. (1999). Toward Opening Book Learning. *International Computer Chess Asociation 22*(2), 98-102.

[20] Campbell, M.S. (1981). Algorithms for the parallel search of game trees. Dissertação de Mestrado, Universidade de Alberta, Edmonton, Canadá.

[21] Campbell, M.S. e Marsland, T.A. (1983). A comparison of minimax tree search algorithms. *Artificial Intelligence 20*, p.347-367.
[22] Carmel, D. e Markovitch, S. (1993). Learning models of opponent's strategy in game playing. Número FS-93-02, Menlo Park, CA, p.140-147. The AAAI Press.
[23] Carmel, D. e Markovitch, S. (1998). Model-based learning of interaction strategies in multiagent systems. *Journal of Experimental and Theoretical Artificial Intelligence 10*(3), 309-332.
[24] Cazenave, T. (1996). Automatic Acquisition of tactical Go rules. Em *Proceedings of the Game Programming Workshop – 96*, Hakone, Japão.
[25] Cazenave, T. (2000). Abstract Proof Search. *Proceedings of the 2nd International Conference on Computer and Games (CG)*, p.39-54, Springer-Verlag, LNCS 2063.
[26] Cazenave, T. (2002). Gradual Abstract Proof Search. *International Computer Games Association Journal, 25*(1), p.3-15.
[27] Chellapilla, K. (2001). Evolving an Expert Checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation 5*(4), p. 422-428.
[28] Dahl, F.A. (2001). Honte, a Go-Playing program using Neural Nets. Em *Machines that Learn to Play Games*, capítulo 10, p.205-223, Nova Science Publishers, Huntigton, NY.
[29] Diderich, C.G. (1992). Evaluation des performances de l'Algorithme SSS* avec phases de synchronisation sur une machine parallèle à mémoires distribuées. Artigo técnico LITH-99, Swiss Federal Institue of Technology, Lausanne, Suíça.
[30] Donninger, C. (1996). A graphical language for expressing chess knowledge. *International Computer Chess Association Journal 19*(4), 234-241.
[31] Epstein, S.L. (2001). Learning to play expertly: a tutorial on hoyle. Em *Machines that learn to play games*, capítulo 8, p.153-178. Huntington, NY: Nova Science Publishers.
[32] ESA – Entertainment Software Association. Disponível online em http://www.theesa.com/members.html.
[33] Fang, H. e Hsu, T. e Hsu, S. (2000). Construction of Chinese Chess Endgame Databases by Retrograde Analysis. *Proceedings of the 2nd International Conference on Computer and Games (CG)*, p.96-114, Springer-Verlag, LNCS 2063.
[34] Fraenkel, A. S. (1996). Combinatorial Games: Selected bibliography with a succinct gourmet introduction. *Games of No Chance*, p.493-537, Cambridge U. Press, NY.
[35] Fürnkranz, J. e Kubat, M. (2001). Machine Learning in Games: A survey. Em *Machines that Learn to Play Games*, capítulo 2, p.11-59, Nova Science Publishers, Huntigton, NY.
[36] Gasser, R. (1991). Applying Retrograde Analysis to nine men's morris. *Heuristic Programming in Artificial Intelligence 2*, p.161-173.
[37] Gobet, F. e Simon, H. A. (2001). Human learning in game playing. Em *Machines that Learn to Play Games*, capítulo 3, p.61-80, Nova Science Publishers, Huntington, NY.
[38] Goot, R. (2000). Awari retrograde analysis. *Proceedings of the 2nd International Conference on Computer and Games (CG)*, p.87-95, Springer-Verlag, LNCS 2063.
[39] Greenblatt, R.D. e Eastlake, D.E. e Crocker, S.D. (1967). The greenblatt Chess program. Em Proceedings of the Fall Joint Computer Conference, vol.31, p.801-810.
[40] Greer, K.R.C. e Ojha, P.C. e Bell, D.A. (1999). A pattern-oriented approach to move ordering: the chessmaps heuristic. *International Computer Chess Association Journal 22*, 13-21.

[41] Hamilton, S. e Garber, L. (1997). Deep Blue's Hardware and Software Synergy, *IEEE Computer 30*, p.29-35.
[42] Hornik, K. e Stinchcombe,M. e White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks 2*, p.359-366.
[43] Huebsch, R. (2001). Utilizing Parallel Computing to Play a Better Game of Chess. Disponível online em http://www.cs.berkeley.edu/~huebsch/cs267/chess.html.
[44] Jansen, P.J. (1992). KQKR: Awareness of a fallible opponent. *International Computer Chess Association Journal 15*(3), 111-131.
[45] Jansen, P.J. (1993). KQKR: Speculatively thwarting a human opponent. *International Computer Chess Association Journal 16*(1), 3-17.
[46] Junghanns, A. e Schaeffer, J. (1997). Search Versus Knowledge in Game-Playing Programs Revisited. Em *Proceedings of International Joint Conferences on Artificial Intelligence - 97*, p.692-697, Nagya, Japão.
[47] Knuth, D.E. e Moore, R.W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence 6*(3), p.293-326.
[48] Lake, R. e Schaeffer, J. e Lu, P. (1994). Solving large retrograde analysis problems using a network of workstations. Em *advances in Computer Chess 7*, p.135-162, Universidade de Limburg, Holanda.
[49] Lake, R. e Schaeffer J. e Treloar, N. (1992). The 3B1b3W Endgame. *Checkers*, p.32-38.
[50] Leifker, D.B. e Kanal L.N. (1985). A Hybrid SSS*/Alpha-beta algorithm for Parallel Search of Game Trees. Em *Proceedings of International Joint Conferences on Artificial Intelligence - 85*, p.1044-1046.
[51] Lincke, T.R. (2000). Strategies for the automatic construction of opening books. *Proceedings of the 2nd International Conference on Computer and Games (CG)*, p.74-86, Springer-Verlag, LNCS 2063.
[52] Lorenz, U. e Rottman V. (1995). Controlled Conspiracy Number Search. *International Computer Chess Association Journal 18*(3), p.135-147.
[53] Michie, D. (1983). Game-playing programs and the conceptual interface. *Computer Game-Playing – Theory and Practice*, p.11-25, Ellis Horwood.
[54] Minsky, M. (1963). Steps towards artificial intelligence. Em *Computers and Thought*, McGraw-Hill, NY.
[55] Morales, E. (1997). PAL – A pattern-based first-order inductive system. *Machine Learning 26*(2-3), p.227-252. Edição especial sobre programação lógica indutiva.
[56] Nareyek, A. (2001). Constraint-Based Agents - An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds. Springer-Verlag, LNAI 2062.
[57] Palay A.J. (1982). The B* tree search algorithm – new results. *Artificial Intelligence 19*, p.145-163.
[58] Pearl, J. (1980). Asymptotic properties of minimax trees and game-searching procedures. *Artificial Intelligence 14*, p.113-138.
[59] Pearl, J. (1984). Heuristics: Intelligent search strategies for computer problem solving. Addison-Wesley Longman publishing Co., Boston, MA.
[60] Pinhanez, C.S. (1989). Algoritmos paralelos para busca em árvores de jogo. Dissertação de Mestrado, departamento de ciência da computação, Instituto de Matemática e Estatística - Universidade de São Paulo, São Paulo, Brasil.

[61] Plaat, A. (1996). Research Re: Search & Re-search. Tese de PhD, Universidade Carnegie Melon, Pittsburgh, PA.
[62] Pollack, J. (1997). Why did TD-Gammon work?. *Advances in Neural Information Processing Systems 9*, p.10-16.
[63] Ribeiro, C.H. (1999). A tutorial on reinforcement learning techniques. Em *International Joint Conference on Neural Networks*.
[64] Rijswijck, J. (2000). Learning from perfection – A Data Mining approach to evaluation function learning in Awari. *Proceedings of the 2nd International Conference on Computer and Games (CG)*, p.74-86, Springer-Verlag, LNCS 2063.
[65] Russel, S. e Norvig, P. (2002). Artificial Intelligence – A Modern Approach (2nd edition). Prentice Hall, NJ.
[66] Sadikov, A. e Bratko, I. e Kononenko, I. (2003). Search versus Knowledge: An Empirical Study of Minimax on KRK. Em *Proceedings of Advances in Computer Games 10*, p.33-44, Graz, Áustria.
[67] Schaeffer, J (1983). The history heuristic. *International Computer Chess Association Journal 6*(3), 16-19.
[68] Schaeffer, J. (1989). The history heuristic and alpha-beta search enhancements in practice. IEEE Transactions on Patterns Analysis and Machine Intelligence, PAMI-11(11), p.1203-1212.
[69] Schaeffer, J. e Culberson, J.e Treloar, N. e Knight, B. e Lu, P. e Szafron, D. (1991). Reviving the game of checkers. *Heuristic Programming in Artificial Intelligence; The Second Computer Olympiad*, Ellis Horwood, Londres, p.119-136.
[70] Schaeffer, J. e Culberson, J. e Treloar, N. e Knight, B. e Lu, P. e Szafron, D. (1992). A World Championship Caliber Checkers Program. *Artificial Intelligence 53*(2-3), p.273-290.
[71] Schank, R. C. (1986). Explanation patterns: Understanding mechanically and creatively. Lawrence Erlbaum Associates, Hillsdale, NJ.
[72] Schrüfer, G. (1986). Presence and absence of pathology on game trees. Em *Advances in Computer Chess 4*, p.101-112.
[73] Shinghal, R. e Shved, S. (1991). Proposed modifications to parallel State Space Search of game trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 5(5), p.809-833.
[74] Steinberg, I.R. e Solomon, M. (1990). Searching Game Trees in parallel. Em *Proceedings of the 1990 International Conference on Parallel Processing*, vol.3, p.9-17. Penn. State University Press.
[75] Stockman, G.C. (1979). A minimax algorithm better than alpha-beta?. *Artificial Intelligence 12*, p.179-196.
[76] Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning 3*, p.9-44.
[77] Sutton, R.S. e Barto A.G. (1998). Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.
[78] Sutton-Smith, B. (2001). The ambiguity of play. Harvard University Press, Cambridge, MA.
[79] Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM 38*(3), p.58-68.

[80] Tesauro, G. (2001). Comparison Training of Chess evaluation functions. Em *Machines that Learn to Play Games*, capítulo 6, p.117-130, Nova Science Publishers, Huntigton, NY.
[81] The Game AI Page. Disponível online em http://www.gameai.com/.
[82] The University of Alberta GAMES Group. Disponível online em http://web.cs.ualberta.ca/~games/index.html.
[83] Thomsen, T. (2000). Lambda-search in game trees – with application to Go. *International Computer Games Association Journal, 23*(4),p.203-217.
[84] Treglia, D. (2002). Game Programming Gems 3. Charles River Media, Hingham, MA.
[85] Tschöke, S. e Polzer, T. (1996). Portable Branch-and-Bound Library (PPBB-lib) User Manual, versão 2.0. Disponível online em http://www.uni-paderborn.de/fachbereich/AG/monien/SOFTWARE/PPBB/ppbblib.html.
[86] Upton, R.J. (1999). Dynamic Stochastic Control - A New Approach to Game Tree Searching. Tese de PhD, Universidade de Warwick, Warwick, Inglaterra.
[87] Utgoff, P.E. (2001). Feature construction for game playing. Em *Machines that Learn to Play Games*, capítulo 7, p.131-152, Nova Science Publishers, Huntigton, NY.
[88] Walczak, S. e Dankel II, D.D. (1993). Acquiring tactical and strategic knowledge with a generalized method for chunking of game pieces. *International Journal of Intelligent Systems 8*(2),249-270.

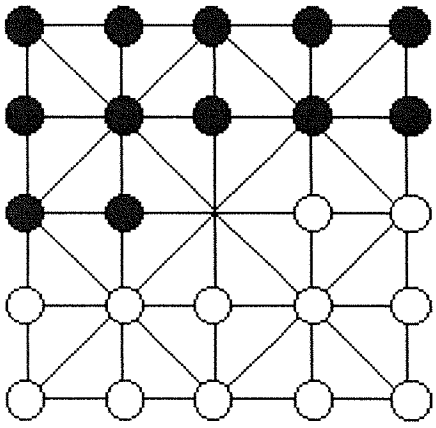
# Apêndice A

# Descrição dos jogos de tabuleiro

Neste apêndice, estão ilustrados e descritos os jogos citados nessa dissertação, em ordem alfabética, para facilitar a compreensão de determinados comentários dependentes de jogo.

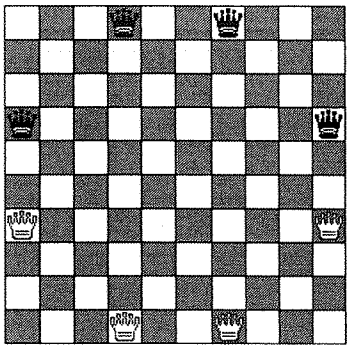
A Descrição das regras foi realizada de forma sucinta e não contempla todos os detalhes envolvidos nos jogos. Para alguns jogos, foram também citados programas famosos ou atuais campeões do jogo (que foram campeões de algum campeonato recente do jogo ou campeões das últimas olimpíadas de computadores).

## A.1 – Alquerque



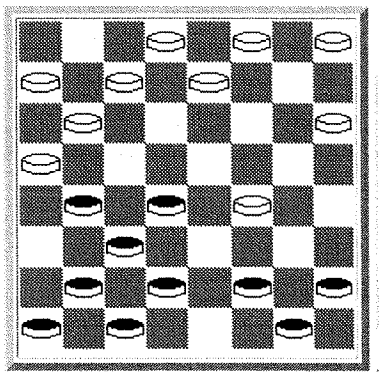
O Jogo de Alquerque é considerado parente do jogo de Damas devido a algumas semelhanças na regras. Acredita-se que o jogo foi inventado em 1.400 A.C., mas só foram encontrados registros das suas regras no século XIII. O jogo começa como na figura acima com 12 peças pretas contra 12 brancas. Uma peça pode se mover de um ponto para outro adjacente que esteja vazio ou capturar uma peça do adversário que esteja adjacente e cujo ponto seguinte esteja vazio. As múltiplas capturas são obrigatórias. Vence quem capturar todas as peças do adversário.

## A.2 – Amazons



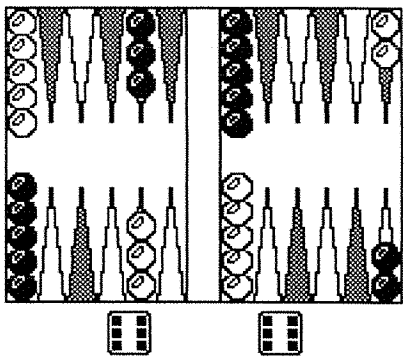
O jogo de Amazons foi criado em 1988. Ele é jogado em um tabuleiro 10x10. Cada jogador tem 4 peças. Uma jogada consiste em mover uma peça e atirar uma flecha. A peça se move como a rainha do Xadrez (por múltiplas casas vazias ortogonalmente ou diagonalmente), exceto pelo fato que ela não pode capturar. Ao parar em uma casa, uma flecha deve ser arremessada. A flecha se move da mesma forma que as peças. A casa em que a flecha para é interditada para o resto do jogo (nem as peças nem as flechas podem passar). O objetivo é deixar o outro jogador sem movimentos. Não existe empate nesse jogo, pois quem não conseguir movimentar as suas peças primeiro, perde. O atual melhor programa jogador de Amazons é o Amazong, vencedor da Oitava Olimpíada de Computadores realizada em novembro de 2003, em Graz, na Áustria.

## A.3 – Damas



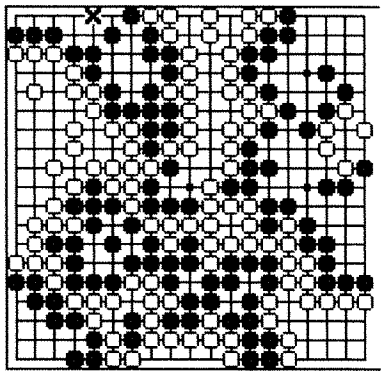
O Jogo de Damas foi provavelmente inventado no sul da França há 3.000 anos atrás. O jogo é jogado em um tabuleiro 8x8 com cada jogador começando com 12 peças da sua cor. O objetivo do jogo é capturar todas as peças do oponente ou deixá-lo sem movimentos. Os movimentos são feitos movendo-se a peça para uma casa livre na diagonal. A captura é feita na diagonal quando existe uma peça e logo em seguida uma casa livre. Ao chegar no canto oposto do tabuleiro, as peças tornam-se Damas. Apesar de existirem diferentes variações, a regra internacional diz que as peças podem apenas andar para frente, mas capturar para trás e para frente, uma ou várias peças em seqüência. As Damas podem se mover várias casas na diagonal e capturar peças na mesma diagonal desde que haja uma casa livre em seguida. O primeiro programa jogador famoso foi escrito por Arthur Samuel em 1947. O grande campeão, no entanto, é o programa Chinook, criado na Universidade de Alberta por Jonathan Schaeffer e outros.

## A.4 – Gamão



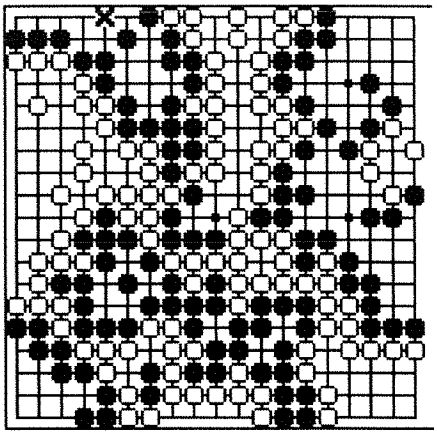
No jogo de Gamão, cada jogador tem 15 peças que percorrem 24 triângulos de acordo com a sorte de 2 dados. A situação inicial do jogo é mostrada acima. O objetivo de cada jogador é levar todas as suas peças até o final do tabuleiro percorrendo o tabuleiro em círculo pelos triângulos adjacentes. O final do tabuleiro é após o triângulo inicial do outro jogador (o que começa com 2 peças). A cada jogada, os dois dados são jogados e o jogador precisa escolher uma peça para movimentar o número de triângulos escritos em um determinado dado, depois a mesma coisa para o outro dado. O jogador não pode movimentar a sua peça para um triângulo que tenha duas ou mais peças do adversário. Quando o movimento é feito para cima de uma peça do adversário, essa peça sai do jogo e recomeça a partir da posição inicial. O programa jogador mais famoso é o TD-Gammon.

## A.5 – Go



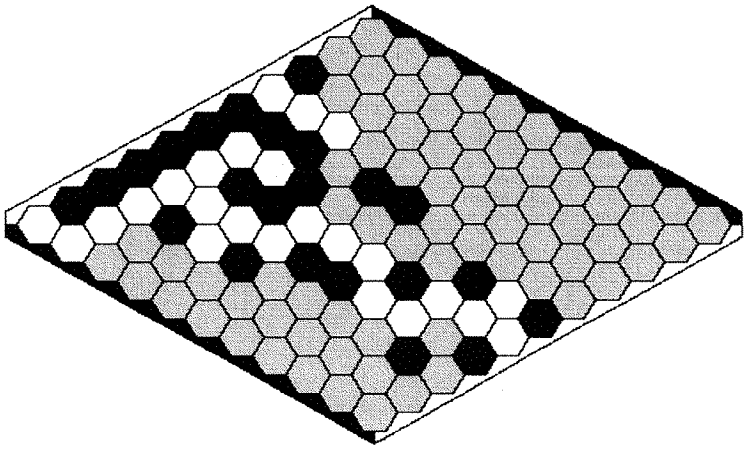
Go é um antigo jogo de tabuleiro originado na China entre 2000A.C. e 200A.C. O jogo é jogado em tabuleiro 19x19 e as pretas começam. Uma vez jogada, a peça não se move mais. Muitos pesquisadores tentam atualmente criar programas fortes de Go através de técnicas de aprendizado uma vez que as técnicas de buscas não são bem sucedidas diante do grande fator de ramificação de sua árvore de jogo. Se uma peça do adversário é circundada por peças do jogador, essa peça é removida. Um jogador não pode jogar uma peça em uma posição se ela recriar a posição anterior. Um jogador deve passar quando nenhum movimento produtivo puder ser realizado. Dois passes consecutivos terminam o jogo. A contagem de pontos de cada jogador é realizada somando-se o número de peças com o número de espaços vazios circundados pelas peças do jogador. Um dos programas mais fortes da atualidade é o "The many faces of Go", vencedor do "World Computer Go Champion 2002".

## A.6 – Go-moku



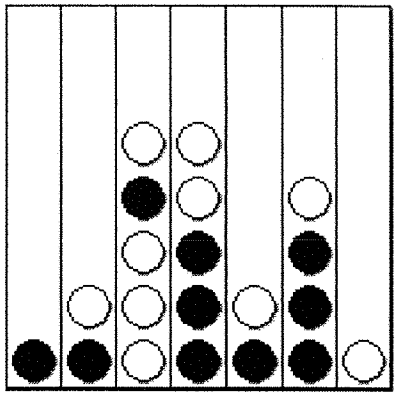
Go-Moku é um tradicional jogo de tabuleiro japonês, jogado com o mesmo tabuleiro e as mesmas peças do jogo de Go. Vence o jogador que colocar 5 peças em seqüência de sua cor ortogonalmente ou diagonalmente. As pretas sempre começam. Existe uma variante em que o jogador é obrigado a ganhar com exatamente 5 peças. Em 1992, Victor Allis resolveu o jogo utilizando pn-search, mostrando vitórias para as pretas (supondo jogo perfeito).

## A.7 – Hex



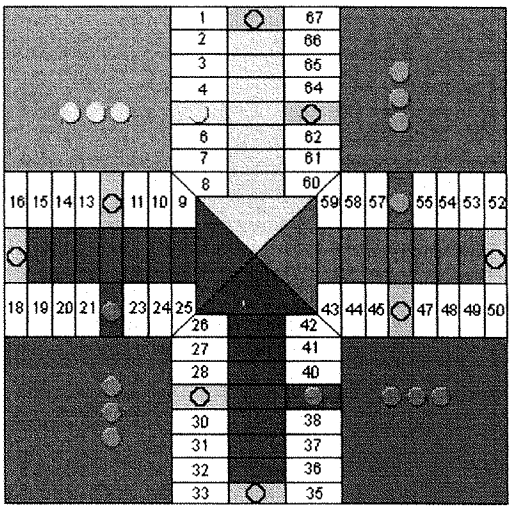
O jogo de Hex é jogado em um grid de hexágonos usualmente 11x11. O jogo foi inventado por um matemático dinamarquês em 1942. Na sua vez, cada jogador coloca uma peça da sua cor (preta ou branca) no tabuleiro preenchendo um hexágono com essa cor. O objetivo é criar um caminho unindo os lados opostos do paralelogramo com a mesma cor de borda da cor do jogador. Um ponto interessante desse jogo é que não existe empate possível, a única maneira de impedir o adversário de formar o caminho, é formando antes dele. Os dois programas mais fortes conhecidos são o Six e o Hexy.

## A.8 – Lig-4



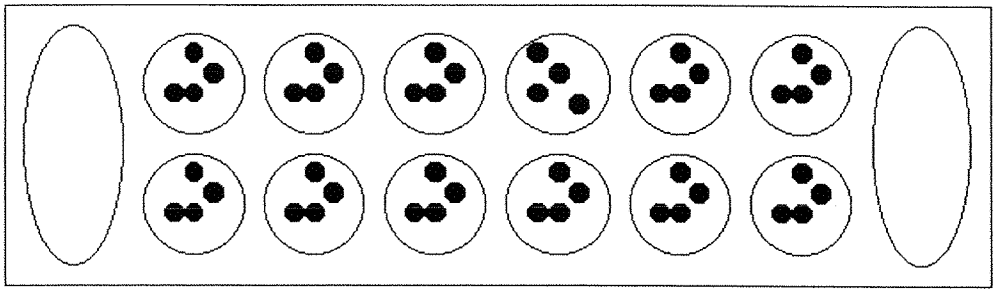
O Jogo de Lig-4 é jogado em um tabuleiro 7x6 em posição vertical. Cada jogador na sua vez, solta uma peça de sua cor do alto até que ele pare sobre a última peça da coluna jogada. O objetivo do jogo é formar 4 peças da mesma cor em linha, coluna ou diagonal antes do adversário. O jogo foi resolvido por James Allen por força bruta e por Victor Allis por uma abordagem baseada no conhecimento, mostrando que o jogo é vitória para o primeiro jogador supondo jogadas perfeitas.

## A.9 – Ludo



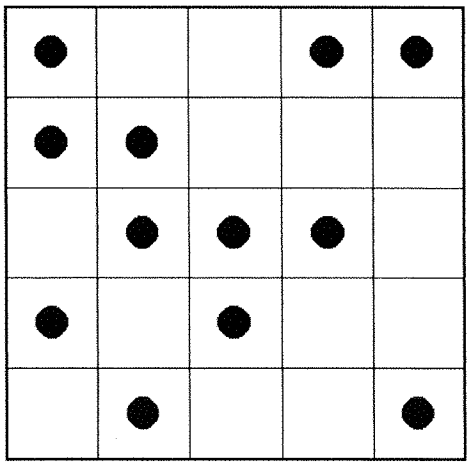

Este jogo é uma versão simplificada do jogo indiano Pachisi. A sua primeira versão surgiu na Inglaterra em 1896. O jogo pode ser jogado por mais de 2 jogadores, mas o jogo em questão aqui é o jogado por apenas 2. Cada jogador controla 8 peças que precisam partir das posições iniciais, seguir por todo o tabuleiro até atingir a parte central da mesma cor da peça. O jogador joga apenas 1 dado e escolhe uma das peças para mover. As peças só podem sair da posição inicial se o jogador tirar um 6 no dado. Se uma peça cair em cima da peça de um adversário, a peça do adversário é removida e volta para o início.

## A.10 – Mancala



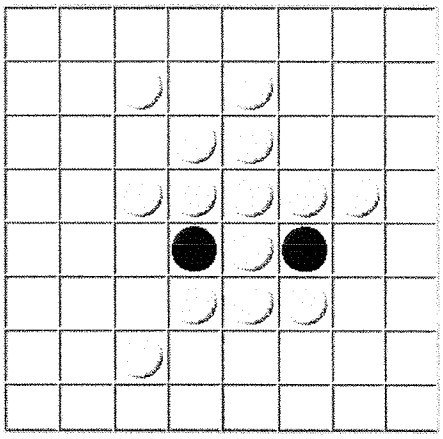
Mancala é, na verdade, o nome de uma família de jogos semelhantes provavelmente originados na Etiópia. Uma das variantes mais conhecidas dos pesquisadores é o jogo de Awari. Cada jogador começa com 24 peças no seu lado distribuídas igualmente por 6 minas. A cada jogada, o jogador deve escolher uma das minas, retirar todas as peças dela e distribuir uma a uma nas minas seguintes no sentido anti-horário. Se a última peça for colocada em uma mina oponente e completar duas ou três peças, as peças são capturadas e colocadas na Mancala do jogador (mina maior lateral). O jogo termina quando um jogador não tem mais peças para mover. O vencedor é aquele que tiver mais peças na sua mancala. O jogo de Awari foi resolvido em 2002, por força-bruta, mostrando que o jogo é um empate para jogadores perfeitos.

## A.11 – Nim



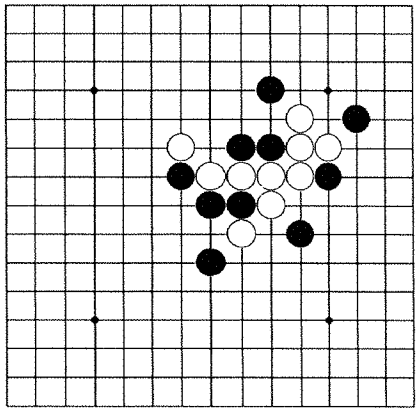
O Jogo de Nim possui muitas variantes, algumas com solução matemática simples. Cada jogador deve tirar um número de peças qualquer de uma única linha na sua vez. Adicionalmente, ele pode adicionar uma única peça do lado direito de uma que está sendo retirada (a menos que ela já esteja na última coluna à direita). O jogador que tirar a última peça perde o jogo.

## A.12 – Othello



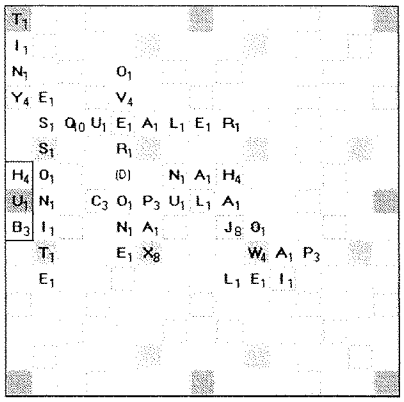
O jogo de Othello, também conhecido por Reversi, é jogado em um tabuleiro 8x8 com peças de 2 cores. A cada jogada, o jogador precisa colocar uma peça da sua cor em uma casa vazia. A única restrição para essa casa ser um movimento válido, é que, ortogonalmente ou diagonalmente, existam só peças do adversário entre essa nova peça e outra peça da mesma cor do jogador. Sendo assim, todas as peças, nessas condições, em todas as direções, são alteradas para a cor do jogador. Um dos mais fortes programas jogadores da história foi o Logistello que derrotou o campeão mundial humano Takeshi Murakama por 6:0.

## A.13 – Renju



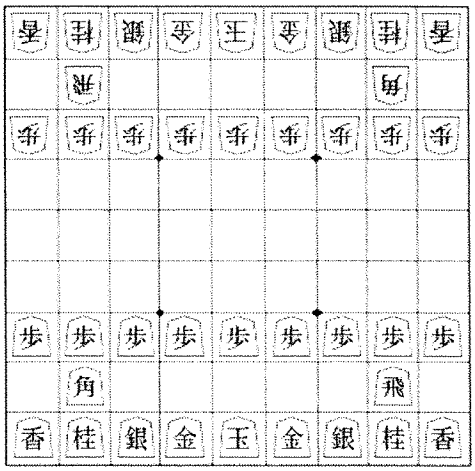
O jogo de Renju é uma versão profissional do jogo Go-moku. Ele é jogado em um tabuleiro de Go 15x15. Existe uma seqüência especial em que os jogadores jogam peças suas e do adversário e também retiram. Depois o jogo segue normalmente. O objetivo das pretas é colocar 5 peças em seqüência, enquanto o objetivo das brancas é colocar 5 ou mais peças em seqüência ou obrigar as pretas a fazer um dos movimentos proibidos (colocar peça que gera uma das seguintes situações: 6 ou mais pretas em linha, duas seqüências com 4 pretas ou duas seqüências não-quebradas com 3 pretas). Um dos melhores programas jogadores foi o Vertex.

## A.14 – Scrabble



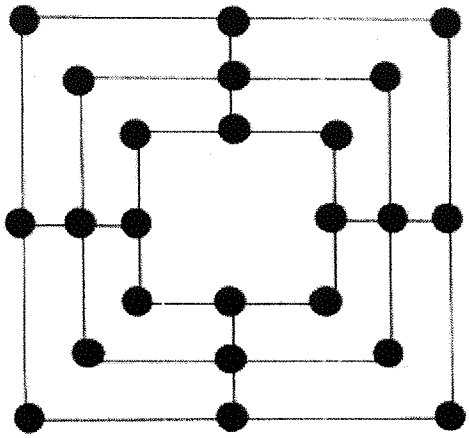
Scrabble (1938) é um jogo onde os jogadores usam 100 peças com letras escritas para formar palavras em tabuleiro 15x15 com certas casas especiais que valem pontos. O nome é uma marca registrada da Hasbro nos Estados Unidos e no Canadá. No início do jogo, cada jogador tem 7 letras à disposição (escondidas). A cada jogada, o jogador pode passar, trocar algumas peças suas por um mesmo número de peças do monte, ou fazer uma palavra no tabuleiro marcando pontos (esquerda para direita ou cima para baixo). Após fazer uma palavra, o jogador pega mais peças para completar as suas 7. O jogo acaba quando acaba o monte e as peças na mão de um jogador. As palavras aceitas são normalmente entradas primárias de um dicionário e suas formas flexionadas. O melhor programa de Scrabble conhecido é o Maven, criado por Brian Sheppard.

## A.15 – Shogi



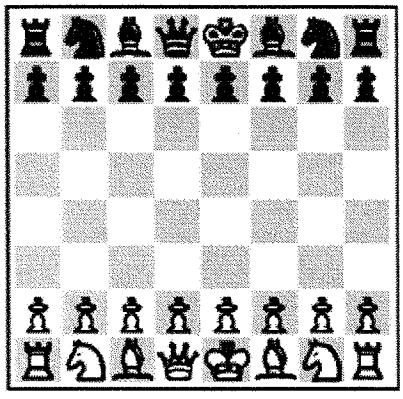
O Shogi, também conhecido como Xadrez Japonês, é um jogo jogado em um tabuleiro 9x9 cujo objetivo é capturar o Rei do oponente. Cada jogador inicia com 20 peças: 1 Rei, 2 generais ouro, 2 generais prata, 2 cavalos, 2 arqueiros, 1 bispo, 1 torre e 9 peões. A figura acima corresponde à configuração inicial do jogo com rei na casa central da primeira linha de cada jogador. Cada peça move de uma forma diferente. Diferente do Xadrez tradicional, as peças capturadas podem voltar ao jogo sob o controle do outro jogador. As peças podem ser promovidas se estiverem nas três linhas mais distantes do jogador. Elas devem ser viradas e assumem uma outra função. Nem o Rei nem o General ouro podem ser promovidos. O atual melhor programa jogador de Shogi é o YSS, vencedor da Oitava Olimpíada de Computadores.

## A.16 – Trilha



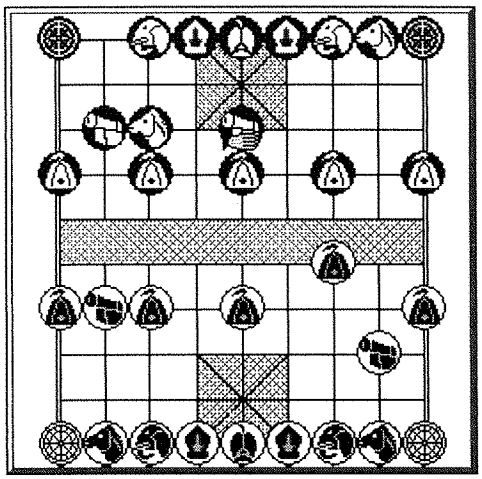
No jogo de trilha, cada jogador tem 9 peças para movimentar pelas 24 posições existentes no jogo. O objetivo do jogo é remover todas as peças do adversário. O jogo começa com o tabuleiro vazio e cada jogador coloca a uma peça na sua vez. Quando as peças terminam de ser colocadas, os jogadores começam a se movimentar. Um movimento consiste em mover uma peça para uma posição adjacente. Em 1993, Ralph Gasser resolveu o jogo de Trilha, mostrando que o jogo é empate para jogadores perfeitos. O melhor programa conhecido de trilha é o Bushy.

## A.17 – Xadrez



O jogo de Xadrez é provavelmente derivado do jogo Chaturanga e foi criado há 1.400 anos atrás. Ele é jogado em um tabuleiro 8x8 com 16 peças para cada lado (1 rei, 1 dama, 2 cavalos, 2 bispos, 2 torres e 8 peões). O objetivo do jogo é colocar o rei adversário em Xeque-mate (posição em que o Rei está sendo atacado e qualquer movimento recai em uma outra situação de ataque). Cada peça move-se de uma forma diferente, sendo que apenas o cavalo pode passar por cima das peças. A captura se dá com uma peça sendo posicionada na mesma posição de uma peça adversária. O programa jogador de Xadrez mais conhecido foi o Deep Blue que foi descontinuado após sua vitória sobre Kasparov em 1997.

## A.18 – Xadrez chinês



O Xadrez Chinês, também chamado de Xiangqi, tem a mesma origem do Xadrez e do Shogi. O jogo é jogado em um tabuleiro 9x10 e cada jogador inicia com 16 peças (1 general, 2 guardas, 2 ministros, 2 cavalos, 2 carruagens, 2 canhões e 5 soldados). As peças são colocadas na intersecção das linhas. A região central cinza de cada um dos lados corresponde ao palácio de onde os guardas e o rei não podem sair. A linha central do tabuleiro corresponde ao rio do qual os ministros não podem atravessar. Apesar de ser um jogo mais rápido do que o Xadrez tradicional (uma vez que não há a barreira dos peões e os canhões saltam para capturar), a complexidade da árvore de jogo é bem maior, cerca de $10^{150}$. O Atual melhor programa jogador de Xadrez Chinês é o ZMBL, vencedor da Oitava Olimpíada de Computadores.