

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Prado Prandini Faria

**Investigação de Desempenho dos Algoritmos
Paralelos de Busca YBWC (síncrono) e APHID
(assíncrono) nos Jogadores Automáticos**

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Prado Prandini Faria

**Investigação de Desempenho dos Algoritmos Paralelos de
Busca YBWC (síncrono) e APHID (assíncrono) nos
Jogadores Automáticos**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Ciência da Computação.

Orientadora: Rita Maria da Silva Julia

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2017

Matheus Prado Prandini Faria

Investigação de Desempenho dos Algoritmos Paralelos de Busca YBWC (síncrono) e APHID (assíncrono) nos Jogadores Automáticos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho (aprovado). Uberlândia, Brasil, 08 de dezembro de 2017:

Prof. Dra. Rita Maria da Silva Julia

Prof. Dra. Gina Maira Barbosa de Oliveira

Prof. Dr. Paulo Henrique Ribeiro Gabriel

Uberlândia, Brasil
2017

Dedico este trabalho aos meus pais, Cássia e Marcelo, e à minha irmã, Luísa, por me incentivarem em todos os aspectos da minha vida.

Agradecimentos

Agradeço à Deus, pelo dom da vida e por todas as oportunidades que me foram concedidas.

Agradeço aos meus pais e à minha irmã por estarem comigo em todos os momentos da minha vida, além do apoio, carinho e companheirismo incondicional que sempre compartilhamos.

Agradeço à Professora Rita Maria da Silva Julia pela orientação, dedicação, confiança, enfim, por todo o apoio e aprendizado que me proporcionou para a execução deste trabalho e também por toda a contribuição durante o percurso da minha graduação dentro e fora da sala de aula. Sou muito grato pela amizade e carinho que sempre teve comigo.

Agradeço à Lídia Bononi Paiva Tomaz pela co-orientação, dedicação, paciência, aprendizado, experiência e seriedade que sempre teve no processo de me auxiliar desde que iniciei as minhas atividades de pesquisa junto à Professora Rita. Sou muito grato pela sua amizade e pela contribuição no meu aprendizado durante a minha graduação.

Agradeço a todos os Professores de graduação da FACOM/UFU que pude ter o prazer de conhecer durante a graduação pelo conhecimento e experiência compartilhados ao longo desse período.

Por fim, agradeço a todos os amigos que fiz desde que ingressei no curso de Ciência da Computação por todo o companheirismo, amizade e incentivo que sempre tivemos uns com os outros.

Obrigado!

"Lembre-se de que você mesmo é o melhor secretário de sua tarefa, o mais eficiente propagandista de seus ideais, a mais clara demonstração de seus princípios, o mais alto padrão do ensino superior que seu espírito abraça e a mensagem viva das elevadas noções que você transmite aos outros. Não se esqueça, igualmente, de que o maior inimigo de suas realizações mais nobres, a completa ou incompleta negação do idealismo sublime que você apregoa, a nota discordante da sinfonia do bem que pretende executar, o arquiteto de suas aflições e o destruidor de suas oportunidades de elevação - é você mesmo." - Chico Xavier

Resumo

Os algoritmos de busca da Inteligência Artificial YBWC (*Young Brother Wait Concept*) e APHID (*Asynchronous Parallel Hierarchical Iterative Deepening*) apresentam duas formas distintas de paralelismo do algoritmo sequencial Alfa-Beta. O primeiro segue uma abordagem síncrona, enquanto o segundo segue uma abordagem assíncrona. Dessa forma, o presente trabalho tem as seguintes metas principais: efetuar uma comparação conceitual entre as vantagens e as lacunas propiciadas pela abordagem síncrona do YBWC e pela abordagem assíncrona do APHID operando em arquiteturas de memória distribuída (por serem mais disponíveis, em função do custo, nos meios acadêmicos); implementar um ambiente que permita validar esse estudo teórico por meio da avaliação automática de desempenho de agentes jogadores construídos com base em cada um desses algoritmos. Neste trabalho, foram usados os seguintes agentes jogadores de Damas como base de avaliação do desempenho de ambos os algoritmos: *D-VisionDraughts* (baseado no YBWC) e o *APHID-Draughts* (baseado no APHID). Neste contexto, o segundo objetivo acima citado foi executado por meio da implementação de uma interface que permitiu conectar tais jogadores automáticos à plataforma internacional de Damas conhecida como *CheckerBoard*. Tal plataforma, além de integrar os principais agentes automáticos de Damas existentes, permite também a realização de jogos *on-line* envolvendo humanos. Neste cenário, a interface implementada no presente trabalho possibilitou a avaliação do desempenho dos agentes usados como ferramentas de teste do YBWC (*D-VisionDraughts*) e do APHID (*APHID-Draughts*) da seguinte forma: as escolhas de movimento efetuadas por ambos em torneios em que eles se enfrentaram foram submetidas, via interface implementada, ao melhor agente disponível na plataforma. O desempenho de cada um foi avaliado por meio de uma comparação automática entre as taxas de coincidência de movimento com relação ao consagrado agente *Cake*. Os resultados obtidos por tal análise permitiram validar a superioridade do agente *APHID-Draughts* sobre o *D-VisionDraughts*, o que corrobora o fato de que o algoritmo paralelo de busca APHID apresenta um desempenho melhor - considerando arquitetura de memória distribuída - em comparação ao algoritmo YBWC.

Palavras-chave: Aprendizagem de Máquina, Asynchronous Parallel Hierarchical Iterative Deepening, Busca Paralela, Damas, Young Brother Wait Concept.

Lista de ilustrações

Figura 1 – Exemplo de árvore explorada pelos algoritmos (a) Minimax e (b) poda Alfa-Beta	20
Figura 2 – (a) Estados do Processador Principal; (b) Estados dos Processadores Auxiliares	33
Figura 3 – (a) Estados do Processador Mestre; (b) Estados do Processador Escravo	35
Figura 4 – Localização dos pontos de sincronização nos algoritmos APHID e YBWC (BROCKINGTON; SCHAEFFER, 2000)	38
Figura 5 – (a) Árvore de Busca de exemplo (b) Ilustração da estrutura de controle de nós (pilha) do processador $P0$ na expansão do nó $N1$	39
Figura 6 – Solicitações de tarefas efetuadas pelo processador $P1$	40
Figura 7 – Possível distribuição de tarefas em um determinado momento a partir do YBWC	41
Figura 8 – Exemplo da árvore explorada pelo processador mestre no APHID . . .	42
Figura 9 – Exemplo da estrutura Tabela Aphid no processador mestre ($P0$) e escravos ($P1$, $P2$ e $P3$)	43
Figura 10 – Exemplo da exploração do nó $N16$ no processador escravo $P1$ por ID considerando o valor do incremento igual a 1	45
Figura 11 – Mecanismo de atualização da janela alfa-beta no algoritmo YBWC . .	48
Figura 12 – Mecanismo de atualização da janela alfa-beta no algoritmo APHID . .	49
Figura 13 – Exemplificação da alocação de tarefas a um processador quando utilizada a estratégia de busca por Aprofundamento Iterativo no YBWC (a) e no APHID (b)	52
Figura 14 – Arquitetura do sistema de automatização da análise estatística de movimentos	56
Figura 15 – Arquitetura geral do jogador D-VisionDraughts	58
Figura 16 – Tempo de treinamento (em minutos) dos jogadores: <i>VisionDraughts</i> , <i>D-VisionDraughts</i> (P=10) e <i>D-VisionDraughts</i> (P=16) (TOMAZ; JULIA; BARCELOS, 2013)	62
Figura 17 – Tempo de treinamento (em minutos) dos jogadores D-VisionDraughts e APHID-Draughts por profundidade	68

Lista de tabelas

Tabela 1 – Complexidade do espaço de estados e fator de ramificação de alguns jogos	13
Tabela 2 – Divisão de tarefas segundo o modo round-robin entre três processadores	43
Tabela 3 – Pontos positivos do algoritmo YBWC	54
Tabela 4 – Pontos negativos do algoritmo YBWC	54
Tabela 5 – Pontos positivos do algoritmo APHID	55
Tabela 6 – Pontos negativos do algoritmo APHID	55
Tabela 7 – <i>D-VisionDraughts</i> (P=16) contra <i>VisionDraughts</i> (TOMAZ; JULIA; BARCELOS, 2013)	63
Tabela 8 – Taxas de coincidência de movimentos com o Cake	64
Tabela 9 – Método <i>Wilcoxon signed rank test</i> aplicado aos resultados dos torneios entre o <i>D-VisionDraughts</i> e <i>VisionDraughts</i> da Tabela 7	65
Tabela 10 – Taxas obtidas pelo <i>APHID-Draughts</i> em jogos disputados contra o <i>D-VisionDraughts</i>	69
Tabela 11 – Método <i>Wilcoxon signed rank test</i> aplicado aos resultados do torneio entre o <i>APHID-Draughts</i> e o <i>D-VisionDraughts</i>	71
Tabela 12 – Taxas de coincidência de movimentos com o Cake	71

Lista de abreviaturas e siglas

APHID	<i>Asynchronous Parallel Hierarchical Iterative Deepening</i>
IA	Inteligência Artificial
ID	<i>Iterative Deepening</i>
MLP	<i>Multilayer Perceptron</i>
PDN	<i>Portable Draughts Notation</i>
TT	Tabela de Transposição
YBWC	<i>Young Brothers Wait Concept</i>

Sumário

1	INTRODUÇÃO	12
1.1	Objetivos	15
1.2	Metodologia	15
1.2.1	Metodologia referente ao objetivo 1: Comparar conceitualmente as abordagens síncrona (YBWC) e assíncrona (APHID) do algoritmo Alfa-Beta	16
1.2.2	Metodologia referente ao objetivo 2: Implementação de uma ferramenta avaliadora da qualidade da tomada de decisão dos algoritmos YBWC e APHID aplicados a agentes jogadores de Damas	17
2	REVISÃO BIBLIOGRÁFICA	19
2.1	Referencial Teórico	19
2.1.1	Algoritmo de Busca Alfa-Beta	19
2.1.2	YBWC	20
2.1.3	APHID	21
2.2	Trabalhos Correlatos	23
3	PLATAFORMA CHECKERBOARD	25
3.1	Arquivo PDN	25
4	AGENTES AUTOMÁTICOS JOGADORES DE DAMAS	27
4.1	Agentes Automáticos Jogadores de Damas: Supervisionados	27
4.1.1	Chinook	27
4.1.2	Cake	28
4.2	Agentes Automáticos Jogadores de Damas: Não Supervisionados	28
4.2.1	VisionDraughts	28
4.2.2	D-VisionDraughts	29
4.2.3	D-MA-Draughts	29
4.2.4	APHID-Draughts	30
5	COMPARAÇÃO ENTRE OS ALGORITMOS YBWC E APHID	31
5.1	Comunicação entre os processadores	32
5.1.1	Comunicação no YBWC	32
5.1.2	Comunicação no APHID	34
5.1.2.1	Tabela Aphid	36
5.2	Distribuição e Sincronização de Tarefas	37
5.2.1	Distribuição e sincronização de tarefas no YBWC	38

5.2.2	Distribuição e sincronização de tarefas no APHID	41
5.2.2.1	Atividades do Processador Mestre	42
5.2.2.2	Atividades dos Processadores Escravos	44
5.3	Janela de Busca Alfa-Beta	47
5.4	Tabela de Transposição em Arquitetura de Memória Distribuída . .	51
5.4.1	TT distribuída no YBWC	52
5.4.2	TT distribuída no APHID	52
5.5	Portabilidade	53
5.6	Considerações Finais	54
6	MÉTODO AUTOMATIZADO PARA AVALIAÇÃO DE QUALIDADE DE MOVIMENTOS DE AGENTES JOGADORES DE DAMAS . .	56
6.1	Geração do Arquivo PDN	57
6.2	Sub-Módulo AAC: Análise Automática do <i>Cake</i>	59
6.3	Sub-Módulo ECM: Estatística de Coincidência de Movimentos . . .	60
6.4	Considerações Finais	61
7	EXPERIMENTOS E RESULTADOS	62
7.1	Estudo de Caso 1: VisionDraughts x D-VisionDraughts	62
7.1.1	Análise Estatística de Wilcoxon	64
7.1.2	Considerações do Estudo de Caso 1	66
7.2	Estudo de Caso 2: D-VisionDraughts x APHID-Draughts	67
7.2.1	Tempo de Treinamento dos Agentes	67
7.2.2	Taxas de Vitória	69
7.2.2.1	Análise Estatística de Wilcoxon	70
7.2.2.2	Análise da Qualidade dos Movimentos com relação ao <i>Cake</i>	71
7.2.3	Considerações finais sobre o estudo de caso 2	72
8	CONCLUSÃO	73
	REFERÊNCIAS	74

1 Introdução

A Teoria dos Jogos tornou-se um importante segmento da matemática que foi aplicada de forma mais intensa durante a metade do século passado, especialmente após a publicação do clássico livro *The Theory of Games and Economic Behavior* (NEUMANN; MORGENTERN, 1953). Desde então, tem despertado interesse no campo da Ciência da Computação, onde tem sido usada nos avanços da Inteligência Artificial (IA) e da Cibernética. A Teoria dos Jogos se propõe a encontrar estratégias racionais para problemas onde o resultado não depende apenas do ambiente e da estratégia de um único agente, mas também das estratégias de outros agentes que interagem com o primeiro.

Neste contexto, este trabalho utiliza o jogo de Damas como “laboratório” de estudo para aplicação de técnicas de IA. A escolha de Damas como domínio de aplicação se deve ao fato de esse tipo de jogo apresentar muitas semelhanças com problemas práticos do dia a dia, uma vez que tais problemas também requerem combinações de estratégias por parte do agente para tomada de decisões em situações em que um oponente tenta minimizar o efeito positivo de suas ações (TOMAZ; JULIA; BARCELOS, 2013). Além disso, o agente que atua nesse domínio deve aprender através de sua interação com o ambiente, bem como através das mudanças de estado que ocorrem após a execução de uma ação (movimento). Como exemplo de problema prático, cita-se o problema de iteração entre homens e máquinas por meio de diálogo (WALKER, 2000). Outra motivação para utilização do jogo de Damas como “laboratório” de estudo está no fato de esse jogo possuir uma elevada complexidade do espaço de estados¹ e fator de ramificação² (HERIK; UITERWIJK; RIJSWIJCK, 2002), o que representa um desafio no processo de tomada de decisão de um agente jogador, conforme ilustrado na Tabela 1 (CAMPOS; LANGLOIS, 2003).

Tais dados retratados na Tabela 1 reforçam os argumentos que tornam os jogos de tabuleiro - dentre eles, destacadamente o jogo de Damas - um domínio bastante rico a ser explorado em Aprendizagem de Máquina. De fato, vários agentes jogadores de Damas vêm sendo propostos pela comunidade científica internacional, tanto no contexto da aprendizagem supervisionada (quando o processo de aprendizagem é fortemente norteado por especialistas humanos ou por bases de dados), quanto no contexto da não-supervisionada (quando o agente aprende baseado essencialmente em suas próprias experiências) (RUSSELL; NORVIG, 2004). Os melhores jogadores automáticos de Damas da atualidade

¹ A complexidade do espaço de estados é definida pelo número de posições legais possíveis no jogo, que podem ser alcançadas a partir do estado inicial do jogo.

² O fator de ramificação determina a complexidade da árvore de jogo, uma vez que é formado pelo número de folhas na árvore de busca para a solução da posição atual (ou estado)

Tabela 1 – Complexidade do espaço de estados e fator de ramificação de alguns jogos

Jogo	Fator de Ramificação	Espaço de Estados
Xadrez	30-40	10^{50}
Damas	8-10	10^{17}
Gamão	≈ 420	10^{20}
Othello	≈ 5	$\approx 10^{30}$
Go 19x19	≈ 360	$\approx 10^{160}$
Abalone	≈ 80	$\approx 3^{61}$

pertencem ao grupo dos jogadores supervisionados. Dentre esses jogadores destacam-se o *Chinook*, atual campeão mundial homem-máquina (SCHAEFFER et al., 1996), e o *Cake* (FIERZ, 2010), o qual já derrotou uma versão preliminar do *Chinook* - versão esta famosa por ter derrotado o campeão mundial de Damas da época, *Marion Tinsley*. Convém salientar que o presente trabalho enquadra-se em atividades de pesquisa voltadas para uma aprendizagem de máquina mais autônoma, em que as técnicas de IA aplicadas ao jogo de Damas estão direcionadas à criação de um agente que aprende com o mínimo de supervisão humana, desenvolvidas em um escopo global de Doutorado.

Tal pesquisa de Doutorado é desenvolvida por Lídia Bononi Paiva Tomaz - coorientadora deste trabalho e também aluna orientada pela Professora Doutora Rita Maria da Silva Julia - na área de aperfeiçoamento de técnicas de Aprendizagem de Máquina cujo estudo de caso é a plataforma multiagente jogadora de Damas em ambiente de alto desempenho, denominada *D-MA-Draughts*. Particularmente, o objetivo principal dessa atividade de Doutorado é propor uma nova versão de algoritmo de busca distribuído a ser usada como ferramenta tomadora de decisões (escolha de movimentos apropriados) no referido sistema jogador. Esta pesquisa de Doutorado vem sendo desenvolvida ao longo das seguintes fases:

- Estudo dos algoritmos de busca distribuídos mais relevantes e utilizados em jogos. Esta atividade é concentrada na análise dos algoritmos YBWC e APHID, os quais são baseados nas abordagens síncrona e assíncrona, respectivamente.
- Implementação de um sistema mono-agente jogador de Damas baseado no algoritmo de busca APHID denominado *APHID-Draughts* 4.2.4. É importante destacar que a equipe pesquisadora já implementou, em trabalhos anteriores, as seguintes plataformas jogadoras: *VisionDraughts* 4.2.1, sistema mono-agente baseado no algoritmo de busca serial Alfa-Beta; mono-agente *D-VisionDraughts* 4.2.2 e Multiagente *D-MA-Draughts* 4.2.3, baseados no algoritmo de busca distribuído YBWC;
- Análise comparativa detalhada entre os algoritmos de busca distribuídos YBWC e APHID, a fim de averiguar os principais pontos positivos e negativos de ambos.

Ressalta-se que nesta etapa, além do levantamento comparativo de características conceituais entre tais algoritmos, o desempenho em jogos dos agentes implementados também será considerado;

- Por fim, com base nos resultados da fase anterior, propor um novo algoritmo de busca distribuído - denominado *Asynchronous Distributed Alpha-Beta Algorithm* (ADABA) - de modo a aproveitar os aspectos positivos de cada abordagem (síncrona, YBWC, e assíncrona, APHID). Tal algoritmo produzido será implementado na plataforma multiagente de Damas *D-MA-Draughts*, a fim de aperfeiçoar o processo de tomada de decisão de tal sistema jogador.

Salienta-se que as atividades aqui expostas são desenvolvidas simultaneamente com uma Iniciação Científica de mesmo tema e representam a continuidade de um trabalho preliminar (de Iniciação Científica), conduzido pelo mesmo aluno. Dentre as atividades realizadas nesse trabalho preliminar, destacam-se as seguintes: desenvolvimento de uma interface de comunicação entre o jogador automático *DMA-Draughts* e a plataforma internacional de Damas *CheckerBoard* (??), permitindo que o referido agente dispute partidas “online” contra adversários humanos ou automáticos que estejam conectados com tal plataforma (tal interface é uma ferramenta muito útil para permitir a avaliação de desempenho do agente em foco); a segunda atividade preliminar que foi desenvolvida naquela primeira atuação em Iniciação Científica refere-se ao auxílio nos trabalhos de implementação dos algoritmos YBWC e APHID usados nas plataformas jogadoras produzidos pela equipe pesquisadora, a fim de permitir que os módulos de busca de tais plataformas armazenem as informações de movimentos realizados em partidas disputadas em arquivos padrões de jogos de Damas, denominados *Portable Draughts Notation* (PDN). Nesta direção, o presente trabalho tem como objetivo central auxiliar na continuidade das atividades do referido doutoramento, concentrando-se na seguinte meta: auxílio no aprimoramento e refinamento do processo de escolha de movimentos (tomada de decisão) do sistema jogador *D-MA-Draughts*, o qual vem sendo desenvolvido no âmbito da referida pesquisa de Doutorado no qual este trabalho está alocado.

Assim sendo, a fim de almejar tal meta principal, o presente trabalho pretende auxiliar nas análises comparativas dos algoritmos de busca distribuídos YBWC e APHID, além de realizar a implementação de uma ferramenta de testes que permita avaliar o desempenho de agentes de Damas em jogos quanto à qualidade da tomada de decisão. Tal ferramenta deverá ser acusada como instrumento comparador da performance de agentes implementados de acordo com as abordagens de algoritmos de busca citados, objetivando, particularmente, no caso das pesquisas em curso, a avaliar o ganho de desempenho obtido com a proposta do ADABA com relação a seus precursores. Dessa forma, as principais contribuições deste trabalho são as seguintes:

- Auxílio nas atividades de análise comparativa entre as abordagens síncrona (YBWC) e assíncrona (APHID) desenvolvidas no escopo global de Doutorado;
- Implementação de uma ferramenta automática de avaliação de agentes jogadores de Damas. Destaca-se que tal ferramenta foi testada e validada com sucesso nos seguintes cenários: *VisionDraughts* x *D-VisionDraughts* (no qual o uso da ferramenta foi corroborado pela coerência entre os resultados obtidos por ela e pelos resultados alcançados por trabalhos previamente realizados); *APHID-Draughts* x *D-VisionDraughts* - no qual a ferramenta foi aplicada como instrumento de teste em um artigo publicado no *International Conference on Tool with Artificial Intelligence* (ICTAI). Tais estudos de casos serão detalhados no capítulo 7;
- Ao término da implementação do algoritmo de busca ADABA (produzido no escopo global de Doutorado), ele será avaliado também por meio da ferramenta aqui implementada e validada.

1.1 Objetivos

O objetivo geral deste trabalho é propiciar uma estrutura de apoio para o refinamento do processo de escolha de ações (jogadas) dos agentes do (*DMA-Draughts*) previsto para ser cumprido no projeto global no qual o presente trabalho se insere. Para tanto, os objetivos específicos a serem perseguidos aqui são:

1. Investigação dos aspectos positivos e das fragilidades das abordagens paralelas síncrona (YBWC) e assíncrona (APHID) do algoritmo Alfa-Beta implementadas em uma arquitetura de memória distribuída.
2. Implementação de um sistema automático para avaliação da qualidade da tomada de decisão dos algoritmos YBWC e APHID a partir da plataforma *CheckerBoard*, de modo a auxiliar no processo de comparação de desempenho de tais algoritmos aplicados a agentes automáticos jogadores de Damas.

1.2 Metodologia

O desenvolvimento deste trabalho iniciou-se com o auxílio no processo de comparação da performance das abordagens síncrona e assíncrona, representadas pelos algoritmos YBWC e APHID, respectivamente, aplicadas em agentes jogadores de Damas implementados pela equipe pesquisadora. Ressalta-se que para o estabelecimento de tal análise comparativa entre ambas abordagens, o YBWC foi representado por um dos agentes da versão preliminar do *D-MA-Draughts*, denominado *D-VisionDraughts*. Por outro

lado, o APHID foi representado pela arquitetura mono-agente desenvolvida no âmbito do trabalho de doutorado conduzido pela orientadora deste trabalho denominado *APHID-Draughts*. Uma vez concluída a etapa de comparação conceitual dos referidos algoritmos, uma nova análise comparativa foi realizada de forma a avaliar a qualidade das jogadas de tais algoritmos de busca quando aplicados a agentes automáticos jogadores de Damas, por meio de uma ferramenta automática implementada neste trabalho. Dessa forma, as subseções a seguir descrevem a metodologia contemplada em cada um dos objetivos deste trabalho.

1.2.1 Metodologia referente ao objetivo 1: Comparar conceitualmente as abordagens síncrona (YBWC) e assíncrona (APHID) do algoritmo Alfa-Beta

É importante salientar que a abordagem de paralelização síncrona do algoritmo Alfa-Beta é empregada na atual versão do *D-MA-Draughts*. Contudo, nessa análise comparativa, tal abordagem será representada por um dos agentes da versão preliminar do referido agente de Damas, denominado *D-VisionDraughts*. Por outro lado, a abordagem assíncrona será representada pela arquitetura mono-agente desenvolvida no âmbito do trabalho de doutorado conduzido pela orientadora deste trabalho, denominada *APHID-Draughts*. Desta forma, tornou-se possível criar um cenário comparativo entre ambas abordagens o qual contemplou os seguintes aspectos técnicos:

- **Comunicação entre Processadores:** característica básica dos algoritmos paralelos, uma vez que deve haver um canal que possibilite o envio e o recebimento de informações para que cada processador tenha condições de se direcionar corretamente para o conjunto de instruções que deve executar. A forma mais comum para realizar esta comunicação é por troca de mensagens. Neste contexto, tanto o YBWC quanto o APHID possui o seu próprio conjunto de mensagens que são trocadas entre os processadores a fim de permitir a evolução da busca [5.1](#).
- **Sincronização de Tarefas:** para que os algoritmos paralelos cheguem à solução para a qual foram propostos, é imprescindível que eles obtenham as informações que foram distribuídas entre o conjunto de processadores disponíveis e finalizem as instruções que irão conduzi-los à solução final. Para isso, deve existir uma sincronização dos resultados que foram explorados separadamente. Neste contexto, cada abordagem de paralelização - síncrona e assíncrona - propõe um modelo de sincronização dos dados de modo a permitir que o algoritmo centralize os resultados dos nós explorados por outros processadores e retorne a solução final [5.2](#).
- **Janela de Busca Alfa-Beta:** o algoritmo Alfa-Beta possui a característica de que os ramos necessitam de informações fornecidas por outros ramos - a janela de busca

- a fim de melhorar a eficiência do algoritmo através da poda. Neste sentido, as abordagens paralelas para o Alfa-Beta propõem meios de realizar o compartilhamento de informações dos limites da janela de busca de modo a reduzir a sobrecarga da busca. Particularmente, os algoritmos YBWC e APHID possuem mecanismos muito distintos para tratar a janela de busca, os quais serão explicados e detalhados 5.3.
- **Tabela de Transposição:** uma tabela de transposição (TT) atua como um repositório de estados já avaliados evitando que sejam reprocessados. Neste sentido, há duas formas de implementação de uma TT: distribuída e compartilhada 5.4. A arquitetura disponível para implementação destes algoritmos é de memória distribuída, logo a utilização de tabela de transposição compartilhada tem um custo proibitivo. Tal fato é devido à carga elevada de dados que trafegariam pela rede dado que todos os processadores realizariam as ações de leitura/escrita simultaneamente e com frequência. Dessa forma, pretende-se verificar qual algoritmo utiliza de modo mais eficaz tabelas de transposição local.
- **Portabilidade:** um dos fatores que influenciam na escolha de um algoritmo a ser inserido em um problema é a complexidade da implementação sobre a aplicação. Se esta complexidade for pequena, o desenvolvedor e/ou pesquisador terá maior motivação em adotar tal algoritmo. Neste sentido, almeja-se verificar as consequências da integração dos algoritmos YBWC e APHID em sistemas que utilizam uma versão sequencial do algoritmo Alfa-Beta 5.5.

1.2.2 Metodologia referente ao objetivo 2: Implementação de uma ferramenta avaliadora da qualidade da tomada de decisão dos algoritmos YBWC e APHID aplicados a agentes jogadores de Damas

No cenário das pesquisas relacionadas a agentes jogadores de Damas, é muito importante que se conte com uma ferramenta apropriada de avaliação do desempenho dos mesmos durante seu processo de aprendizagem. Diversos agentes jogadores automáticos propostos na literatura têm o desempenho avaliado a partir de parâmetros como: o número de vitórias obtidas em torneios disputados, o tempo de tomada de decisão (ou escolha de movimento) e o tempo de treinamento do agente (CAIXETA; JULIA, 2008), (TOMAZ; JULIA; BARCELOS, 2013), (DUARTE et al., 2015), (TOMAZ; JULIA; DUARTE, 2017). Trabalhos mais recentes têm realizado a avaliação de desempenho por meio da comparação manual entre as tomadas de decisão dos agentes avaliados e as tomadas de decisão que o renomado agente *Cake* faria nas mesmas condições (NETO; JULIA; CAEXETA, 2014) e (DUARTE et al., 2015). No entanto, tal método de avaliação manual é muito trabalhoso e oneroso em termos de tempo, fato que inviabiliza realizar essa avaliação em uma elevada quantidade de jogos (o que representaria a situação ideal). Por exemplo, em

(NETO; JULIA; CAEXETA, 2014) esta avaliação foi realizada em 14 jogos em que foram considerados apenas os 15 primeiros movimentos. Em (DUARTE et al., 2015), apesar de terem sido considerados jogos completos, a análise ficou restrita ao número de 6 jogos.

Assim sendo, foi possível auxiliar no processo de comparação dos algoritmos YBWC e APHID aplicados a agentes jogadores de Damas produzidos pela equipe pesquisadora por meio da implementação de um sistema automático que realiza a avaliação da qualidade da tomada de decisão de tais agentes a partir da plataforma *CheckerBoard* (Capítulo 3). Para tanto, o algoritmo de busca (YBWC ou APHID) dos agentes a serem avaliados - cujo papel é definir o melhor movimento que ele deve executar a partir do tabuleiro corrente - gera um arquivo no formato padrão para o jogo de Damas denominado PDN (*Portable Draughts Notation*) (Seção 3.1). Tal arquivo contém todas as decisões tomadas pelo agente ao longo de jogos reais nos quais tenha atuado. A partir deste arquivo, o sistema automático efetua uma conexão com a plataforma *CheckerBoard* de forma a comparar, estatística e automaticamente, a taxa de coincidência entre tais tomadas de decisão e as escolhas de movimento que seriam efetuadas, nas mesmas situações, pelo renomado agente de Damas *Cake* - considerado um excelente "conselheiro" de movimentos por ser um jogador de alto nível (Seção 4.1.2) e estar disponível na referida plataforma.

Dessa forma, a partir do referido sistema automático produzido nesta etapa, foi efetuada uma análise comparativa mais adequada e precisa do desempenho dos agentes automáticos em jogos reais (Capítulo 7). Tal análise foi realizada por meio de dois estudos de caso:

Estudo de Caso 1: engloba os jogadores automáticos *VisionDraughts* e *D-VisionDraughts*.

Tal estudo busca validar o uso da ferramenta em análises da qualidade de escolha de movimentos de agentes automáticos a partir de resultados obtidos em trabalhos desenvolvidos anteriormente envolvendo os referidos jogadores.

Estudo de Caso 2: comparação acerca dos mono-agentes *D-VisionDraughts* e *APHID-Draughts*. Tal estudo busca comparar, a partir de torneios, a performance dos algoritmos YBWC e APHID, de modo a corroborar a superioridade do segundo agente, o qual utiliza a abordagem assíncrona.

2 Revisão Bibliográfica

Neste capítulo, são apresentados os conceitos relevantes para a realização deste trabalho e os trabalhos correlacionados. Primeiramente, os conceitos básicos para o entendimento desta dissertação são expostos, como os algoritmos paralelos de busca YBWC e APHID, baseados no algoritmo serial Alfa-Beta, o qual também é explicado. Ressalta-se que tais algoritmos distribuídos serão brevemente explicados neste capítulo, porém detalhados e explorados no capítulo 5. Na sequência, é feita uma breve síntese do trabalho relacionado que influencia na execução do presente trabalho.

2.1 Referencial Teórico

Nesta seção, são apresentados conceitos importantes para o entendimento e realização desta monografia. Dessa forma, o algoritmo serial de busca Alfa-Beta é explicado inicialmente, e, na sequência, os algoritmos paralelos de busca YBWC e APHID são expostos.

2.1.1 Algoritmo de Busca Alfa-Beta

Um jogo de Damas pode ser visualizado como uma árvore de estados de tabuleiro. Cada nó da árvore representa um estado e cada ramo representa um movimento. Para lidar com esse tipo de jogo, o algoritmo de busca Alfa-Beta é muito utilizado. Ele consiste de uma otimização do algoritmo de busca Minimax.

O Minimax pode ser simplificado como um algoritmo recursivo que busca o melhor movimento a partir do estado de tabuleiro atual através de uma busca em profundidade da esquerda para a direita em árvores de jogo (CAIXETA; JULIA, 2008) (RUSSELL; NORVIG, 2004). Neste caso, a raiz da árvore corresponde ao estado do tabuleiro e o melhor movimento é escolhido de acordo com a avaliação calculada para os nós folhas. Por exemplo, na Figura 1 (a) o Minimax indica que o agente deve escolher o *Movimento A* a partir do estado corrente (nó raiz).

A desvantagem do Minimax é que ele examina mais nós do que o necessário. A fim de resolver este problema, o algoritmo de busca poda Alfa-Beta o otimiza ao eliminar partes da árvore que de fato não possuem o melhor movimento (CAIXETA; JULIA, 2008). Este algoritmo recebe como parâmetro o estado de tabuleiro atual, bem como dois valores: Alfa e beta. Alfa e beta delimitam o intervalo para o valor de avaliação correspondente ao melhor movimento a ser executado a partir do estado corrente do tabuleiro. A avaliação de um nó minimizador pode ser interrompida no momento em que o valor de avaliação

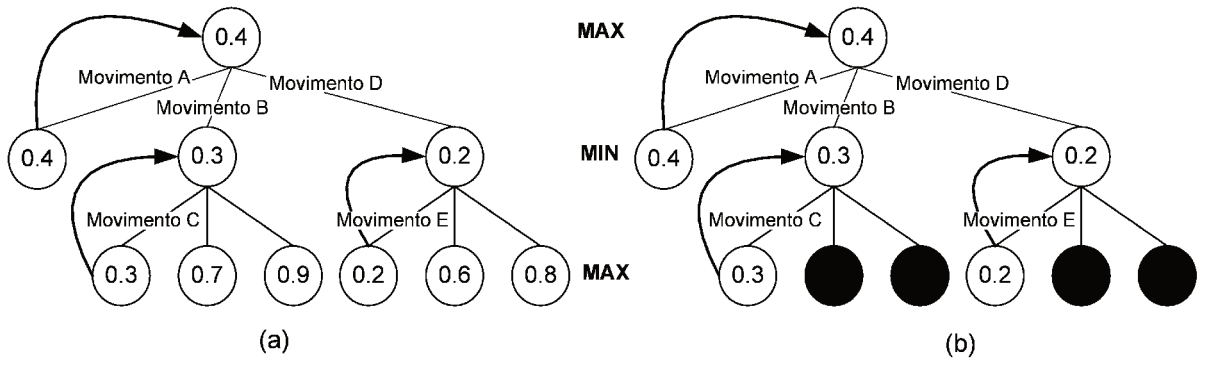


Figura 1 – Exemplo de árvore explorada pelos algoritmos (a) Minimax e (b) poda Alfa-Beta

calculado é menor que Alfa (conhecido como poda Alfa). Analogamente, a avaliação de um nó maximizador pode ser interrompida quando o valor de avaliação calculado para um filho é maior que beta (conhecido como poda beta). Por exemplo, a Figura 1 (b) apresenta a mesma solução da Figura 1 (a). Todavia, o algoritmo Alfa-Beta detecta que não é necessário avaliar os nós destacados. Desta forma, o melhor movimento (Movimento A) é encontrado mais rapidamente.

O algoritmo Alfa-Beta tem sido muito empregado na solução de jogos como Damas, todavia, nestes jogos há uma forte correlação entre a profundidade de busca que um jogador consegue atingir (visão futura da partida) e a sua força em partidas. Desta forma, devido à complexidade do problema (veja tabela 1), diversas propostas de aprimoramento do Alfa-Beta tem sido propostas na literatura com destaque à paralelização deste algoritmo. Dentre as propostas de paralelização do Alfa-Beta encontram-se o YBWC (FELDMANN et al., 1990) e o APHID (BROCKINGTON; SCHAEFFER, 2000).

2.1.2 YBWC

A busca paralela é o meio utilizado para melhorar a velocidade de busca através do uso adicional de processadores (MANOHARARAJAH, 2001). No caso do jogo de damas, o uso de mais processadores permite buscas mais profundas, e, conseqüentemente, melhora o *look-ahead* do jogador, isto é, a capacidade de alcançar níveis mais profundos na árvore de jogo.

O algoritmo YBWC (Young Brothers Wait Concept) (FELDMANN et al., 1990) é uma versão distribuída do algoritmo serial Alfa-Beta. Resumidamente, o YBWC funciona da seguinte maneira: primeiramente, expande-se o sucessor mais à esquerda, referido como irmão mais velho ou *eldest brother*, do estado atual. A ideia é tentar estreitar a janela de busca durante a expansão do irmão mais velho de tal modo que aumente o número de

podas na expansão dos irmãos mais novos, denominados *young brothers*. Este algoritmo utiliza o modelo mestre-escravo de paralelismo, no qual um processador que envia uma tarefa se torna o mestre do processador que recebe a referida tarefa. E uma vez que um nó tenha sido atribuído a um determinado processador, a responsabilidade deste nó não pode ser transferida para nenhum outro processador (BROCKINGTON; SCHAEFFER, 2000).

A tarefa de distribuição do algoritmo Alfa-Beta não é trivial, já que este depende de um conhecimento a priori proveniente de outras partes da árvore de jogo. A finalidade do uso desse conhecimento prévio é eliminar a exploração desnecessária de nós. No entanto, como esse conhecimento percorre simultaneamente ao longo de diferentes partes da árvore, em uma versão distribuída, pode acontecer de não estar sempre disponível com a mesma precisão que ocorre no algoritmo sequencial. Este problema é denominado de sobrecarga de busca.

O algoritmo YBWC tenta resolver este problema do seguinte modo: quando um nó é designado para avaliação por um processador P_j , espera o retorno da janela de busca obtida a partir da exploração do *eldest brother*. A partir daí, pode distribuir os *young brothers* para serem processados por outros escravos considerando esta janela de busca. Cada vez que o processamento de um determinado escravo causa um estreitamento na janela de busca, assim que P_j recebe essas informações, envia uma mensagem aos outros escravos comunicando os novos limites da janela de busca. Este procedimento aumenta significativamente a possibilidade de ocorrência de poda (BARCELOS; JULIA; MATIAS, 2011).

Outros problemas relacionados aos custos indiretos que podem ser mencionados na implementação desta categoria de algoritmos são: sobrecarga de comunicação associada à troca de mensagens entre processos e possíveis períodos de ociosidade, que podem ocorrer entre os processadores, além da sobrecarga de sincronização, que pode ocorrer se um processador precisa esperar por informações provenientes de outros processadores (BROCKINGTON; SCHAEFFER, 2000).

2.1.3 APHID

Em (BROCKINGTON; SCHAEFFER, 2000) foi proposto o algoritmo APHID, que é uma versão paralela assíncrona do Alfa-Beta. Tal algoritmo foi concebido com o objetivo de amenizar algumas limitações identificadas em abordagens síncronas, tais como:

1. numerosos pontos de sincronização global os quais resultam em processadores ociosos;
2. o uso necessário de tabelas de transposição compartilhadas e/ou aprofundamento iterativo para alcançar uma boa ordenação de movimentos e de performance devido

ao método de divisão de tarefas, onde um processador pode “pegar” a tarefa de outro e perder informações adicionadas previamente à tabela de transposição caso esta seja distribuída, tal fato resulta na necessidade de uma arquitetura de memória compartilhada;

3. diversos nós que seriam explorados da melhor forma sequencialmente por não demandarem muita carga de processamento são distribuídos. Por exemplo, se o primeiro ramo de um nó não causar uma poda, o YBWC permite que todos os outros nós irmãos sejam analisados em paralelo. Porém, se o segundo nó gerar uma poda, então todo o paralelismo no terceiro e nos demais nós foi desperdiçado;
4. a busca paralela em árvores de jogos normalmente requer um esforço significativo de engenharia de software para integrar um algoritmo paralelo síncrono a uma aplicação já existente.

Assim sendo, o algoritmo APHID foi projetado para lidar com os problemas citados da seguinte forma: o algoritmo é assíncrono por natureza, isto é, remove todos os pontos de sincronização globais da busca Alfa-Beta e do aprofundamento iterativo; não requer o uso de tabela de transposição compartilhada para informação de ordenação de movimentos; o paralelismo só é aplicado em nós que possuem alta probabilidade de necessitarem do mesmo, tal decisão é baseada na melhor informação disponível em um dado momento; e, finalmente, o algoritmo é projetado de modo a se enquadrar facilmente em um algoritmo de busca sequencial Alfa-Beta existente.

O algoritmo APHID adota o modelo mestre-escravo de paralelismo e funciona, de maneira geral, da seguinte forma: subdivide a árvore em vários pedaços distintos de tal modo que cada processo pode explorar independentemente dos outros processos. Um processador mestre inicia a busca expandindo a árvore de jogo da raiz até uma profundidade de busca d' . Os nós folhas dessa subárvore definem uma fronteira e serão distribuídos uniformemente entre os demais processadores, denominados escravos, que irão realizar a busca por aprofundamento iterativo até a profundidade d (limite máximo da busca), onde $d > d'$.

O processador mestre faz repetidas buscas ou passagens sobre sua árvore de jogo. A parte de cima da árvore, correspondente a $d - d'$, é pequena e rapidamente analisada pelo mestre. Este avisa aos escravos sobre as novas porções de trabalho descobertas e pode remover aquelas que não são mais relevantes para a busca. Além disso, o mestre pode transferir tais porções de um escravo a outro com a finalidade de balancear as cargas.

Cada porção de trabalho tem sua importância, a qual é repassada aos processadores escravos pelo mestre. Cada escravo tem a responsabilidade de analisar todos os pedaços de trabalho alocados na mesma profundidade. E, ao invés de esperar o mestre informar até qual profundidade cada pedaço deve ser analisado, o escravo usa aprofundamento

iterativo de modo a analisar cada porção mais profundamente até que o mestre avise-o para parar. Dessa forma, o escravo está continuamente analisando uma de suas porções de trabalho, independentemente dos outros processadores escravos. Ele informa ao mestre o valor do minimax encontrado tão logo for determinado, além de processar informações importantes enviadas pelo mestre periodicamente.

Assim sendo, o desempenho do algoritmo não depende da implementação de uma memória global compartilhada ou de uma rápida rede de interconexão entre os processadores. A comunicação entre mestre e escravo é realizada de modo assíncrono por meio de uma estrutura denominada Tabela Aphid. Cada processador escravo possui uma cópia desta tabela que é condizente com sua porção de trabalho. Apenas o processador mestre possui visão global da tabela. Desta forma, quando o mestre deseja encaminhar informações para os escravos, ele as registra nesta tabela. Os processadores escravos, por sua vez, leem sua tabela local e realizam a exploração dos nós que lhe foram atribuídos por um identificador. Ao finalizarem, gravam o resultado em sua tabela com cópia para a Tabela Aphid do mestre.

Durante este processo, o processador mestre realiza diversas passagens pela sua subárvore até a profundidade d' sempre utilizando os valores mais atuais dessa fronteira que estão gravados na sua Tabela Aphid até constatar que todos os ramos explorados na fronteira alcançaram a profundidade d . A definição da janela Alfa-beta é definida empiricamente de acordo com a natureza do problema, no entanto, utiliza uma margem que tem como ponto central o valor atual (valor hipotético) do retorno minimax da exploração feita pelo mestre em sua subárvore.

2.2 Trabalhos Correlatos

A investigação de algoritmos que adotam as abordagens síncrona e assíncrona não é realizada com muitos detalhes na literatura. Tal análise é efetuada com mais detalhes em (BROCKINGTON; SCHAEFFER, 2000), trabalho que apresentou pela primeira vez o algoritmo assíncrono APHID, mostrando a sua superioridade com relação às abordagens síncronas em termos teóricos. Destaca-se que em tal trabalho foi considerado um cenário comparativo entre o APHID e o YBWC aplicados ao jogo de Damas.

No referido trabalho, Brockington, o autor do APHID, realizou algumas comparações conceituais entre as abordagens síncronas (incluindo o YBWC) e o algoritmo APHID. Estas comparações focaram apenas em parâmetros relacionados às fontes de ineficiência de abordagens distribuídas do algoritmo alfa-beta - as principais são sobrecarga de busca, sobrecarga de sincronização e sobrecarga de comunicação 5 - e velocidade de execução do algoritmo. Os experimentos realizados foram conduzidos em arquitetura de memória compartilhada. No caso particular da comparação entre o YBWC e o APHID, o autor

utilizou o domínio do jogo Othello por meio do jogador Keyano ([BROCKINGTON, 1997](#)) como estudo de caso, no entanto, foram avaliados movimentos individuais de modo a focalizar nos parâmetros inerentes à avaliação proposta. De fato, não existe na literatura uma comparação entre os algoritmos APHID e YBWC aplicados a um mesmo problema prático. O próprio Brockington sugeriu em ([BROCKINGTON, 1998](#)) que uma comparação desta natureza seria muito interessante para colocar em perspectiva o desempenho real destes algoritmos em uma situação prática. Outros trabalhos mais recentes investiram em contribuições para o YBWC ([URA; YOKOYAMA; CHIKAYAMA, 2013](#)) e para o APHID ([URA; TSURUOKA; CHIKAYAMA, 2015](#)) ambos utilizando o domínio do jogo Soghi (xadrez japonês). Todavia, apesar de terem utilizado o mesmo domínio de jogo, não realizaram tal comparativo.

3 Plataforma CheckerBoard

A *CheckerBoard* (??) é uma plataforma internacional livre para jogos de Damas e suporta a notação PDN (*Portable Draughts Notation*). Esta plataforma disponibiliza algumas das *engines* (agentes automáticos) de maior sucesso para Damas como por exemplo *Cake* e *Kingsrow* (FIERZ, 2010), (GILBERT, 2000) e (??). A interface desta plataforma permite que jogadores humanos e automáticos possam jogar contra as *engines* disponíveis nela. Além disso, a *CheckerBoard* disponibiliza base de dados contendo milhares de jogos de competições entre os grandes jogadores de Damas, incluindo jogos entre *Marion Tinsley* e *Chinook* (SCHAEFFER et al., 1996).

Destaca-se que uma das principais funcionalidades desta plataforma é a análise completa de jogos executada pelo agente *Cake*. Neste contexto, tal agente percorre todos os movimentos realizados em uma partida armazenada em um arquivo PDN de modo a indicar quais seriam as melhores jogadas para cada estado de tabuleiro atingido. Desta forma, é possível comparar cada tomada de decisão em uma partida com a ação que o *Cake* executaria na mesma situação de jogo.

Neste trabalho, a *CheckerBoard* foi utilizada como meio para reproduzir as partidas dos agentes jogadores a serem avaliados de modo a permitir a comparação das tomadas de decisão de tais agentes com o *Cake*.

3.1 Arquivo PDN

O formato de arquivo padrão de jogos de Damas é o PDN. Independente da plataforma e/ou linguagem de programação que o agente jogador foi desenvolvido, o arquivo PDN segue a mesma estrutura. Desta forma, um arquivo PDN deve conter os seguintes atributos:

- *Event*: nome da partida;
- *Date*: data de realização da partida;
- *Black*: identificação do jogador preto;
- *White*: identificação do jogador branco;
- *Result*: resultado da partida;
- Registro de todos os movimentos realizados ordenados do início ao fim de partida.

O objetivo do arquivo PDN é criar um meio simples para armazenar todos os passos existentes em uma partida de tal modo que seja possível compartilhar um jogo completo. Neste contexto, existem diversas coleções de arquivos PDN com a finalidade de estudo e utilização no aprimoramento das habilidades de jogadores automáticos.

É importante ressaltar que várias partidas podem ser armazenadas em um único arquivo PDN, porém, neste trabalho, cada arquivo PDN representa somente uma partida.

4 Agentes Automáticos Jogadores de Damas

Este capítulo apresenta alguns dos principais agentes jogadores automáticos para Damas existentes na literatura enfocando aqueles relacionados a este trabalho. Dessa forma, os trabalhos correlacionados são divididos em dois tópicos principais: jogadores com forte supervisão humana (aqui classificados como jogadores supervisionados) e jogadores com o mínimo de intervenção humana (aqui classificados como jogadores não supervisionados).

4.1 Agentes Automáticos Jogadores de Damas: Supervisionados

Estes jogadores estão englobados no contexto da aprendizagem supervisionada, isto é, quando o processo de aprendizagem é fortemente norteado por especialistas humanos ou por bases de dados. Desta forma, dois jogadores destacam-se dos demais no cenário mundial: *Chinook* (SCHAEFFER et al., 1996) e *Cake* (FIERZ, 2010).

4.1.1 Chinook

O projeto Chinook foi iniciado em 1989 na tentativa de melhor entender as buscas heurísticas. O Chinook se tornou o mais famoso e mais forte jogador de Damas do mundo (SCHAEFFER et al., 1996). Ele é o campeão mundial homem-máquina para o jogo de Damas e conta com um conjunto de funções de avaliação ajustadas manualmente, ao longo de vários anos, para estimar quanto um determinado estado do tabuleiro é favorável para o jogador. Além disso, ele tem acesso a coleções de jogadas utilizadas por grandes mestres na fase inicial do jogo (*opening books*) e um conjunto de bases de dados para as fases finais do jogo com 39 trilhões de estados do tabuleiro (todos os estados com 10 peças ou menos) com valor teórico provado (vitória, derrota ou empate). Para escolher a melhor ação a ser executada, o jogador utiliza um procedimento de busca minimax distribuída com poda Alfa-Beta, aprofundamento iterativo e tabela de transposição.

O *Chinook* conseguiu, em 1994, o título de campeão mundial de Damas ao empatar 6 jogos com o Dr. *Marion Tinsley* que, até então, defendia seu título mundial a mais de 40 anos.

Em Damas, é fundamental evitar algumas posições no início de um jogo. Por isso a utilização de bases de dados de início de jogos (*opening books*) é bastante interessante, uma vez que um movimento inicial ruim pode fazer com que, antes do quarto movimento, seja encontrada uma situação de derrota forçada. A utilização de bases de dados de finais de jogos em Damas tem papel fundamental para a construção de agentes jogadores de alto

desempenho. No *Chinook*, essas bases contribuíram significativamente para seu sucesso contra os melhores humanos. Tais bases representam cerca de 400 bilhões de estados possíveis do tabuleiro (SCHAEFFER et al., 1996). Uma vez que as bases de dados são indispensáveis para o sucesso do jogador, um eficiente algoritmo de busca é utilizado para permitir rápida localização e execução da melhor ação.

Em 2007, sua equipe executadora provou que o jogo de Damas estava fracamente resolvido, isto é, partindo-se do tabuleiro inicial padrão do jogo de Damas, existe uma prova computacional de que o *Chinook*, no pior caso, obtém um empate no jogo (SCHAEFFER et al., 2007).

4.1.2 Cake

O *Cake* é um jogador concebido de modo supervisionado (FIERZ, 2010). Ele conta com uma base de abertura de jogos, que possui cerca de 2 milhões de movimentos por padrão e uma base de final de jogo que permite com que esse jogador tenha perfeito conhecimento do valor de qualquer estado de tabuleiro com até 8 peças. Para atuar no jogo entre as fases de início e fim de jogo (fases em que o jogador não possui acesso às bases de dados) o *Cake* utiliza o algoritmo de busca *MTD(f)*. Tal algoritmo tem capacidade de processar em média 2 milhões de posições por segundo (FIERZ, 2010). O *Cake* é conhecido por ter vencido uma versão preliminar do *Chinook* - versão esta famosa por ter derrotado o campeão mundial da época *Marion Tinsley*. Este jogador está disponível por meio da plataforma *CheckerBoard* (??).

4.2 Agentes Automáticos Jogadores de Damas: Não Supervisionados

Estes jogadores estão englobados no contexto da aprendizagem não-supervisionada, isto é, quando o processo de aprendizagem do agente é baseado essencialmente em suas próprias experiências. Desta forma, os seguintes jogadores automáticos destacados são os mais relevantes para a realização deste trabalho: *VisionDraughts* (CAIXETA, 2008); *D-VisionDraughts* (BARCELOS, 2011), (TOMAZ; JULIA; BARCELOS, 2013); *D-MA-Draughts* (TOMAZ, 2013); e, *APHID-Draughts*.

4.2.1 VisionDraughts

O *VisionDraughts* (CAIXETA, 2008) é um jogador que consiste de uma Rede Neural Multicamadas (MLP - *Multilayer Perceptron*) que aprende sem supervisão humana através dos Métodos das Diferenças Temporais $TD(\lambda)$ aliado à estratégia de treino por *self-play* com clonagem. A ideia básica desta estratégia de treino é realizar diversos jogos

contra uma cópia de si próprio de modo a ajustar os parâmetros do agente que está treinando. À medida tal agente melhora sua performance, um novo clone é produzido e o treino continua com o agente ajustado e seu clone. A representação do tabuleiro é feita através de um conjunto de funções que descrevem características inerentes ao próprio jogo de Damas. Tal representação é denominada NET-FEATUREMAP. A escolha da melhor ação a ser executada pelo jogador é conduzida pelo algoritmo de busca Alfa-Beta aliado com Tabelas de Transposição (TT), que é um repositório para armazenar estados de tabuleiro já avaliados.

4.2.2 D-VisionDraughts

O *D-VisionDraughts* (BARCELOS, 2011), (TOMAZ; JULIA; BARCELOS, 2013) estende a arquitetura do *VisionDraughts* aprimorando o processo de escolha da melhor ação. Para isso, o algoritmo de busca serial Alfa-Beta é substituído pela versão distribuída do Alfa-Beta YBWC. Em (TOMAZ; JULIA; BARCELOS, 2013), os autores realizaram uma análise comparativa entre o *D-VisionDraughts* e o *VisionDraughts* em termos de tempo de treinamento da rede jogadora e da performance dos jogadores no que se refere ao número de vitórias obtidas em torneios. Os resultados mostraram que o *D-VisionDraughts* teve tempo de treinamento 43% menor e, nos torneios, o *D-VisionDraughts* apresentou uma performance 17.5% superior ao seu oponente *VisionDraughts*. Desta forma, os resultados da literatura apontam que a utilização do algoritmo de busca distribuído fez com que o *D-VisionDraughts* se mostrasse superior ao seu predecessor *VisionDraughts*.

4.2.3 D-MA-Draughts

O *D-MA-Draughts* (*Distributed Multiagent Draughts*) (TOMAZ, 2013) é composto por 26 agentes, sendo um agente denominado IIGA (*Initial Intermediate Game Agent*) que atua na partida em fases iniciais e intermediárias (estados com no mínimo 13 peças); e 25 agentes de final de jogo, cada um denominado EGA (*End Game Agent*), que atuarão em estados de final de jogo (estados com, no máximo, 12 peças). A indicação de qual EGA é o mais apropriado para tomar decisões (escolher movimento) em etapas de final de jogo é realizada por uma rede Kohonen-SOM. Cada agente do *D-MA-Draughts* consiste de uma rede neural multicamadas (*Multilayer Perceptron - MLP*) que aprende por reforço através do método de Diferenças Temporais (DT(λ)) (SUTTON, 1988) aliado à estratégia de treino por self-play com clonagem. Com relação à busca pelo melhor movimento na árvore de jogo¹, a primeira versão do *D-MA-Draughts* é baseada no algoritmo de busca paralelo síncrono *Young Brothers Wait Concept* (YBWC) (FELDMANN et al., 1990).

¹ Em jogos de tabuleiro como Damas, a evolução do jogo pode ser representada em uma estrutura de árvore, denominada árvore de jogo. Cada nó da árvore representa um estado do tabuleiro e cada ramo representa um possível movimento possível de ser executado a partir do estado (nó) de origem.

Uma segunda versão multi-agente baseada no algoritmo paralelo assíncrono *Asynchronous Parallel Hierarchical Iterative Deepening* (APHID) (BROCKINGTON; SCHAEFFER, 2000) é proposta no escopo global no qual esse trabalho se insere. Atualmente, há uma versão mono-agente denominada *APHID-Draughts*.

4.2.4 APHID-Draughts

O mono-agente *APHID-Draughts* estende a arquitetura do *D-VisionDraughts* - também mono-agente - aperfeiçoando o mecanismo de escolha de melhor movimento. Para tanto, a versão distribuída do algoritmo Alfa-Beta denominada YBWC é substituída pela também versão distribuída APHID. No artigo intitulado “*APHID-Draughts: comparing the synchronous and asynchronous parallelism approaches for the Alpha-Beta algorithm applied to Checkers*”, aceito na *29th International Conference on Tools with Artificial Intelligence*, desenvolvido pela equipe deste trabalho, os autores efetuaram uma investigação comparativa entre o *APHID-Draughts* e o *D-VisionDraughts* com relação aos seguintes aspectos: tempo de treinamento da rede jogadora, desempenho dos jogadores no que se refere ao número de vitórias obtidas em torneios e qualidade da tomada de decisão de tos agentes. Os resultados obtidos mostraram que o *APHID-Draughts* apresentou uma performance 65% superior ao seu oponente *D-VisionDraughts*. A expressiva performance do *APHID-Draughts* corrobora o fato de que a utilização do algoritmo de busca distribuído APHID é mais vantajosa do que a versão distribuída YBWC. Tal investigação será detalhada na seção 7.2

5 Comparação entre os algoritmos YBWC e APHID

Os algoritmos YBWC e APHID buscam o melhor desempenho em problemas com elevados espaço de estados e fator de ramificação. E como visto nas subseções 2.1.2 e 2.1.3, ambos possuem distintas formas de funcionamento, de tal forma que buscam uma compensação das partes ineficientes desses algoritmos que comprometem seus desempenhos. Dessa forma, é traçada aqui uma análise comparativa conceitual mais detalhada entre ambos algoritmos de modo a expor algumas características que foram identificados pelo estudo de cada um. Destaca-se que tal estudo é uma extensão da investigação efetuada no escopo de Doutorado no qual este trabalho está inserido.

Assim sendo, antes de detalhar a análise comparativa dos referidos algoritmos, é necessário compreender algumas das fontes de ineficiência dos algoritmos paralelos (TOMAZ, 2013). Tais fontes são as seguintes:

- **Sobrecarga de busca:** refere-se aos nós explorados pelo algoritmo paralelo que não são explorados pelo algoritmo sequencial. Esta sobrecarga ocorre primariamente pela carência de informação pelos processadores. Em algoritmos do tipo *branch-and-bound* como o Alfa-Beta, a informação obtida na busca de um ramo da árvore pode causar uma poda em outros ramos. Assim, se os sucessores de um nó são expandidos paralelamente, um nó pode não tirar vantagem de toda informação que estaria disponível na busca serial, resultando em busca desperdiçada.
- **Sobrecarga de sincronização:** refere-se à ociosidade de um processador o qual é forçado a esperar por informações providas de outros processadores. Existem estratégias para reduzir tanto a sobrecarga de busca quanto a de sincronização. Porém, é um problema difícil, pois há uma estreita ligação entre as duas sobrecargas e compensações devem ser feitas com cuidado.
- **Sobrecarga de comunicação:** refere-se à troca de informações (mensagens) entre os processadores e geralmente envolve o envio e recebimento de resultados entre processadores. Por conveniência, tudo o que não é atribuído às sobrecargas de busca ou sincronização é definido como sobrecarga de comunicação.

Assim sendo, com estes conceitos clarificados, pode-se obter um melhor entendimento da comparação conceitual referentes aos algoritmos YBWC e APHID a partir das seguintes características definidas na seção 1.2.1: comunicação entre os processadores; sincronização e/ou distribuição de tarefas; formação e/ou atualização da janela de busca

Alfa-beta; utilização de tabelas de transposição em arquiteturas de memória distribuída; e, portabilidade.

5.1 Comunicação entre os processadores

As seções 5.1.1 e 5.1.2 apresentam as mensagens com suas respectivas finalidades e o funcionamento da troca de informações entre os processadores nos algoritmos YBWC e APHID, respectivamente.

5.1.1 Comunicação no YBWC

No algoritmo YBWC a troca de informações entre os processadores é realizada de forma síncrona. Isto é, um determinado processador P_i , o qual está explorando um nó N_x e possui tarefas relacionadas a este nó distribuídas entre processadores escravos, somente finalizará a exploração de N_x quando todos os escravos enviarem suas respostas. Devido a tal fato, neste modelo de comunicação a troca de mensagens apresenta um volume muito elevado, o que eleva a sobrecarga de comunicação. As mensagens no YBWC são identificadas do seguinte modo:

- **Início da Busca:** mensagem enviada do processador principal aos demais a fim de informar o início da busca, de modo que estes inicializem suas requisições de tarefas.
- **Fim da Busca:** mensagem enviada do processador principal aos demais a fim de notificar o fim da busca, informando que a solução final foi encontrada e que as requisições de tarefas sejam cessadas.
- **Requisição de tarefa:** mensagem enviada de um determinado processador, denominado A, a outro processador, denominado B, com a finalidade de requisitar uma tarefa.
- **Resposta à Requisição de Tarefa - Retorno Positivo:** mensagem enviada por um processador que recebeu uma requisição de tarefa. Tal mensagem contém a informação de retorno positivo, ou seja, que existe alguma tarefa disponível para ser repassada. Dessa forma, a tarefa compartilhada é enviada como resposta.
- **Resposta à Requisição de Tarefa - Retorno Negativo:** mensagem enviada por um processador que recebeu uma requisição de tarefa. Tal mensagem informa que não há tarefas disponíveis no momento da requisição - retorno negativo.
- **Resultado de Avaliação:** mensagem que contém o retorno da avaliação da tarefa explorada pelo processador em questão.

- **Nova Janela de Busca:** mensagem que possui a finalidade de informar os novos valores dos limites de busca identificados por algum processador durante o processo de busca.
- **Poda:** mensagem enviada a algum processador escravo quando é identificado que a tarefa de sua responsabilidade é irrelevante à solução final da busca, desta forma, ela deve ser interrompida imediatamente.

As Figuras 2(a) e 2(b) ilustram os possíveis estados do processador principal e dos processadores auxiliares, respectivamente, considerando a arquitetura do jogador automático *D-Vision Draughts*. De modo a exemplificar a relação entre os estados dos processadores e as mensagens trocadas entre eles, o seguinte cenário será considerado:

- Conjunto de processadores $P = \{P0, P1, P2, \dots, P16\}$;
- O processador $P0$ é denominado Processador Principal;
- Os processadores restantes P_i ($1 \leq i \leq 16$) são denominados Processadores Auxiliares;

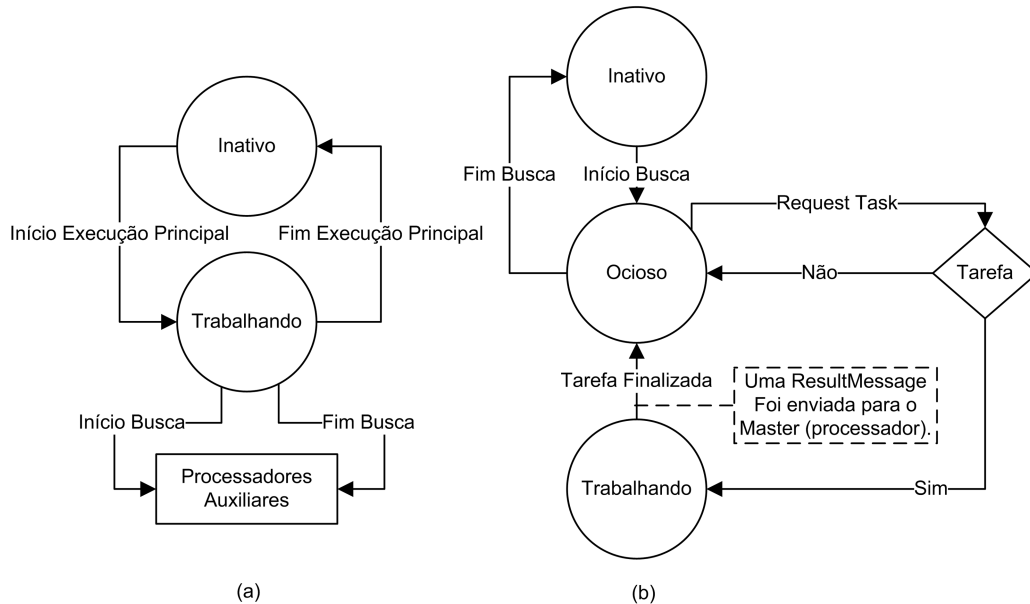


Figura 2 – (a) Estados do Processador Principal; (b) Estados dos Processadores Auxiliares

O Processador Principal $P0$ atua em todos os módulos de um agente jogador, enquanto que os Processadores Auxiliares atuam exclusivamente no módulo de busca de um jogador. Assim sendo, o processador $P0$ permanece *Inativo* até que a partida seja iniciada. Além disso, enquanto $P0$ realiza operações de rotina fora do escopo de busca, os

Processadores Auxiliares ficam no estado *Inativo*. Tão logo $P0$ envia a mensagem *Início da Busca* aos demais processadores, estes mudam o seus estados de *Inativo* para *Ocioso* - não possuem nenhuma tarefa alocada, porém podem requisitar tarefas a partir deste momento.

Na etapa de busca, todos os processadores do conjunto P podem requisitar tarefas, o que significa que um mesmo processador pode ser, ao mesmo tempo, tanto mestre quanto escravo de outros processadores. Dessa forma, quando um processador *Ocioso* P_j envia uma mensagem do tipo *Requisição de tarefa* a outro processador P_k , dois cenários podem ocorrer:

1. O processador P_k não possui nenhum nó (tarefa) disponível e responde com a mensagem *Resposta à Requisição de Tarefa - Retorno Negativo*. Assim, P_j continua no estado *Ocioso*.
2. P_k possui um nó (tarefa) disponível para repassar ao processador P_j e envia como resposta a mensagem *Resposta à Requisição de Tarefa - Retorno Positivo*. Dessa forma, P_j agora possui uma tarefa para processar e, então, muda seu estado de *Ocioso* para *Trabalhando*. Além disso, P_k torna-se mestre de P_j .

Quando um processador escravo conclui o processamento de uma tarefa que lhe foi repassada, ele envia o resultado obtido para o seu mestre (por meio da mensagem *Resultado de Avaliação*) e volta a ficar *Ocioso*. Quando todo o processo da etapa de busca é concluído, $P0$ envia a mensagem *Fim de Busca* para todos os processadores auxiliares, os quais voltam ao estado de inatividade. Deste ponto em diante, apenas o processador principal estará no estado *Trabalhando* e permanecerá em tal estado durante uma partida. Assim, $P0$ volta ao estado *Inativo* apenas quando a partida é encerrada.

5.1.2 Comunicação no APHID

Já no algoritmo APHID, a comunicação entre os processadores é feita de forma assíncrona, isto é, quando um processador mestre P_i recebe um nó N_x para exploração, ao distribuir as subárvores de N_x aos processadores escravos, P_i não fica esperando as respostas dos escravos para dar continuidade à busca. Ele mantém o gerenciamento da busca atualizando de acordo com as respostas que chegam dos escravos, de modo a evitar períodos de ociosidade dos processadores. Além disso, mantém apenas um ponto de comunicação que é estabelecido pela fronteira da subárvore do mestre com o objetivo de diminuir a sobrecarga de comunicação. Dessa forma, há menos tipos mensagens trafegando na rede de interconexão entre os processadores, concentrando a comunicação na estrutura Tabela Aphid, detalhada em 5.1.2.1. Neste contexto, o APHID trabalha com as seguintes mensagens:

- **Envio de tarefa:** o processador mestre grava as tarefas na estrutura Tabela Aphid de cada processador escravo.
- **Valor hipotético:** o processador mestre envia o valor mais recente obtido da exploração de sua subárvore para formação da janela de busca nos escravos.
- **Resposta de tarefa:** os processadores escravos gravam na sua porção da estrutura Tabela Aphid do mestre o resultado de suas explorações.
- **Finalização da Busca:** o processador mestre envia uma mensagem a todos os escravos informando que a busca principal finalizou. É importante ressaltar que esta é a única mensagem em que não é utilizada a Tabela Aphid como intermediária de comunicação.

As Figuras 3(a) e 3(b) ilustram os possíveis estados de um processador mestre e de um processador escravo, respectivamente, considerando a arquitetura do jogador automático *APHID-Draughts*. Diferentemente do YBWC, cuja natureza síncrona permite que os processadores desempenhem papéis de mestre e escravo simultaneamente, no APHID cada processador assume o papel de mestre ou escravo, mas não ambos.

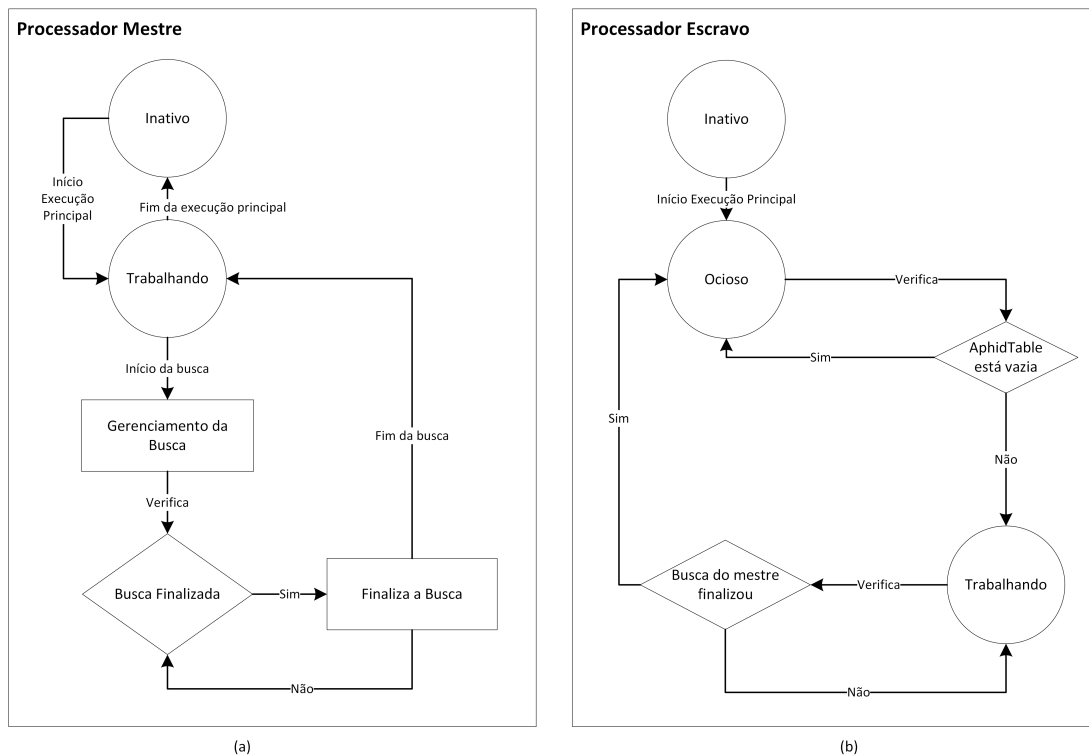


Figura 3 – (a) Estados do Processador Mestre; (b) Estados do Processador Escravo

O processador mestre estará no estado *Inativo* até que a partida seja iniciada. Portanto, quando isto ocorre, o mestre passará ao estado *Trabalhando*, pois ele atua em todos os módulos da arquitetura do jogador. Sempre que for necessária uma tomada de

decisão para indicar uma ação (movimento), ou seja, quando o módulo de busca é ativado, o mestre iniciará o processo de *Gerenciamento da Busca* (detalhado na seção 5.2.2.1). O mestre permanecerá neste processo de gestão até que a busca finalize. Neste caso, ele notificará os processadores escravos sobre o encerramento da busca e, na sequência, voltará a atuar nos demais módulos do *APHID-Draughts*. Desta forma, tal processador apenas finalizará suas atividades quando o sistema *APHID-Draughts* for encerrado, neste caso, ele retornará ao estado *Inativo*.

Um processador escravo também é iniciado junto à execução do sistema *APHID-Draughts*. Todavia, ele assume o estado *Ocioso*, uma vez que não há tarefas atribuídas a ele. Apesar de estar no estado *Ocioso*, o escravo verifica constantemente a existência de tarefas em sua estrutura Tabela Aphid (canal de comunicação entre mestre e escravo, conforme explanado na seção 5.1.2.1). Neste contexto, se a Tabela Aphid estiver vazia, o escravo permanece no estado *Ocioso*. Por outro lado, se for constatado a existência de tarefas, ele passará ao estado *Trabalhando* e irá executar todas as atividades descritas na seção 5.2.2.2 até que seja notificado pelo mestre para encerrar suas atividades, pois a busca principal finalizou. Neste caso, o processador limpa todos os recursos utilizados até o momento e retorna ao estado *Ocioso* para aguardar novas informações que ocorrerão quando o mestre iniciar um novo processo de tomada de decisão.

Finalmente, o algoritmo APHID tem a finalidade de manter um menor número de troca de mensagens entre os processadores, de modo que a comunicação entre eles esteja concentrada nas operações sobre a Tabela Aphid 5.1.2.1. Desta forma, tal estrutura é o que garante a natureza assíncrona do APHID, minimizando os efeitos das sobrecargas de comunicação e de sincronização. Isto permite que os processadores trabalhem com informações mais flexíveis e independentes entre si, ou seja, sem sincronização. Tal fato faz com que os processadores realizem algum trabalho na maior parte do tempo, evitando longos períodos de ociosidade (ao contrário do que ocorre no YBWC, em que há longos períodos nos quais os processadores ficam ociosos). Por outro lado, como os processadores efetuam explorações continuamente, a probabilidade de explorar alguma tarefa sem importância nenhuma para a busca aumenta e, conseqüentemente, a sobrecarga de busca também aumenta.

5.1.2.1 Tabela Aphid

A comunicação entre mestre e escravos no algoritmo APHID é feita a partir da estrutura Tabela Aphid, a qual contém as seguintes informações relevantes para o referido contexto:

- **Identificador:** define o processador que a estrutura está representando;
- **Tarefas:** armazena o conjunto de tarefas atribuídas pelo mestre;

- **Valor Hipotético:** armazena o valor obtido pela última passagem do mestre por sua subárvore;
- **Escravos:** referencia a Tabela Aphid de cada um dos escravos. É importante destacar que esta estrutura permite a criação de uma hierarquia entre os processadores. Todavia, na versão do APHID implementado no *APHID-Draughts*, foi considerado um mestre e os demais processadores escravos.

Cada escravo terá a cópia da Tabela Aphid a qual é responsável. Neste contexto, quando o processador mestre P_0 desejar se comunicar com um determinado escravo P_i , ele escreverá na porção da Tabela Aphid referente a este último processador e gravará uma cópia em sua Tabela Aphid local. É função de P_i verificar constantemente sua Tabela Aphid a fim de identificar a existência de tarefas a serem executadas. Sempre que P_i quiser retornar informações de um nó ao mestre ele o fará por meio da Tabela Aphid. Desta forma, não há ligação direta entre mestre e escravos. Assim, esta estrutura permite a comunicação assíncrona entre os processadores, o que minimiza significativamente a sobrecarga de sincronização e comunicação do APHID com relação ao YBWC.

5.2 Distribuição e Sincronização de Tarefas

Os algoritmos YBWC e APHID possuem estratégias muito distintas quanto à distribuição e sincronização de tarefas entre os processadores, conforme mencionado na seção 1.2.1.

No primeiro algoritmo, os processadores podem requisitar tarefas a partir do momento em que a exploração do *eldest brother* é finalizada. Desta forma, há uma avaliação de nós que poderiam ser processados localmente por não demandarem muito esforço, diminuindo a sobrecarga de comunicação. Por outro lado, como a exploração dos *young brothers* pode ser paralelizada somente após a avaliação do *eldest brother*, há vários momentos em que os processadores em espera de tal avaliação tornam-se ociosos. Tal fato configura um ponto de sincronização. Conforme a árvore de busca aumenta, o número de pontos de sincronização também aumentam, o que eleva a sobrecarga de sincronização do YBWC. Já no segundo algoritmo, pelo fato de definir uma profundidade inicial d' para a exploração da subárvore do mestre, evita-se que nós muito rasos sejam distribuídos. Além disso, o processador mestre pode identificar no decorrer da busca nós que não precisam ser distribuídos, isto é, nós que ultrapassem o limite d' , mas que não demandam muito processamento. Nestes casos, tais nós são explorados pelo próprio mestre. A Figura 4 ilustra a localização dos pontos de sincronização de tarefas nos referidos algoritmos de busca.

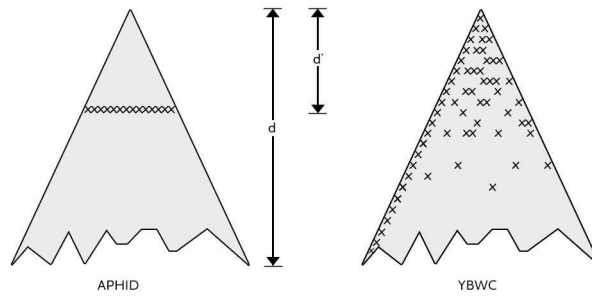


Figura 4 – Localização dos pontos de sincronização nos algoritmos APHID e YBWC (BROCKINGTON; SCHAEFFER, 2000)

5.2.1 Distribuição e sincronização de tarefas no YBWC

Conforme mencionado anteriormente e ilustrado na Figura 4, o YBWC estabelece diversos pontos de sincronização. Assim sendo, as Figuras 5(a) e 5(b) ilustram a árvore de busca do nó $N1$ utilizando o YBWC e a estrutura de controle de nós do processador $P0$, respectivamente. Tais figuras serão utilizadas para exemplificar a distribuição e sincronização de tarefas no YBWC considerando o seguinte cenário:

- Conjunto de processadores P , tal que $P = \{P0, P1, P2, P3\}$;
- $P0$ representa o processador principal. Dessa forma, é o único processador que não possui mestre;
- O nó $N1$ deve ser explorado pelo algoritmo até a profundidade máxima de busca d .

No início da busca, apenas o nó $N1$ está disponível para ser explorado, pois refere-se à raiz da árvore. $P0$ inicia a busca explorando $N1$ e o registra em sua pilha local. Na sequência, ele gera os filhos deste nó e explora o filho mais à esquerda (*eldest brother*), $N2$. De forma análoga, o processador principal continua a gerar os filhos dos nós mais à esquerda da árvore até alcançar um nó folha (estabelecido pela profundidade máxima de busca d), ou seja, tal processo termina somente quando $P0$ alcança o nó $Ni1$. Ressalta-se que todos os nós gerados também são armazenados na pilha local de $P0$. Ao final deste processo, o paralelismo é permitido nos nós que foram armazenados na pilha local do processador principal.

Dessa forma, caso algum dos outros processadores - $P1, P2, P3$ - tenha enviado uma mensagem de requisição de tarefas para $P0$, este efetua a escolha do nó localizado mais à esquerda e na maior profundidade de sua pilha local que esteja disponível para exploração. Como um dos fatores mais relevantes para distribuição de tarefas no YBWC é a profundidade, a estratégia de armazenar os nós na estrutura de pilha de acordo com suas respectivas profundidades torna-se interessante, conforme ilustrado na Figura 5(b).

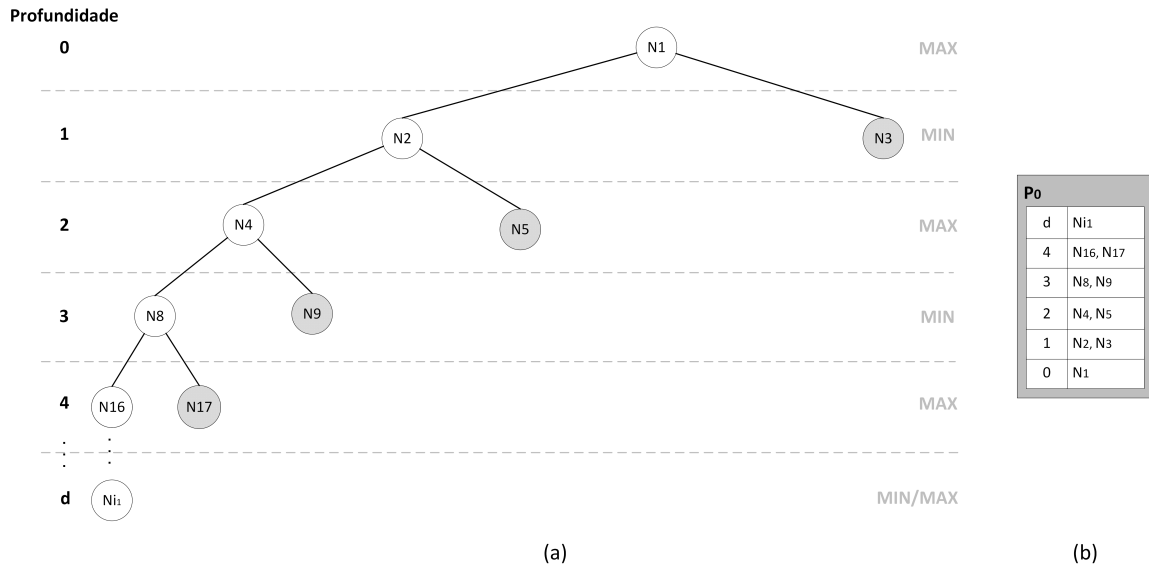


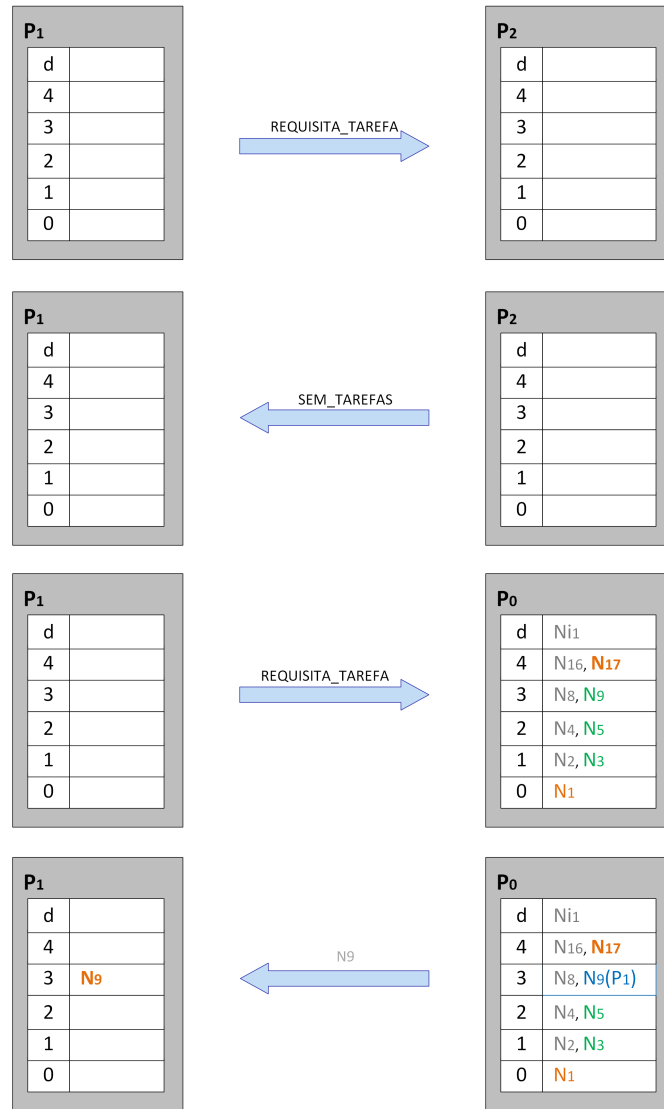
Figura 5 – (a) Árvore de Busca de exemplo (b) Ilustração da estrutura de controle de nós (pilha) do processador $P0$ na expansão do nó $N1$

No YBWC, um processador que deseja solicitar uma tarefa, seleciona algum dos processadores presentes no conjunto P e envia uma mensagem de requisição de tarefa. O processador que receberá esta requisição é de responsabilidade da política de seleção, definida na implementação do algoritmo. Por exemplo, em (BARCELOS; JULIA; MATIAS, 2011) é realizada uma política de escolha aleatória.

Assim, um processador que recebe uma mensagem de requisição de tarefas verifica, em sua pilha local, se há alguma tarefa disponível. Caso negativo, é enviada uma mensagem informando que não há tarefas disponíveis. Caso positivo, a tarefa é enviada como resposta. A Figura 6 ilustra a visão geral deste processo após a exploração do nó $Ni1$ realizada por $P0$.

Na figura 6, inicialmente, o processador $P1$ envia uma mensagem de requisição de tarefas a $P2$. Como este não possui nenhuma tarefa para distribuir, retorna uma mensagem informando que não há tarefas disponíveis ao processador requisitante. Neste cenário, $P1$ envia uma nova solicitação de tarefas, desta vez para $P0$. Este último verifica em sua pilha local se há tarefas disponíveis para serem distribuídas. Os nós representados pela cor *verde* estão disponíveis, ao passo que aqueles representados pela cor *laranja* e pela cor *cinza* estão em processo de exploração e já tiveram sua exploração concluída, respectivamente. Neste sentido, considerando que a escolha do nó a ser distribuído deve ser realizada pela localização mais à esquerda e na maior profundidade, $P0$ envia o nó $N9$ como resposta a $P1$, o qual inicia a exploração de tal nó.

No YBWC, um processador responsável por uma determinada tarefa t , não pode redistribuí-la a nenhum outro processador. Neste sentido, a tarefa $N9$ atribuída a $P1$,

Figura 6 – Solicitações de tarefas efetuadas pelo processador $P1$

não pode ser transferida, sendo $P1$ responsável por ela até o fim. Assim sendo, apenas as sub-tarefas de $N9$ podem ser distribuídas com base na mesma dinâmica realizada por $P0$ ao iniciar a exploração de $N1$. Devido ao fato de que, no YBWC, um processador estará apto a requisitar novas tarefas apenas após a conclusão da tarefa pela qual está responsável, um processador pode ficar ocioso aguardando por retornos vindos de seus processadores escravos. Por exemplo, na situação ilustrada na Figura 7, $P0$ mesmo tendo concluído a exploração de $N17$, manter-se-á ocioso até que seus escravos $P1$, $P2$ e $P3$ lhes enviem os resultados do processamento de $N9$, $N5$ e $N3$, respectivamente, uma vez que esses últimos nós integram o processamento do nó $N1$ alocado a $P0$. Tal ócio se deve à natureza intrinsecamente síncrona do YBWC.

Portanto, o YBWC permite a paralelização de tarefas a partir do momento que a exploração do irmão mais velho é finalizada, como já mencionado. Isto é realizado pelo fato de que, como a árvore de jogo analisada é ordenada no modo *best-to-worst*, há uma maior

P ₀		P ₁		P ₂		P ₃	
d	N ₁	d		d		d	
4	N ₁₆ , N ₁₇	4		4		4	
3	N ₈ , N ₉ (P ₁)	3	N ₉	3		3	
2	N ₄ , N ₅ (P ₂)	2		2	N ₅	2	
1	N ₂ , N ₃ (P ₃)	1		1		1	N ₃
0	N ₁	0		0		0	

Figura 7 – Possível distribuição de tarefas em um determinado momento a partir do YBWC

probabilidade de que o ramo mais à esquerda gere uma poda e melhore a janela de busca dos irmãos mais novos. Porém, esse processo contribui para a grande quantidade de pontos globais de sincronização existentes no algoritmo, nos quais os processadores que ficam a espera da exploração do nó mais à esquerda tornam-se ociosos. Tal fato é determinante para o aumento da sobrecarga de sincronização, uma vez que aumenta a dependência de um processador efetuar a análise de uma tarefa a partir dos resultados obtidos por outro processador. Por outro lado, tarefas pequenas que não demandam muito esforço não são paralelizadas, ou seja, são avaliadas localmente, o que minimiza um pouco a sobrecarga de comunicação, já que há uma constante troca de mensagens entre os processadores.

5.2.2 Distribuição e sincronização de tarefas no APHID

Diferentemente da maneira síncrona com que os processadores trabalham no YBWC, o APHID tenta eliminar os pontos de sincronização durante a execução do algoritmo a fim de não haver períodos de ociosidade entre os escravos. Desse modo, em um conjunto P de processadores, apenas um atuará como mestre e os demais atuarão como escravos, de tal forma que o mestre é responsável por gerenciar a busca alocando tarefas para os escravos e retornar a solução final, enquanto que os escravos processam os nós que lhes foram distribuídos pelo mestre baseado na estratégia de busca com aprofundamento iterativo, *Iterative Deepening*¹, (a partir deste momento referenciada por ID).

A fim de auxiliar na compreensão das características dos processadores que atuam no APHID e no modo com que as tarefas são distribuídas entre eles, as subseções 5.2.2.1 e 5.2.2.2 detalharão as atividades executadas pelo processador mestre e pelos processadores escravos, respectivamente. Além disso, o seguinte cenário será considerado para exemplificar a distribuição de tarefas no APHID:

- Existe um conjunto P de processadores, tal que $P = \{P_0, P_1, P_2, P_3\}$;

¹ A ideia básica do aprofundamento iterativo é realizar uma série de buscas, em profundidade, independentes, cada uma com um *look-ahead* acrescido de um nível, de modo a garantir que seja encontrado o caminho mais curto para a solução justamente como a busca em largura encontraria (RUSSELL; NORVIG, 2004).

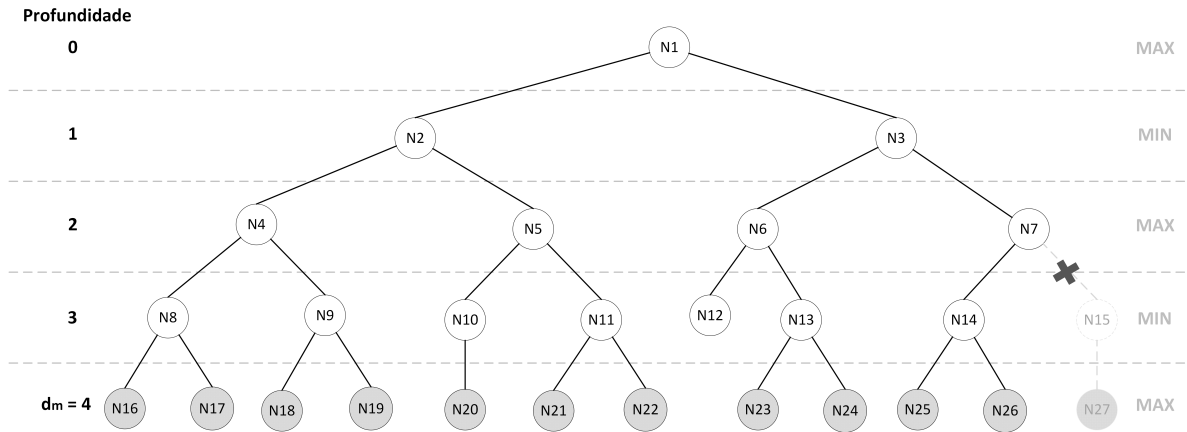


Figura 8 – Exemplo da árvore explorada pelo processador mestre no APHID

- O processador $P0$ é o mestre e os demais são os escravos;
- É apresentado ao algoritmo de busca um estado denominado $N1$, que deve ser explorado até a profundidade máxima d .

5.2.2.1 Atividades do Processador Mestre

A árvore de busca ilustrada na Figura 8, apresenta um exemplo da exploração de $N1$ por $P0$.

Como é possível observar, o processador mestre explorará a árvore de busca a partir do estado $N1$ segundo o algoritmo Alfa-beta serial *fail soft* (BROCKINGTON, 1998) até a profundidade cujos estados estão em destaque, dm , a qual representa a fronteira de exploração de $P0$. No esquema, observa-se que o nó $N15$ foi podado na busca. Além disso, ressalta-se que $dm < d$ e que, na Figura 8, $dm=4$. A partir desta profundidade, as tarefas serão distribuídas aos demais processadores, o que os torna escravos de $P0$.

A exploração compreendida entre a raiz da árvore de busca e dm é denominada *passagem* do mestre por sua subárvore (a partir deste ponto, referida, simplesmente por *passagem*). O processador mestre realizará diversas *passagens* a fim de atualizar as informações inerentes à fronteira. Este processo é executado até a busca ser concluída. Saliente-se que a profundidade dm é definida empiricamente, todavia, deve possuir um valor que não comprometa a performance do algoritmo, visto que o mestre realizará o processo de *passagem* constantemente.

Dessa forma, como a parte de cima da árvore de jogo na qual o processador mestre realiza as várias *passagens* é relativamente pequena em comparação à árvore de jogo como um todo até a profundidade máxima de busca d , ela é rapidamente analisada por $P0$. Este tem a tarefa de informar aos processadores escravos sobre novos pedaços de trabalho descobertos e desconsiderar (remover) àqueles que não são mais relevantes para a busca. Além disso, ele também pode transferir tais pedaços de um escravo a outro. É

importante destacar que tal situação ocorre exclusivamente quando é necessário realizar balanceamento de cargas entre os processadores escravos.

A distribuição de tarefas entre os escravos segue o método *round-robin* (cada folha é atribuída, sequencialmente, a um dos processadores). A Tabela 2 apresenta o resultado desta distribuição entre $P1$, $P2$ e $P3$.

Processador	Tarefas
$P1$	$N16, N19, N22, N25$
$P2$	$N17, N20, N23, N26$
$P3$	$N18, N21, N24$

Tabela 2 – Divisão de tarefas segundo o modo round-robin entre três processadores

Conforme mencionado na seção 5.1, a comunicação entre o mestre e os escravos é estabelecida por meio de uma estrutura de dados denominada Tabela Aphid disponível em cada processador. Como mostrado na seção 5.1.2.1, tal estrutura armazena as seguintes informações: o identificador do processador; a lista de tarefas (nós/estados) atribuídas a ele; o valor hipotético (valor minimax k produzido pelo mestre a partir da exploração de sua subárvore em sua última passagem); e, referência da Tabela Aphid de cada escravo. Desta forma, apesar de cada processador possuir sua própria Tabela Aphid, apenas $P0$ detém uma cópia completa das tabelas dos escravos. A Figura 9 ilustra as estruturas denominadas Tabelas Aphid do mestre $P0$ e dos escravos. Salienta-se que a aparição das tarefas nas Tabelas Aphid segue a ordem decrescente de prioridade das mesmas. Por exemplo, a tarefa $N16$ tem maior prioridade para $P1$, enquanto que $N17$ e $N18$ são as mais relevantes para o processo de busca de $P2$ e $P3$, respectivamente, como será explicado na seção 5.2.2.2.

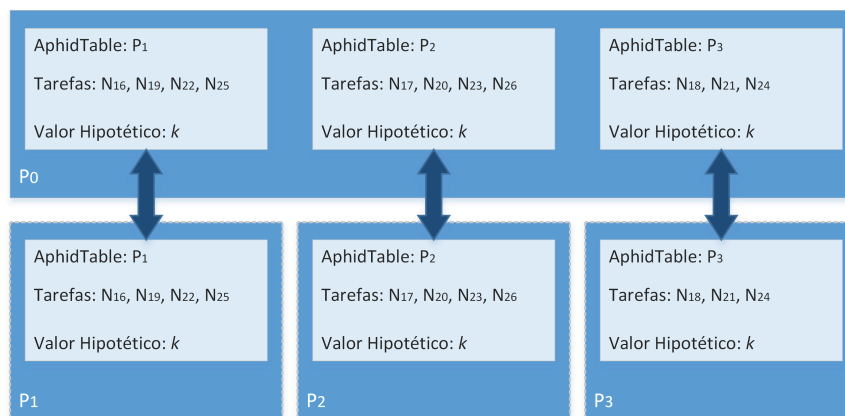


Figura 9 – Exemplo da estrutura Tabela Aphid no processador mestre ($P0$) e escravos ($P1$, $P2$ e $P3$)

Resumidamente, o processador mestre realiza as seguintes cinco etapas em cada *passagem*:

1. Execução de uma busca da raiz até a profundidade dm da árvore segundo o algoritmo Alfa-beta serial;
2. Caso seja encontrado um valor preciso de minimax da profundidade máxima d durante a etapa anterior, então finalizar o processo de busca;
3. Atualização da fronteira de busca e, conseqüentemente, das listas de trabalho;
4. Notificação aos escravos acerca das alterações realizadas em suas listas de trabalho;
5. Espera de uma nova informação provinda de um escravo antes de voltar ao primeiro passo, isto é, de iniciar uma nova *passagem*.

5.2.2.2 Atividades dos Processadores Escravos

No início do processo de busca, os processadores escravos aguardam o registro de tarefas em suas respectivas Tabelas Aphid. Logo que uma tarefa é registrada, eles iniciam suas atividades. Conforme apresentado na Tabela 2, cada escravo pode receber uma lista de tarefas. Desta forma, ele deve escolher uma tarefa para iniciar o processo de busca. A escolha de um nó em relação aos demais é baseado nos seguintes critérios de prioridade:

1. *Profundidade de busca*: os nós localizados em profundidade mais rasa tem maior prioridade, pois representam mais trabalho a ser realizado (lembrando que as profundidades de exploração dos nós da Tabela Aphid vão variando ao longo da estratégia ID).
2. *Localização do nó na subárvore de busca do mestre*: como no APHID os filhos dos nós são ordenados do modo *best-to-worst*, explorar os nós localizados mais a esquerda pode aumentar o número de podas, beneficiando o processo de busca. A localização de um nó é informada pelo mestre.
3. *Participação na fronteira de busca do mestre*: a cada passagem do mestre por sua subárvore, a fronteira de busca pode sofrer alguma alteração. Desta forma, novos nós recebem prioridade maior devido ao critério 1. Por outro lado, os nós que não apareceram na passagem corrente do mestre por sua subárvore recebem prioridade zero.

Considerando o processador escravo $P1$, sua lista de tarefas ($N16$, $N19$, $N22$, $N25$) e partindo do princípio de que estas tarefas acabaram de ser repassadas a $P1$ por $P0$, todas possuem a mesma profundidade de busca, logo, o critério 1 de prioridade não classifica tarefa alguma em relação às demais. Além disso, como tais tarefas foram obtidas pela primeira passagem do mestre, todas estão presentes na fronteira de $P0$ portanto, o critério 3 também não pode ser aplicado. No entanto, a localização destes nós na árvore de busca

é diferente e o critério 2 pode ser empregado. Assim, o nó localizado mais a esquerda corresponde à tarefa *N16*, portanto, ela será selecionada para *P1* iniciar suas atividades. Similarmente, ocorre o mesmo considerando *P2* e *P3* e suas listas de tarefas (*N17*, *N20*, *N23*, *N26*) e (*N18*, *N21*, *N24*), respectivamente. O nó *N17* será selecionado para *P2* iniciar sua busca, enquanto *N18* será escolhido para *P3* começar suas atividades. Quando nenhum dos critérios classifica as tarefas, a primeira da lista é a escolhida. É importante ressaltar que a prioridade das tarefas atribuídas aos processadores pode sofrer alterações no decorrer da busca, em virtude das novas passagens do mestre por sua subárvore.

O processador escravo *P1* inicia a exploração de sua tarefa, o nó *N16*, segundo o algoritmo Alfa-beta serial aliado à estratégia de busca por ID. Desta forma, *P1* realiza a exploração de todos os nós de *N16* na profundidade $dm + i$ inicialmente, onde i equivale ao valor de incremento de cada iteração da busca. Considerando $i = 1$, *P1* explorará *N16* na profundidade $dm = 5$. A Figura 10 ilustra o processo de expansão da subárvore de *N16* da profundidade 5 (Figura 10(A)) até d (Figura 10 (D)).

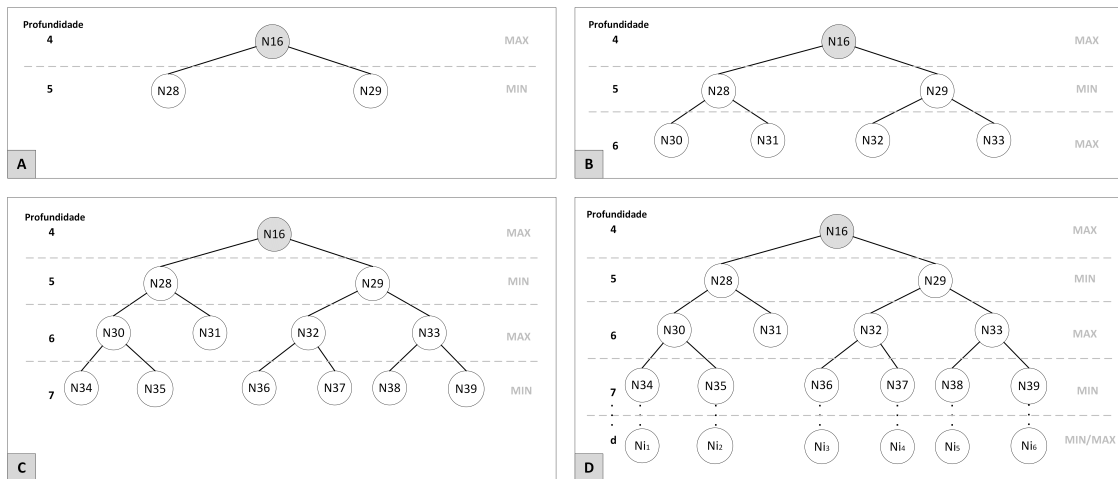


Figura 10 – Exemplo da exploração do nó *N16* no processador escravo *P1* por ID considerando o valor do incremento igual a 1

Assim que *P1* finaliza a exploração do nó *N16* na profundidade 5, o novo valor obtido será enviado ao processador mestre (*P0*) para atualização de sua fronteira de busca. Além disso, *N16* terá sua prioridade alterada na Tabela Aphid de *P1* em função da nova profundidade 6 alcançada (10(B)). Assim sendo, pode ser que *P1* não retome a exploração de *N16* após concluir sua expansão na profundidade 5, pois, como teve sua prioridade alterada, há a possibilidade de que algum outro nó tenha uma prioridade maior naquele momento - ressalta-se que os escravos sempre selecionam os nós para realizar a exploração de maior prioridade em suas respectivas tabelas. Em resumo, a profundidade de busca adotada para o nó selecionado corresponde à última exploração desta tarefa acrescida do valor do incremento de cada iteração da busca. Esse procedimento irá se repetir até que todas as tarefas sejam exploradas na profundidade máxima d , conforme ilustrado na

Figura 10.

Como já mencionado na subseção 5.2.2.1, quando uma tarefa é atribuída a um escravo ela irá permanecer com ele até o final da busca, exceto no caso de balanceamento de carga, o qual é gerenciado pelo processador mestre. Neste caso, $P0$ verifica se algum processador escravo já finalizou o processamento de todas as suas tarefas na profundidade d . Caso positivo, ele examinará, ordenadamente dentre os demais escravos, o primeiro que está sobrecarregado, ou seja, que possui pelo menos um nó em sua Tabela Aphid o qual se encontra em "compasso de espera". Tão logo esse primeiro escravo sobrecarregado é encontrado, sua tarefa em "compasso de espera" de maior prioridade é repassada ao escravo ocioso. Por exemplo, considerando a árvore de exploração de $N1$ (Figura 8), caso $P2$ esteja explorando o nó $N17$ na profundidade $d' = 6$ e $P3$ tenha finalizado a exploração dos nós que compreendem sua lista de tarefas ($N18$, $N21$ e $N24$) na profundidade máxima de busca d , o mestre irá atribuir a $P3$, a tarefa não finalizada de $P2$ com maior prioridade, neste caso, $N20$. Desta forma, $N20$ passa a pertencer a $P3$ e, caso tal nó já tenha sido explorado em profundidades mais rasas, tais informações serão descartadas e ele será reexplorado. Salienta-se que, no APHID, um nó pode ser redistribuído por balanceamento de carga uma única vez, isto é, como $N20$ foi realocado a $P3$, ele não poderá ser redistribuído novamente durante o processo de busca. Tal fato ocorre por dois motivos:

1. Para não sobrecarregar a rede com o tráfego de informações referentes a realocações constantes. Isto é, de forma a evitar sobrecarga de comunicação;
2. Para que o desempenho do algoritmo não seja prejudicado com o reprocessamento destes nós. Tal fato minimiza o efeito que o balanceamento de carga causa no que diz respeito à sobrecarga de busca.

Resumidamente, os processadores escravos realizam as seguintes três etapas até que o mestre informe que a busca foi finalizada:

1. Determinação do melhor nó (tarefa) para realizar a exploração dentre os compreendidos em sua cópia da Tabela Aphid;
2. Exploração do melhor nó selecionado;
3. Notificação do resultado da exploração da etapa anterior ao mestre (visando obter alguma atualização na sua Tabela Aphid local).

Assim sendo, o APHID tenta minimizar os pontos globais de sincronização definindo uma profundidade d' , denominada fronteira, na qual o único processador mestre é responsável por realizar passagens da raiz até d' . De d' até a profundidade máxima de busca d , a efetuação da busca é de responsabilidade dos processadores escravos. Dessa

forma, apenas os nós da profundidade d' configuram pontos de sincronização, isto é, há uma redução da sobrecarga de sincronização. Além disso, é necessário destacar que o método de distribuição de tarefas a partir do *round-robin* colabora para que os nós disponíveis para avaliação sejam distribuídos de modo equilibrado entre os processadores. Soma-se a isso a estratégia de balanceamento de carga, a qual visa que os processadores não fiquem ociosos - sem tarefas para explorar. Por outro lado, como os processadores estão continuamente explorando tarefa de modo independente uns dos outros, é provável que sejam efetuadas avaliações desnecessárias de nós, isto é, avaliações que não seriam realizadas pelo algoritmo serial, o que contribui para o aumento da sobrecarga de busca.

5.3 Janela de Busca Alfa-Beta

O algoritmo YBWC tenta sempre manter os limites da janela alfa-beta o mais próximo possível do que ocorreria se o algoritmo fosse executado da forma serial, ou seja, tenta manter os processadores envolvidos na busca com as informações mais atualizadas com relação à janela de busca. Tal fato contribui para a atenuação da sobrecarga de busca. Além disso, a própria natureza de exploração síncrona contribui para informar aos processadores escravos sobre possíveis estreitamentos de janela. Mesmo no cenário em que algum processador tenha iniciado a exploração de uma tarefa que esteja fora dos limites da janela, ao receber uma mensagem que identifique uma poda, ele aborta imediatamente esta exploração. Este procedimento apesar de aumentar a sobrecarga de comunicação, objetiva diminuir a sobrecarga de busca.

A Figura 11 exemplifica o mecanismo de compartilhamento das informações da janela alfa-beta no YBWC. Inicialmente, é determinada uma janela a partir da exploração do *eldest brother*. Esta janela é passada de P_0 para os seus processadores escravos (P_1 , P_2 , P_3 e P_4), conforme a Figura 11(A). No cenário considerado na Figura 11(B), o processador P_3 identifica um estreitamento da janela de busca e repassa tal informação ao seu mestre, P_0 . Desta forma, ao constatar a redução dos limites da janela alfa-beta, P_0 replica a informação acerca da nova janela aos demais processadores por meio da mensagem *Nova Janela de Busca 5.1.1*, ilustrado na Figura 11(C). Ressalta-se que, se esta mensagem de atualização da janela alfa-beta não fosse enviada, haveria a possibilidade de um determinado processador P explorar uma tarefa t com os limites de busca maiores do que o melhor limite disponível. Tal fato poderia ocasionar em uma situação na qual P não identificaria uma possível poda na exploração de t , a qual ocorreria caso a exploração fosse realizada com o melhor limite. Logo, a atualização destes nova janela em tais processadores pode ser bastante promissora para o processo de podas. A Figura 11(D) ilustra a situação em que um processador (no caso P_4) identifica que a tarefa que estava explorando é inútil à solução final do algoritmo e a interrompe. De fato, os limites de busca podem possibilitar, por exemplo, que o processador P_4 , executando uma tarefa t , identifique que

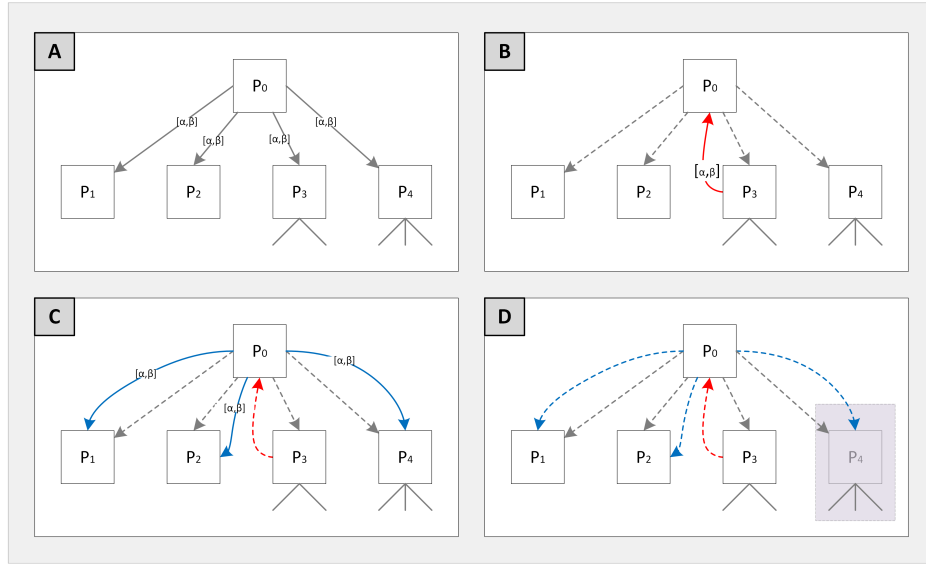


Figura 11 – Mecanismo de atualização da janela alfa-beta no algoritmo YBWC

um dado sucessor t' de t não precisa ser explorado. Neste caso, será necessário avaliar os seguintes cenários:

- Se P_4 tiver iniciado a exploração de t' , ele abortará este processo imediatamente;
- Se t' estiver de posse de outro processador, P_4 enviará uma mensagem de *Poda* 5.1, para que tal processador interrompa sua exploração. Se a exploração de t' por tal processador tiver sido iniciada, ela será abortada imediatamente. Caso contrário, a exploração não ocorrerá.

Este esquema de atualização de janela de busca permite que o YBWC execute o mais próximo possível de como a versão serial do Alfa-Beta executaria, o que permite reduzir a avaliação desnecessária de nós e, consequentemente, diminuir a sobrecarga de busca. Para tanto, é necessário a sincronização dos processadores acerca da atualização de informações relevantes à busca a fim de notificar possíveis estreitamento de janela. Tal fato contribui para elevar a quantidade de mensagens trocadas entre os processadores, o que implica no aumento das sobrecargas de comunicação e de sincronização, pois todas as ocorrências de poda e de atualização dos limites da janela de busca devem ser repassadas a todos os processadores. Sendo assim, em uma árvore ordenada, os limites da janela de busca serão no melhor caso, iguais aos limites praticados pela versão serial. Tal fato permite que o YBWC retorne a mesma solução que a versão serial retornaria. Além disso, salienta-se que nesta situação, a sobrecarga de busca seria bem próxima de nula.

O algoritmo APHID não possui troca de mensagens referente à atualização da janela alfa-beta. Cada processador irá trabalhar com sua própria janela de busca, sendo que esta é baseada no valor hipotético e sua largura estará condicionada a uma constante

definida nas configurações do algoritmo. Este fato almeja reduzir significativamente a sobrecarga de comunicação, uma vez que não é necessário sincronizar as informações da janela de busca entre os processadores. A definição da “abertura” desta janela é definida empiricamente de acordo com cada problema em que o algoritmo venha a ser aplicado. Tal fato apresenta uma fragilidade ao algoritmo, pois o valor desta constante pode comprometer fortemente a performance do algoritmo gerando algum efeito colateral indesejável: se a largura da janela resultante da aplicação desta constante ao valor hipotético for muito pequena, o algoritmo se tornará mais rápido, todavia, ele poderá perder a melhor solução da busca, pois ela poderá ficar fora dos limites desta janela. Neste caso, a resposta final do algoritmo pode ser diferente da resposta que seria retornada pela versão serial do Alfa-Beta. Por outro lado, se esta “largura” for muito grande, o algoritmo ficará mais lento, e, conseqüentemente, aumentará a sobrecarga de busca, uma vez que diversos nós desnecessários serão explorados.

A Figura 12 ilustra o mecanismo de atualização do alfa-beta no APHID. A janela de busca inicial utilizada nas passagens realizadas pelo processador mestre (P_0) possui abertura máxima, ou seja, $\alpha = -\infty$ e $\beta = \infty$. Cada vez que o mestre recebe informação de novos resultados providos de seus escravos, ele reinicia uma nova passagem (sempre com janela $[-\infty - \infty]$), com base nesses novos valores. A cada *passagem* do mestre, a exploração da subárvore de P_0 produz um valor minimax k correspondente ao nó raiz (no caso da Figura 8 na seção 5.2.2.1, $N1$). Este valor é denominado valor hipotético 5.2.2.1, o qual é repassado aos processadores escravos para formação/atualização de suas janelas de busca.

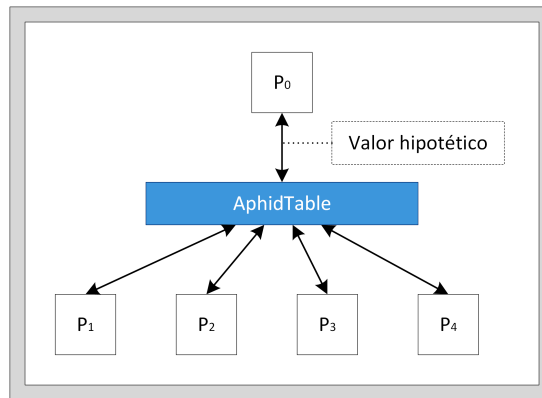


Figura 12 – Mecanismo de atualização da janela alfa-beta no algoritmo APHID

A partir do momento que P_0 efetua a distribuição de tarefas aos escravos, os valores da avaliação dos nós de sua fronteira serão constantemente atualizados pelos resultados recebidos dos escravos que processaram as subárvores desses nós segundo a estratégia de busca por ID. É importante ressaltar que, devido à estratégia de busca, um escravo envia diversos resultados de avaliação de um mesmo nó ao mestre em momentos diferentes, isto é, cada avaliação corresponde a uma profundidade de busca em nível mais profundo.

Desta forma, as novas passagens de $P0$ por sua subárvore irão considerar os últimos valores recebidos dos escravos (os mais atuais) obtidos em profundidades maiores. Tal fato pode ocasionar as seguintes situações:

- Ao longo de uma passagem, $P0$ pode obter um novo valor hipotético k' para $N1$ que deverá ser enviado aos escravos para que eles atualizem suas janelas de busca.
- Nós que existiam na fronteira de busca de uma dada passagem de $P0$, podem ser podados em uma passagem subsequente. Neste caso, se esses nós foram distribuídos a outros processadores escravos, $P0$ irá informar a eles que tais tarefas devem ter a prioridade alterada para a mínima (zero).
- Nós que haviam sido podados em uma passagem anterior de $P0$ por sua subárvore, podem aparecer na fronteira de busca de uma passagem subsequente. Neste caso, haverá duas possibilidades: caso tal nó não tenha sido atribuído a um escravo previamente, ele será distribuído pelo mestre segundo o método *round-robin*; caso contrário, ele será mantido com o mesmo processador, mas com a prioridade reavaliada.
- A ordenação dos nós da fronteira de busca pode ser alterada.

Com relação aos processadores escravos, antes de iniciarem a exploração de suas tarefas, eles definem a janela de busca inicial cujos limites são definidos a partir do valor hipotético enviado por $P0$ e de um valor constante C - definido nas configurações do algoritmo - que irá estabelecer a largura da janela da seguinte forma:

- $\alpha = \text{valor hipotético} - C$;
- $\alpha = \text{valor hipotético} + C$;

Ainda que a janela de busca de um escravo sofra um estreitamento no processamento de uma determinada tarefa, os novos limites não são repassados aos demais processadores, uma vez que no APHID não há atualização de janela entre os escravos (cada processador trabalha com sua própria janela de busca). Assim sendo, os escravos somente atualizam suas janelas em função do valor hipotético recebido do mestre. Considerando o cenário da Figura 10, caso a exploração de $N16$ por $P1$ já tenha iniciado e um novo valor hipotético tenha sido registrado em sua Tabela Aphid, tal valor apenas será considerado na definição de janela da próxima tarefa selecionada na lista de $P1$, isto é, $N16$ não sofrerá o impacto desta atualização.

Portanto, a abordagem adotada pelo APHID quanto à definição e atualização das janelas de busca dos processadores pode aumentar a sobrecarga de busca, bem como

alterar o retorno final do algoritmo, visto que em determinado momento algum nó pode ter sido desconsiderado. Por outro lado, visa diminuir a sobrecarga de comunicação entre os processadores, uma vez que não é necessário sincronizar as janelas de busca entre eles.

5.4 Tabela de Transposição em Arquitetura de Memória Distribuída

Em um jogo de Damas, pode-se chegar a um mesmo estado do tabuleiro várias vezes e, quando isso ocorre, diz-se que houve uma transposição (MILLINGTON, 2006). Desse modo, ressalta-se que os agentes para Damas podem ter seu desempenho melhorado se contarem com um repositório para armazenar estados do tabuleiro que já tenham sido avaliados de modo a utilizar aqueles que reapareçam por transposição, de tal forma que eles não precisam ser reavaliados. Este tipo de repositório é denominado de Tabela de Transposição (TT).

Em um algoritmo paralelo, a implementação de uma TT pode ocorrer de duas formas:

- **Compartilhada:** todos os processadores têm permissão de leitura e de escrita na tabela, pois compartilham a mesma memória RAM. Neste caso, a TT também pode ser chamada de TT global.
- **Distribuída:** cada processador tem a sua própria TT, visto que os processadores não compartilham da mesma memória RAM. Neste caso, para manter as informações sincronizadas em todos os processadores seria necessário uma troca de informações elevada, fato que resultaria em uma sobrecarga de comunicação muito grande, o que é inviável para garantir a boa performance do algoritmo. Este tipo de TT também é denominada de TT local e a sua desvantagem é que pode haver reprocessamento de estados, uma vez que um estado E_i que já foi processado em um processador P_i , pode aparecer em outro processador P_j . Todavia, como P_i e P_j não tem acesso aos mesmos dados na memória, P_j teria que reprocessar E_i .

Assim sendo, a utilização de uma TT global é a solução ideal para problemas como o do jogo de Damas, visto que todos os processadores têm acesso aos estados já explorados por outros processadores. Contudo, este tipo de TT requer uma infraestrutura de memória compartilhada que, em geral, está disponível com equipamentos de hardware com custo financeiro muito elevado. Neste cenário, em situações onde não se dispõe de tais equipamentos, resta a alternativa de uso das TTs distribuídas.

5.4.1 TT distribuída no YBWC

Devido à natureza de divisão de tarefas do YBWC, tal algoritmo foi projetado para apresentar melhores resultados em arquitetura de memória compartilhada, visto que tal arquitetura permite a utilização de tabela de transposição compartilhada. Isto é, o YBWC apresenta melhor desempenho com o uso de TT global. Por exemplo, em arquitetura de memória distribuída, um estado Ex que foi explorado por um processador P_j , pode aparecer em outro momento para o processador P_k . No entanto, como P_k não tem acesso às informações da TT de P_j , ele vai reprocessar Ex novamente. Esta situação é agravada quando é considerado o uso da estratégia de busca ID, pois, considerando que o YBWC realizou a exploração de um dado estado Ex até a profundidade d' na iteração i , quando ele realizar a exploração de Ex novamente na profundidade d'' na iteração $i + 1$, não haverá garantias de que a distribuição das tarefas ficará sob a responsabilidade dos mesmos processadores. Este caso é ilustrado na Figura 13(a), onde na iteração d' um estado é explorado por P_j , no entanto, na próxima iteração na profundidade d'' ele passa a ser explorado por P_k . Todavia, mesmo considerando estes fatores, é preferível a utilização de TT distribuída a não utilizar.

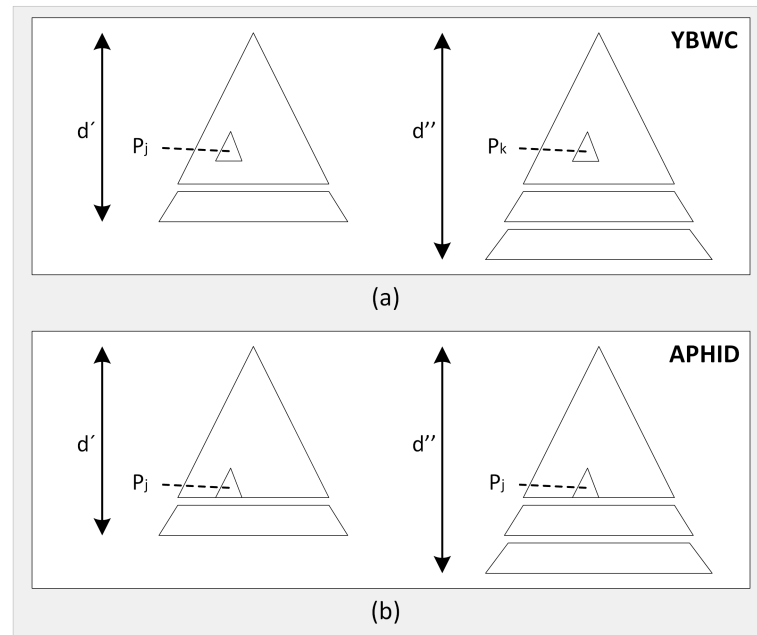


Figura 13 – Exemplificação da alocação de tarefas a um processador quando utilizada a estratégia de busca por Aprofundamento Iterativo no YBWC (a) e no APHID (b)

5.4.2 TT distribuída no APHID

O APHID foi criado para ter bom desempenho não apenas em arquiteturas de memória compartilhada devido ao fato de não criar uma extensa hierarquia entre os processadores, logo cada processador pode explorar ao máximo a utilização de seus recursos

locais. Além disso, como o algoritmo irá explorar um determinado estado Ex até a profundidade d' , e os nós folhas desta exploração serão distribuídos entre os processadores escravos, o algoritmo tenta manter, o máximo possível, a atribuição dos nós aos mesmos processadores durante toda a busca. A Figura 13(B) ilustra esta situação, onde um estado Ex que foi atribuído ao processador P_j na iteração d' será mantido com P_j na iteração d'' , e assim sucessivamente até o final da busca. O estado Ex apenas será atribuído a outro processador se houver a necessidade de realocação de atividades devido a uma sobrecarga de P_j (atividade de balanceamento de carga). Isto significa que, uma vez atribuída uma tarefa a um processador, esta tarefa tem a chance de permanecer com tal processador até o final. Neste caso, a estratégia de busca por aprofundamento iterativo não causa impacto negativo na performance do algoritmo.

5.5 Portabilidade

Um dos fatores que influencia na escolha de um algoritmo a ser inserido em um problema é a complexidade da implementação sobre a aplicação. Se esta complexidade for pequena, o desenvolvedor e/ou pesquisador terá maior motivação em adotar tal algoritmo.

Para lidar com esta situação, a ideia do autor do APHID em (BROCKINGTON, 1998) foi desenvolver uma biblioteca independente. Desta forma, sua integração com aplicações já existentes, que utilizam a versão do Alfa-Beta serial, se torna simples e sem muito impacto para o programador: através de algumas configurações sem complexidade é possível ter a versão paralela do Alfa-Beta funcional. De fato, a biblioteca do APHID foi disponibilizada durante alguns anos na WEB, todavia, na execução do escopo global de doutorado no qual este trabalho está alocado, tal biblioteca não estava disponível. No entanto, a ideia de disponibilizar o algoritmo através de uma biblioteca continua sendo muito interessante, pois cria um meio de facilitação permitindo que outros pesquisadores tenham interesse em aplicar o algoritmo a problemas cujo o propósito seja satisfeito. Neste sentido, tais pesquisadores podem fornecer melhorias ou até mesmo evoluir o algoritmo de busca.

O YBWC Feldmann et al. (1990) não oferece esta facilidade. Para inserir este algoritmo em uma aplicação que já utiliza a versão serial do Alfa-Beta, o programador terá que inserir o mecanismo de troca de mensagens no meio de seu programa. Esta tarefa demanda conhecimento do programador em relação ao processo de trocas de mensagens. Tal fato torna o processo como um todo complexo.

5.6 Considerações Finais

A partir da análise conceitual exposta neste capítulo, pode-se observar que identificar o algoritmo que se destaca em comparação ao outro é uma tarefa bastante complexa, pois o fato de cada um deles (YBWC e APHID) ser baseado em uma abordagem distinta (síncrona e assíncrona) faz com que ambos possuem aspectos positivos que se sobressaem em relação ao outro. De modo a sintetizar e agrupar todas as informações expostas neste capítulo, a Tabela 3 e a Tabela 4 apresentam os principais pontos positivos e negativos levantados, respectivamente, do algoritmo YBWC. Da mesma forma, a Tabela 5 e a Tabela 6 mostram os pontos fortes e fragilidades, respectivamente, do algoritmo APHID.

	Pontos Positivos do YBWC
Comunicação entre os Processadores	- Permite que os processadores trabalhem com as informações mais atualizadas. - Redução da sobrecarga de busca.
Sincronização de Tarefas	- A busca é aprimorada caso os nós mais à esquerda (irmãos mais velhos) gerem uma poda ou um estreitamento na janela de busca. - Tarefas menos complexas podem ser efetuadas localmente.
Janela de Busca Alfa-Beta	- Semelhança com a atualização de janelas do algoritmo serial Alfa-Beta. - Redução da sobrecarga de busca.
Tabela de Transposição	- Sem pontos positivos com relação ao desempenho do algoritmo utilizando TT distribuída.
Portabilidade	- Sem pontos positivos.

Tabela 3 – Pontos positivos do algoritmo YBWC

	Pontos Negativos do YBWC
Comunicação entre os Processadores	- Grande volume de troca de mensagens. - Aumento da sobrecarga de comunicação.
Sincronização de Tarefas	- Diversos pontos globais de sincronização ao longo dos ramos mais à esquerda da árvore de busca. - Aumento da sobrecarga de sincronização. - Causa períodos de ociosidade dos processadores.
Janela de Busca Alfa-Beta	- Necessidade de constante troca de mensagens entre os processadores. - Aumento da sobrecarga de comunicação
Tabela de Transposição	- Possui desempenho melhor com TT global com relação a TT local. - A estratégia de busca por ID agrava a utilização de TT local.
Portabilidade	- Grande esforço de engenharia de software. - Requer conhecimento por parte do programador com relação ao mecanismo de troca de mensagens.

Tabela 4 – Pontos negativos do algoritmo YBWC

	Pontos Positivos do APHID
Comunicação entre os Processadores	- Baixo volume de troca de mensagens. - Comunicação é realizada através da Tabela Aphid. - Redução da sobrecarga de comunicação.
Sincronização de Tarefas	- Pontos globais de sincronização são fixos em uma profundidade pré-estabelecida. - Redução dos períodos de ociosidade dos processadores. - Redução da sobrecarga de sincronização.
Janela de Busca Alfa-Beta	- Redução da sobrecarga de comunicação.
Tabela de Transposição	- Possui melhor desempenho com TT distribuída. - Atribuição dos nós aos mesmos processadores é realizada sempre que possível durante todo o processo de busca.
Portabilidade	- Projetado para ser facilmente integrado a aplicações existentes.

Tabela 5 – Pontos positivos do algoritmo APHID

	Pontos Negativos do APHID
Comunicação entre os Processadores	- Sem pontos negativos.
Sincronização de Tarefas	- Há uma relativa exploração desnecessária de nós. - Aumento da sobrecarga de busca.
Janela de Busca Alfa-Beta	- Cada processador trabalha com uma janela diferente. - Possibilidade de explorar nós não relevantes para a busca. - Aumento da sobrecarga de busca.
Tabela de Transposição Distribuída	- Pequena possibilidade de nós serem designados a processadores distintos durante as iterações de busca.
Portabilidade	- Desafio de se criar uma biblioteca para o algoritmo.

Tabela 6 – Pontos negativos do algoritmo APHID

Assim sendo, a partir das tabelas acima, é possível extrair os melhores aspectos positivos encontrados nos algoritmos YBWC e APHID. Pode-se observar que a principal vantagem do algoritmo YBWC é com relação à atualização da janela de busca alfa-beta, uma vez que o mecanismo utilizado neste algoritmo procura se assemelhar ao método utilizado no algoritmo serial Alfa-Beta. Tal fato contribui significativamente para que o retorno do YBWC seja igual ou parecido com o do algoritmo serial, além de evitar, ao máximo, que seja realizada a avaliação de nós desnecessários para a avaliação da busca. Quanto ao algoritmo APHID, é possível destacar diversos pontos positivos, dentre eles: a sua natureza assíncrona, a qual permite que os processadores realizem buscas simultaneamente sem depender dos resultados de outros processadores, evitando longos períodos de ociosidade; o baixo volume de troca de mensagens, uma vez que os processadores se comunicam entre si por meio da estrutura Tabela Aphid, ao invés da rede de interconexão, o que reduz substancialmente o impacto da sobrecarga de comunicação; ótimo desempenho atuando com TT local. Ressalta-se que estas características destacadas foram consideradas para a construção de um novo algoritmo assíncrono de busca denominado ADABA, no escopo global no qual o presente trabalho está alocado.

6 Método Automatizado para Avaliação de Qualidade de Movimentos de Agentes Jogadores de Damas

Esta seção apresenta a arquitetura do sistema automático que serve como ferramenta avaliadora da qualidade da tomada de decisão de agentes jogadores de Damas a partir da plataforma *CheckerBoard*. Basicamente, ela consiste de uma integração entre dois módulos: Ambiente de Atuação dos Agentes Avaliados (AAAA) e Ferramenta de Avaliação de Qualidade de Movimentos (FAQM), conforme ilustrado na Figura 14 e detalhado na sequência.

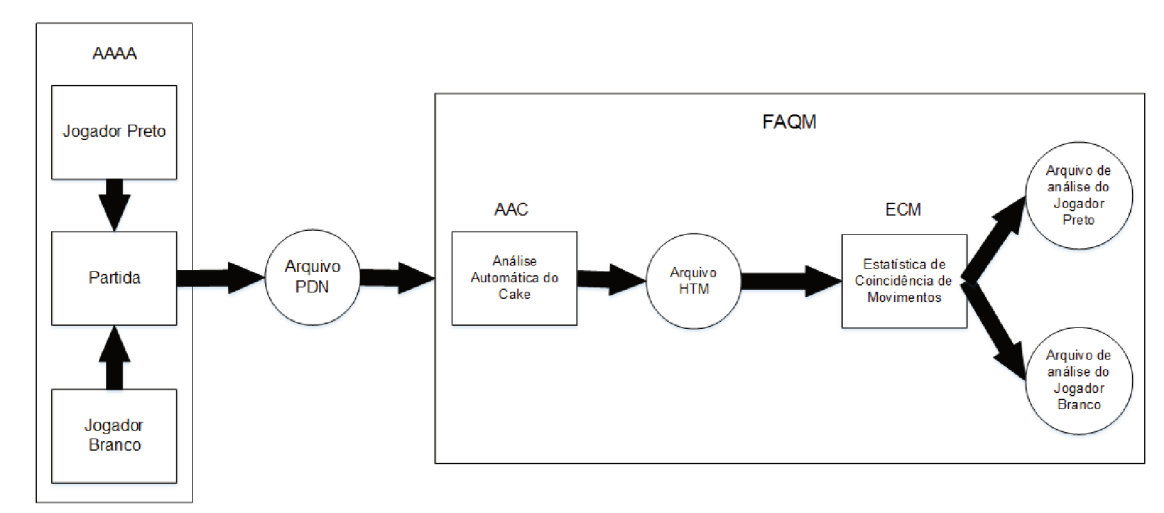


Figura 14 – Arquitetura do sistema de automatização da análise estatística de movimentos

Módulo AAAA: Este componente tem como objetivo gerar os arquivos PDN que armazenam os conjuntos de movimentos executados pelos agentes a serem avaliados ao longo de um torneio composto por diversos jogos-teste. Cada um desses conjuntos se refere a um jogo-teste completo (a partir do qual será gerado um dos arquivos PDN). A seção 6.1 detalha o processo de geração do arquivo PDN a partir do módulo de busca de um agente. Contudo, os tabuleiros, a partir do qual cada um desses jogos tem início, variam de um conjunto para outro - sendo que um deles corresponde, necessariamente, ao tabuleiro inicial padrão de Damas, ao passo que cada um dos demais contém um número de peças pré-estabelecido, respeitando-se a restrição de que, tanto a quantidade, quanto a natureza das peças pretas e brancas

têm de ser a mesma para ambos os jogadores envolvidos. Convém salientar que tal estratégia é fundamental para a acuidade do cálculo da qualidade de movimentos, uma vez que em Damas (como em vários outros problemas práticos) o desempenho do agente é extremamente dependente do estado inicial do qual partiu. Tal fator é tão significativo que, tal como citado na seção 4.1.1, a garantia de invencibilidade do *Chinook* somente ocorre em jogos que partem do tabuleiro inicial padrão do jogo de Damas (SCHAEFFER et al., 2007). Ainda com relação aos jogos-teste, é importante ressaltar que, neles, o agente pode enfrentar qualquer adversário, inclusive ele mesmo (situação a que se denomina jogos por *self-play*). Assim sendo, caso os jogos-teste envolvam dois adversários A (jogador preto) e B (jogador branco) distintos, os arquivos PDN gerados pelo módulo armazenarão, tanto os conjuntos de movimentos executados pelos jogador A (jogando com as peças pretas), quanto os conjuntos executados pelo competidor B (jogando com as peças brancas). Por outro lado, caso os jogos-teste ocorram por *self-play*, os conjuntos armazenados nos arquivos PDN se referirão aos movimentos executados pelo único jogador envolvido na competição jogando, ora com as peças pretas, ora com as peças brancas. Uma vez gerados todos os arquivos PDN correspondentes a todos os jogos-teste do torneio, o módulo AAAA produz uma pasta contendo todos eles.

Módulo FAQM: Este componente tem duas finalidades básicas: 1) Gerar um conjunto de informações relacionadas ao comportamento que o agente *Cake* teria tido diante das várias situações (estados de tabuleiro) enfrentadas pelos agentes durante os jogos-teste; 2) Gerar as taxas de coincidência entre os movimentos executados pelos agentes nos jogos-teste (armazenados no arquivo PDN) e os movimentos que teriam sido executados pelo *Cake* nas mesmas situações. A primeira finalidade é executada no sub-módulo do FAQM denominado Análise Automática do *Cake* (AAC), ao passo que a segunda é cumprida no sub-módulo denominado Estatística de Coincidência de Movimentos (ECM). Tais sub-módulos serão descritos com mais detalhes nas seções 6.2 e 6.3, respectivamente.

6.1 Geração do Arquivo PDN

Conforme mencionado na seção 1.2.2, o arquivo PDN é gerado a partir do módulo de busca de agente jogadores ao longo de partidas. Dessa forma, a Figura 15 - que ilustra a arquitetura geral e o ciclo de operações executadas por um dos jogadores automáticos relevantes para a execução deste trabalho, denominado *D-VisionDraughts* 4.2.2, a cada vez que ele efetua uma tomada de decisão para a escolha de um movimento a partir do estado corrente do jogo - será utilizada para detalhar o processo de geração do PDN. Ressalta-se que o *D-VisionDraughts* possui os seguintes tipos de jogo:

1. *Treino*: trata-se dos jogos compreendidos na etapa de treinamento da *Multilayer Perceptron* (MLP);
2. *Disputa*: trata-se dos jogos em que o *D-VisionDraughts* está apto a jogar uma partida contra um oponente.

Em função do tipo de jogo, o ciclo da Figura 15 apresenta comportamento particular para os jogos de disputa em que somente os passos de #1 a #6 são executados. Os passos restantes (#7 a #13) são realizados somente na etapa de treinamento do referido jogador. Ressalta-se que o arquivo PDN é gerado somente em jogos do tipo *disputa*.

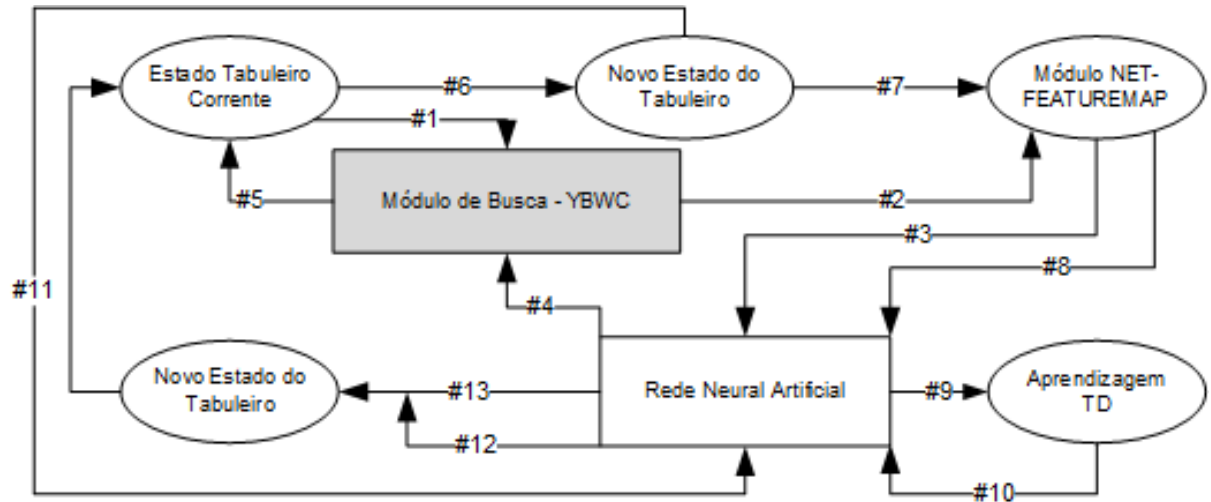


Figura 15 – Arquitetura geral do jogador D-VisionDraughts

O ciclo de operações do *D-VisionDraughts* relativo aos jogo de disputa compreende os seguintes passos:

Passo #1: o estado corrente do tabuleiro (referido aqui como T_x) é apresentado ao *Módulo de Busca*. Assim sendo, T_x foi obtido a partir da execução do último movimento (chamado aqui m_x).

Passo #2: o *Módulo de Busca* realiza uma busca em profundidade limitada ou uma busca por ID, utilizando o algoritmo de busca paralelo YBWC, a partir do estado de tabuleiro T_x . As folhas da árvore de jogo são repassadas ao mapeamento NET-FEATUREMAP.

Passo #3: cada folha da árvore de busca do jogo corresponde a um estado de tabuleiro que, representado por NET-FEATUREMAP, é apresentado à camada de entrada da rede MLP para ser avaliado.

Passo #4: os valores calculados no Passo #3 são retornados ao *Módulo de Busca* para cálculo do melhor movimento m_y a ser executado.

Passo #5: o movimento m_y é calculado e retornado ao sistema para que ele o execute a partir do estado corrente T_x .

Passo #6: o movimento m_y sugerido pelo Passo #5 é executado e o estado corrente do tabuleiro é modificado para T_y (novo estado de tabuleiro). Em jogos de disputa, o ciclo de escolha do melhor movimento termina neste passo (a partir daí, cabe ao oponente escolher o movimento no estado T_y).

Desta forma, é possível observar que o *Módulo de Busca* é responsável pela tomada de decisão do agente em função do estado corrente do tabuleiro por meio de um algoritmo de busca (no caso da Figura 15, o algoritmo paralelo YBWC). Assim sendo, durante a realização de uma partida, o algoritmo de busca deve ser capaz de gravar o melhor movimento calculado ao longo do Passo #5 (m_y) no arquivo PDN, a cada ciclo realizado pelo jogador que efetua a ação. Este processo é executado até o término da partida, no qual o arquivo PDN gerado conterá todos os movimento realizados pelos jogadores envolvidos, além das informações explanadas na seção 3.1. Salienta-se que os algoritmos de busca paralelos YBWC e APHID implementados nos agentes automáticos produzidos pela equipe pesquisadora possuem tal habilidade (esta etapa foi efetuada em um contexto de Iniciação Científica, conforme mencionado no capítulo 1).

6.2 Sub-Módulo AAC: Análise Automática do *Cake*

O objetivo deste sub-módulo do FAQM é gerar um conjunto de informações referentes à maneira como o agente *Cake* 4.1.2 se comportaria em situações de tabuleiro análogas às enfrentadas ao longo dos jogos-teste efetuados em um torneio pelos agentes a serem analisados. Tais informações são armazenadas em arquivos com extensão *htm*, referenciados a partir daqui como HTM. Cada arquivo HTM corresponde a um dado arquivo PDN e é produzido da seguinte forma: o módulo AAC reproduz no ambiente da plataforma *CheckerBoard* a partida armazenada no arquivo PDN que lhe corresponde (conforme seção 3.1). Para cada movimento obtido a partir dessa partida reproduzida, o sub-módulo recupera um conjunto contendo as seguintes informações disponibilizadas pela plataforma relativas àquele movimento (sob a ótica do *Cake*):

- *Movimento-agente-avaliado*: movimento executado pelo agente avaliado (informação recuperada a partir do arquivo PDN correspondente);
- *Lado*: cor (branco ou preto) com que o agente avaliado executou o movimento acima;
- *Movimento-Cake*: movimento que teria sido escolhido pelo *Cake* na mesma situação de tabuleiro e de cor com que foi executado o movimento do agente avaliado.

Conforme pode-se observar, as informações consideradas incluem o parâmetro *lado* representando a cor das peças com que o movimento foi executado. Isso representa um cuidado essencial, pois a avaliação de um dado movimento sofre profundo impacto em função do lado que o executou.

A título de aumentar a eficiência avaliativa do sistema aqui proposto, o sub-módulo AAC foi implementado de modo a estar apto a gerar diversos arquivos HTM simultaneamente. Para tanto, ele conta com dois parâmetros: *torneio* e *nro-jogos*. O primeiro, indica o caminho para a pasta de arquivos PDN produzida pelo módulo AAAA. O segundo, indica a quantidade de arquivos PDN armazenados na referida pasta. Tal habilidade permite que as informações referentes à totalidade dos jogos-teste sejam recuperadas em um único processamento do AAC, o que é um fator determinante para a redução do tempo necessário para se analisar uma quantidade significativa de arquivos PDN.

Uma vez que os arquivos HTM foram gerados, o AAC produz uma pasta contendo os mesmos a ser utilizada pelo sub-módulo ECM exposto na próxima seção.

6.3 Sub-Módulo ECM: Estatística de Coincidência de Movimentos

A finalidade deste sub-módulo é gerar as taxas de coincidência entre os movimentos executados pelos agentes nos jogos-teste (armazenados na pasta de PDNs) e os movimentos que teriam sido executados pelo *Cake* nas mesmas situações (armazenados na pasta de HTMs). Para tanto, o ECM, a partir das informações produzidas pelo sub-módulo AAC relativas aos movimentos executados pelos agentes avaliados no torneio (ou seja, *Movimento-agente-avaliado*, *Lado* e *Movimento-Cake*), calcula a estatística de coincidências entre todos os movimentos executados por eles ao longo do torneio e os movimentos que o *Cake* teria executado nas mesmas condições de jogo (tabuleiro e cor).

Ao final da análise, o ECM gera um arquivo para cada jogador avaliado com as seguintes informações estatísticas:

- Quantidade total de movimentos realizados pelo jogador;
- Quantidade total de movimentos realizados pelo agente coincidentes com o *Cake*;
- Média da porcentagem de movimentos realizados pelo agente coincidentes com o *Cake*;

Tais estatísticas são calculadas, independentemente, para ambos os lados em que o jogador automático pode atuar. Assim sendo, o referido arquivo é composto por três conjuntos de informações estatísticas distintos: os dois primeiros, onde as informações se referem exclusivamente às estatísticas envolvendo os movimentos executados com peças

brancas ou pretas, respectivamente, e, o terceiro, apresentando as informações globais consolidadas relativas aos jogos completos (jogadas com peças brancas e pretas). Tal estratégia é bastante pertinente, uma vez que permite avaliar as taxas de coincidências de movimentos dos agentes avaliados em relação ao *Cake* nos cenários em que jogam como jogador branco e como jogador preto, além de prover uma visão global do desempenho dos mesmos jogando em ambos os lados.

6.4 Considerações Finais

A partir do uso da ferramenta automática explanada neste capítulo, espera-se que o tempo para realizar a análise da qualidade da escolha de movimentos de agentes automáticos de Damas seja substancialmente mais rápido com relação ao método manual de comparação de movimentos com o *Cake* utilizado em trabalhos anteriores. Dessa forma, a principal meta do sistema automático é a avaliação de um número significativo de jogos completos entre jogadores automáticos a fim de verificar, com a maior precisão possível, as taxas de movimentos coincidentes com o agente *Cake* obtidas por tais jogadores, de maneira menos onerosa com relação à avaliação manual.

7 Experimentos e Resultados

Este capítulo apresenta os experimentos realizados a fim de validar tanto a veracidade dos resultados produzidos pelo método automático avaliador da qualidade da tomada de decisão de agentes jogadores de Damas explicado no capítulo 6, quanto a superioridade de desempenho, em jogos reais, de agentes que adotam como módulo de busca o algoritmo APHID sobre aqueles que utilizam o algoritmo YBWC.

Desta forma, foram utilizadas duas aplicações como estudo de caso. A primeira estabelece uma análise comparativa referente à performance dos seguintes agentes automáticos: o *VisionDraughts* (o qual utiliza como mecanismo de busca o algoritmo serial Alfa-beta) e sua versão aprimorada *D-VisionDraughts* (o qual, por sua vez, utiliza a versão distribuída síncrona do Alfa-Beta denominada YBWC). A segunda efetua a comparação de desempenho entre os jogadores automáticos *D-VisionDraughts* e *APHID-Draughts* (cujo módulo de busca é norteado pela versão distribuída assíncrona do Alfa-Beta denominada APHID).

7.1 Estudo de Caso 1: VisionDraughts x D-VisionDraughts

Os agentes avaliados como primeiro estudo de caso são o *VisionDraughts* e o *D-VisionDraughts*. Ressalta-se que trabalhos anteriormente publicados já haviam comprovado, por meio de tempo de treinamento e de número de vitórias, a superioridade do *D-VisionDraughts* em relação a seu oponente (TOMAZ; JULIA; BARCELOS, 2013). A Figura 16 ilustra o tempo necessário de treinamento de tais agentes e a Tabela 7 apresenta os resultados de torneios entre ambos (cada torneio composto de 40 jogos), com relação ao número de características utilizadas para o treinamento. É importante destacar que tanto a Figura 16 quanto a Tabela 7 apresentadas foram retiradas do referido trabalho. Além disso, o valor P encontrado nelas corresponde ao número de processadores utilizados.

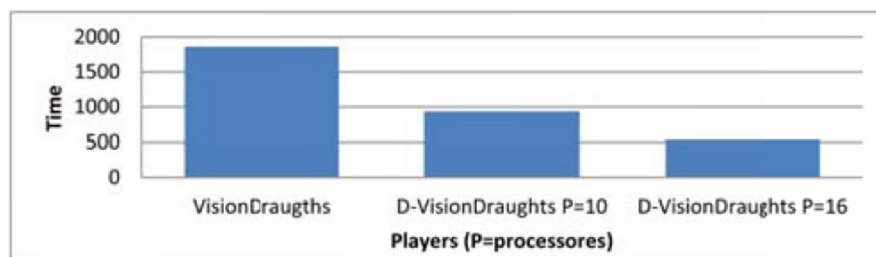


Figura 16 – Tempo de treinamento (em minutos) dos jogadores: *VisionDraughts*, *D-VisionDraughts* ($P=10$) e *D-VisionDraughts* ($P=16$) (TOMAZ; JULIA; BARCELOS, 2013)

Número de Características	Vitória	Empate	Derrota
14	6	34	0
16	7	32	1

Tabela 7 – *D-VisionDraughts* (P=16) contra *VisionDraughts* (TOMAZ; JULIA; BARCELOS, 2013)

Assim sendo, os testes aqui realizados calculam a taxa média de coincidência entre os movimentos escolhidos pelos agentes *VisionDraughts* e *D-VisionDraughts* no curso de jogos-teste em que eles se enfrentam e os movimentos que, nas mesmas situações, seriam executados pelo agente *Cake*, a partir do método implementado e explanado na seção 6. De modo a manter a coerência dos resultados, as versões de tais agentes utilizadas nos experimentos são as mesmas empregadas nos testes realizados em (TOMAZ; JULIA; BARCELOS, 2013). Dessa forma, são apresentados dois cenários de teste (executados em uma máquina com processador *Intel Core i5-4440S CPU @ 2.80GHz com 4 núcleos e 8 GB de RAM*):

Cenário 1: avaliação da qualidade de movimentos dos agentes *VisionDraughts* e *D-VisionDraughts* com relação ao *Cake*, em situação em que jogam com peças pretas;

Cenário 2: avaliação da qualidade de movimentos dos agentes *VisionDraughts* e *D-VisionDraughts* com relação ao *Cake*, em situação em que jogam com peças brancas;

É importante salientar que cada um desses cenários é composto por 20 jogos-teste, totalizando 40 jogos analisados. Além disso, conforme previsto na seção 6, tais jogos partem de estados iniciais pré-definidos e diversos, respeitando-se a restrição de que, tanto a quantidade, quanto a natureza das peças, são as mesmas para ambos os agentes. Tal fato impede que um agente esteja em posição mais vantajosa do que o outro, o que valida a consistência dos resultados, além de permitir que ambos sejam avaliados em situações variadas de jogo (uma vez que, conforme apresentado na seção 6, o tabuleiro inicial tem forte impacto no desempenho do agente).

A Tabela 8 apresenta os resultados comparativos referentes aos cenários definidos. Ao analisar tais resultados, observa-se a superioridade geral (atuando tanto com peças pretas quanto com peças brancas) da qualidade dos movimentos executados pelo *D-VisionDraughts* com relação a seu oponente. De fato, como jogador preto, o *D-VisionDraughts* obteve uma taxa de coincidência de movimentos com relação ao *Cake* de 43,25%, contra 35,01% obtida pelo *VisionDraughts*. Além disso, como jogador branco, essas mesmas porcentagens para o *D-VisionDraughts* e seu oponente foram de 33,05% e de 31,52%, respectivamente. Esses resultados atestam a melhor qualidade de tomada de decisão do *D-VisionDraughts* com relação ao *VisionDraughts*, o que é coerente com o fato de o primeiro

corresponder a uma versão otimizada - implementada em ambiente de alto desempenho - do segundo agente.

Posição do agente no jogo	VisionDraughts	D-VisionDraughts
Peças Pretas	35.01%	43.25%
Peças Brancas	31.52%	33.05%

Tabela 8 – Taxas de coincidência de movimentos com o Cake

7.1.1 Análise Estatística de Wilcoxon

Comparações pareadas são o tipo mais simples de testes estatísticos utilizados para comparar o desempenho de dois ou mais algoritmos quando aplicados a um mesmo problema. Em caso de múltiplos problemas, a comparação requer a definição de um valor para cada par algoritmo/problema. Normalmente, estes valores correspondem a um valor médio que é obtido através de várias execuções (DERRAC et al., 2011). Um dos métodos mais robustos baseado em comparação pareada é o método estatístico chamado *Wilcoxon*, que também é conhecido como teste de ordenação de sinais (do inglês *signed-ranks test*). O objetivo do teste de *Wilcoxon* é comparar o desempenho de cada algoritmo (ou pares de algoritmos, por exemplo A e B) no sentido de verificar se existem diferenças significativas entre os seus resultados (amostras ou populações).

Resumidamente, o método de *Wilcoxon* opera da seguinte forma: os resultados obtidos por A são subtraídos dos resultados obtidos por B e à diferença resultante (d) é atribuído o sinal mais (+) ou, caso seja negativa, o sinal menos (-). Estas diferenças são ordenadas em função de sua grandeza (independentemente do sinal positivo ou negativo). O ordenamento assim obtido é depois apresentado separadamente para os resultados positivos (R^+) de A e negativos (R^-) de B . Os resultados positivos representam momentos em que o algoritmo A supera B e vice-versa para os resultados negativos. O menor dos valores deste segundo grupo, dá-lhe o valor de uma estatística designada por p -value e pode ser consultada na tabela de significância apropriada (DERRAC et al., 2011). A ideia é que se existirem apenas diferenças aleatórias, tal como é postulado pela hipótese nula (aqui denominada por H_0), então haverá aproximadamente o mesmo número de ordens elevadas e de ordens inferiores tanto para as diferenças positivas (R^+) quanto para as negativas (R^-). A hipótese nula é uma declaração sem efeito ou nenhuma diferença e é esperado ser rejeitada pelo experimentador. Um exemplo de hipótese nula é que duas amostras (ou resultados) representam a mesma população (ou seja, não há nenhuma diferença). Caso se verifique uma preponderância de baixos resultados para um dos lados (A ou B), isso significa que há existência de muitos resultados elevados para o outro lado, indicando uma diferença em favor de um dos algoritmos. Tal fato contraria a hipótese nula H_0 . Portanto, dado que a estatística p -value reflete o menor total de ordens (R^+ ou R^-), quanto menor for p -value mais significativas são as diferenças nas ordenações entre os dois algoritmos.

Mais detalhes sobre o método estatístico de *Wilcoxon* podem ser obtidos em (DERRAC et al., 2011).

Particularmente neste estudo de caso, as amostras correspondem aos resultados alcançados pelos algoritmos (agentes automáticos) que são candidatos para resolver um determinado problema (jogo de Damas). Em outras palavras, o objetivo de usar *Wilcoxon* é verificar se o agente *D-VisionDraughts* realmente apresenta um desempenho diferenciado em relação ao seu adversário *VisionDraughts*. Destaca-se que essa análise utilizando *Wilcoxon* não foi utilizada no trabalho (TOMAZ; JULIA; BARCELOS, 2013). Dessa forma, esta subseção tem a finalidade de complementar os resultados encontrados no referido trabalho aplicando tal análise aos torneios realizados entre ambos jogadores (Tabela 7).

Assim sendo, nas análises dos resultados consolidados na Tabela 7, o método de *Wilcoxon* assume, como ponto de partida, a seguinte hipótese nula H_0 : *D-VisionDraughts* possui o mesmo nível de desempenho que o *VisionDraughts*.

De acordo com o referido método, os seguintes passos devem ser seguidos a fim de provar que H_0 deve ser rejeitada (NETO; JULIA; CAEXETA, 2014):

Passo 1: Dado T como a menor soma dos *ranks* (ordenações) positivas e negativas dentre as diferenças dos pares de amostras, isto é, $T = \min\{R^+, R^-\}$ (particularmente neste trabalho, R^+ representa a soma dos *ranks* em que o *D-VisionDraughts* superou o *VisionDraughts*, enquanto R^- é a soma dos *ranks* em que o *D-VisionDraughts* foi superado pelo *VisionDraughts*). Uma apropriada tabela estatística ou ferramenta deve ser utilizada para determinar o *teste estatístico*, *valor crítico* ou *p-value* (dado provido pelo teste);

Passo 2: Rejeita-se a *hipótese nula* H_0 se o *teste estatístico* $\leq$ *valor crítico* ou se *p-value* $\leq \alpha$ (nível de significância). Quanto mais baixo é o valor de α , mais forte é a evidência contra a hipótese nula H_0 .

A Tabela 9 apresenta os testes estatísticos de *Wilcoxon* aplicados sobre os resultados dos jogos executados na Tabela 7, onde a seguinte pontuação foi definida: 2 pontos para vitória, 1 ponto para empate e 0 ponto para derrota. Os valores R^+ , R^- e *p-value* foram computados utilizando o *software* de estatística SPSS.

Número de características	R^+	R^-	<i>p-value</i>
14	21	0	0,014306
16	31,5	4,5	0,033895

Tabela 9 – Método *Wilcoxon signed rank test* aplicado aos resultados dos torneios entre o *D-VisionDraughts* e *VisionDraughts* da Tabela 7

Como é possível observar, o *D-VisionDraughts* se mostrou superior ao *VisionDraughts* com um alto nível de significância de $\alpha = 0.05$ ou 95% nos dois cenários avaliados: o primeiro, considerando 14 características utilizadas no treinamento; e, o segundo, com 16 características no treinamento. Tal fato representa que há 5% de chance da hipótese nula H_0 (equivalência de desempenho entre o *D-VisionDraughts* e o *VisionDraughts*) ocorrer na base de dados (amostra) analisada. Em outras palavras, os dados apresentados na Tabela 9 rejeitam fortemente H_0 , o que significa que os resultados alcançados pelo *D-VisionDraughts*, nos jogos de torneio apresentados na Tabela 7, são de fato superiores em relação ao *VisionDraughts*. Isto corrobora o fato de que utilizar o algoritmo YBWC como mecanismo de busca é mais eficiente comparado ao algoritmo alfa-beta serial.

7.1.2 Considerações do Estudo de Caso 1

O propósito desta seção foi apresentar uma aplicação já existente de comparação entre dois agentes automáticos de Damas (*VisionDraughts* e *D-VisionDraughts*) e, por meio de tal aplicação, autenticar os dados gerados pelo uso do método automático implementado neste trabalho 6.

Dessa forma, tal ferramenta automática avaliadora da qualidade de movimentos apresentada pode ser validada por meio de uma comparação entre os resultados produzidos por ela apresentados na Tabela 8 e os testes avaliativos efetuados pelos autores dos citados agentes em (TOMAZ; JULIA; BARCELOS, 2013), Figura 16 e Tabela 7. De fato, conforme antecipado na seção 4, os criadores do *D-VisionDraughts* e do *VisionDraughts* provaram em seus experimentos que o primeiro tem um desempenho geral em termos de taxas de vitórias em torneios 17,5% superior ao de seu antecessor. Salienta-se que o parâmetro *taxa de vitórias* está diretamente associado à qualidade dos movimentos selecionados pelo agente jogador, o que é coerente com os resultados obtidos na Tabela 8, na qual o *D-VisionDraughts* apresentou taxas de coincidência com o agente *Cake* maiores do que as obtidas pelo *VisionDraughts*. Tais fatos são corroborados pela aplicação do método de *Wilcoxon*, o qual provou que, realmente, o *D-VisionDraughts* apresenta um desempenho melhor do que o *VisionDraughts*, proveniente do seu mecanismo de busca YBWC, a partir dos dados providos da Tabela 9.

Além disso, um outro aspecto relevante a ser destacado é a agilidade que a ferramenta automática aqui proposta proporciona ao processo avaliativo baseado em comparações com os movimentos indicados pelo *Cake*. De fato, o cálculo estatístico automático relativo às 40 partidas envolvidas nos cenários citados foi efetuado pelo sistema proposto em apenas 3 minutos e 10 segundos, aproximadamente. Obviamente, uma avaliação análoga feita manualmente (tal como efetuado em (NETO; JULIA; CAEXETA, 2014) e em (DUARTE et al., 2015)) requereria um tempo substancialmente maior, além de ser passível de falhas, uma vez que precisaria tratar individualmente cada movimento, de cada

partida, comparando-o, manualmente, com o resultado da busca de melhor movimento do *Cake*. Isso diminuiria, drasticamente, a quantidade de jogos-teste e de movimentos que poderiam ser analisados, limitação que desaparece com o uso da ferramenta automática aqui implementada.

7.2 Estudo de Caso 2: D-VisionDraughts x APHID-Draughts

Os agentes avaliados como segundo estudo de caso são o *D-VisionDraughts* e o *APHID-Draughts*. Esta seção apresenta os experimentos realizados com os referidos agentes automáticos, de modo a comparar as abordagens síncrona e assíncrona em um ambiente de arquitetura de memória distribuída. De fato, o *D-VisionDraughts* e o *APHID-Draughts* são similares em relação às suas arquiteturas, diferindo apenas no mecanismo de busca. No primeiro, é empregado o algoritmo distribuído YBWC (abordagem síncrona) e, no segundo, o APHID (abordagem assíncrona). Neste contexto, foi realizada, no escopo global de Doutorado no qual o presente trabalho está alocado, uma comparação das abordagens de cada algoritmo aplicada ao problema do jogo de Damas considerando os seguintes parâmetros: tempo de treinamento dos agentes, taxas de vitória e avaliação da qualidade da escolha de movimentos segundo o renomado agente *Cake* (seção 4.1.2). É importante destacar que a principal contribuição deste trabalho para os testes executados foi a extração das estatísticas referentes às taxas de coincidência com o *Cake* de ambos agentes, a partir da ferramenta automática produzida, a qual foi validada pelo estudo de caso anterior. Além disso, ressalta-se que tal estudo de caso culminou na produção de um artigo intitulado *APHID-Draughts: comparing the synchronous and asynchronous parallelism approaches for the Alpha-Beta algorithm applied to Checkers*, publicado na *29th International Conference on Tools with Artificial Intelligence*, qualis B1, pela equipe deste trabalho.

Salienta-se que todos os testes foram executados a partir de 5 estações de igual configuração: com processadores *Quad Core* - totalizando 20 núcleos de processamento - com 8GB de memória RAM. Todas estas estações foram interligadas por um *Switch Gigabit Ethernet*.

7.2.1 Tempo de Treinamento dos Agentes

As redes MLPs foram treinadas a partir da estratégia *self-play* com clonagem 6. Foram executados 100 jogos no processo de treino de cada MLP. Além disso, este processo foi realizado para cada uma das seguintes profundidades de busca: 10, 12 e 14. A Figura 17 ilustra o gráfico do tempo de treinamento obtido, em minutos, para os agentes *APHID-Draughts* e *D-VisionDraughts* em cada uma das profundidades de busca.

Como é possível observar, com profundidade igual a 10, o tempo de treino do *D-*

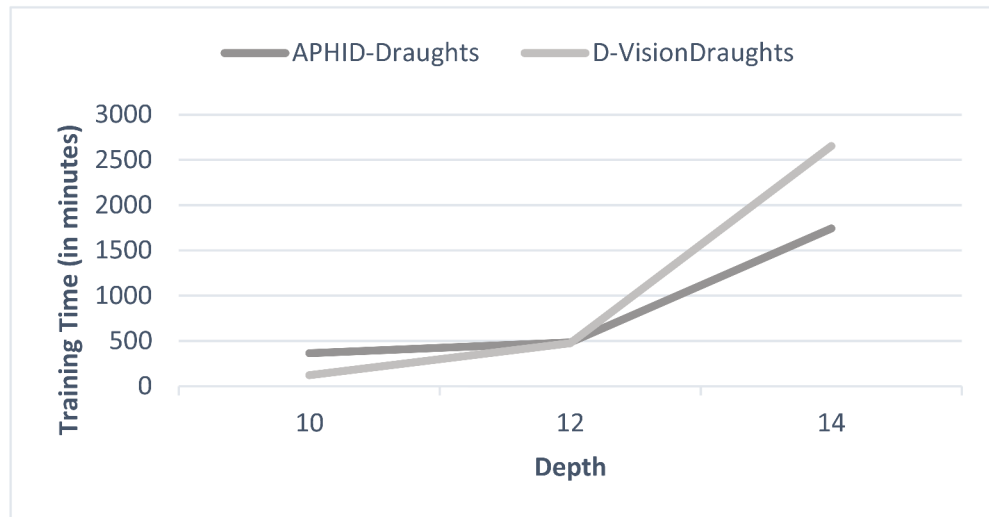


Figura 17 – Tempo de treinamento (em minutos) dos jogadores D-VisionDraughts e APHID-Draughts por profundidade

VisionDraughts foi 66% (122 minutos) menor que o despendido pelo *APHID-Draughts*. No entanto, a medida que a profundidade de busca foi acrescida, este cenário foi invertido: o tempo de busca do *APHID-Draughts* na profundidade 12 (474 minutos) e 14 (1743 minutos) foi 1.6% e 35% menor, respectivamente, daqueles gastos pelo *D-VisionDraughts* (482 minutos para profundidade 12 e 2654 minutos para a profundidade 14). Estes resultados podem ser justificados pelos seguintes pontos:

1. O alto número de sincronizações necessárias no algoritmo YBWC resulta em processadores ociosos diversos períodos durante a busca. Este inconveniente é intensificado em profundidades maiores, pois quanto maior a profundidade de busca, maior o fator de ramificação, uma vez que estados avançados do tabuleiro de busca são encontrados, como por exemplo, aqueles que apresentam maior número de rainhas, o que resulta em uma maior gama de opções de movimentos para o jogador;
2. O método de trabalho do YBWC, que é por roubo de trabalho (*working-stealing*), não é apropriado para arquitetura de memória distribuída. Por outro lado, é adequado para arquitetura de memória compartilhada a partir do uso de TT global, na qual todos os processadores tenham acesso às informações previamente obtidas por outros a respeito da avaliação de um mesmo estado de tabuleiro, conforme explicado na seção 5.4.1.
3. O *APHID-Draughts* pode apresentar sobrecarga de busca acentuada, uma vez que não há compartilhamento de informações referentes a janela de busca. Como o YBWC tenta manter a janela de busca o mais próximo possível da versão serial, devido ao compartilhamento dos limites alfa e beta, em profundidades mais rasas existe um equilíbrio favorável ao YBWC entre o número de podas e a sobrecarga de

comunicação resultante das inúmeras mensagens inerentes ao YBWC. Neste ponto, é importante ressaltar que, a boa performance do *APHID-Draughts* se deve à adequada definição da largura da janela de busca dos escravos no APHID (conforme apresentado na seção 5.3). Neste experimento, a abertura da janela foi definida empiricamente como 0.1.

Apesar desta seção ter considerado apenas o tempo de treinamento nas análises realizadas, é importante ressaltar que, apesar de cada jogador ter a mesma arquitetura e ter executado o treinamento de suas respectivas MLPs em profundidades iguais, as MLPs treinadas resultantes de cada jogador não são iguais. Tal fato ocorre, pois apesar de o APHID e o YBWC serem versões paralelas do algoritmo Alfa-Beta, o resultado final do algoritmo pode não ser idêntico para um mesmo estado de jogo em função da definição dos limites da janela de busca no APHID, pois a solução ótima (que seria retornada pelo algoritmo serial) pode ficar fora do intervalo de busca, como destacado na seção 5.3.

7.2.2 Taxas de Vitória

A Tabela 10 apresenta as taxas de vitória obtidas pelo *APHID-Draughts* em um torneio em que seu oponente foi o *D-VisionDraughts*. Cada jogador utilizou sua respectiva MLP treinada na profundidade 14. A estratégia de busca adotada foi por ID, sendo que cada jogada foi limitada a 30 segundos. Os jogos contemplaram 20 estados diferentes, onde cada estado tinha o mesmo número de peças para cada jogador (preto ou vermelho). Este torneio foi executado em duas rodadas: na primeira, o jogador *APHID-Draughts* jogou com peças pretas e o *D-VisionDraughts* com as peças brancas; e, na segunda, a situação foi invertida, isto é, o *APHID-Draughts* atuou com peças brancas e o *D-VisionDraughts* com peças pretas. Desta forma, o torneio contou com o total de 40 jogos. Assim, da mesma forma realizada no estudo de caso 1, não houve favorecimento de nenhum jogador, uma vez que poderia haver estados de tabuleiro que beneficiassem quem estivesse jogando com as peças pretas e vice-versa.

Partidas	Vitórias	Empates	Derrotas
Rodada I	75%	20%	5%
Rodada II	55%	40%	5%

Tabela 10 – Taxas obtidas pelo *APHID-Draughts* em jogos disputados contra o *D-VisionDraughts*

A partir dos resultados apresentados na Tabela 10 é possível observar que o *APHID-Draughts*, jogando com peças pretas, apresentou uma taxa de 75% de vitórias contra 5% do seu oponente, o *D-VisionDraughts*. Tal fato corresponde a uma taxa de 70% de superioridade do *APHID-Draughts* em relação ao seu oponente. A taxa de superioridade é calculada a partir da diferença entre as porcentagens de vitórias obtidas por cada

jogador. Ao considerar o *APHID-Draughts* jogando com peças brancas, ele apresentou uma taxa de 50% de superioridade em relação ao *D-VisionDraughts*.

Neste contexto, é possível concluir que o *APHID-Draughts* se sobressaiu jogando tanto com peças pretas quanto com peças brancas. A fim de comprovar esta vantagem do *APHID-Draughts* em relação ao *D-VisionDraughts*, foi empregado sobre estes resultados o método estatístico não-paramétrico *Wilcoxon signed rank test* 7.1.1, conforme detalhado a seguir.

7.2.2.1 Análise Estatística de Wilcoxon

Particularmente neste estudo de caso, como também no anterior 7.1, as amostras correspondem aos resultados alcançados pelos algoritmos (agentes automáticos), que são candidatos para resolver um determinado problema (jogo de Damas). Em outras palavras, o objetivo de usar *Wilcoxon* neste estudo de caso é verificar se o agente proposto pelo escopo global deste trabalho, *APHID-Draughts*, apresenta ou não um desempenho diferenciado em relação ao seu adversário *D-VisionDraughts*. Então, nas análises dos resultados consolidados na Tabela 10, o método de *Wilcoxon* assume, como ponto de partida, a seguinte hipótese nula H_0 : *APHID-Draughts* possui o mesmo nível de desempenho que o *D-VisionDraughts*.

De acordo com o método de *Wilcoxon*, os seguintes passos devem ser seguidos a fim de provar que H_0 deve ser rejeitada (NETO; JULIA; CAEXETA, 2014):

Passo 1: Dado T como a menor soma dos *ranks* (ordenações) positivas e negativas dentre as diferenças dos pares de amostras, isto é, $T = \min\{R^+, R^-\}$ (particularmente neste trabalho, R^+ representa a soma dos *ranks* em que o *APHID-Draughts* superou o *D-VisionDraughts*, enquanto R^- é a soma dos *ranks* em que o *APHID-VisionDraughts* foi superado pelo *D-VisionDraughts*). Uma apropriada tabela estatística ou ferramenta deve ser utilizada para determinar o teste estatístico, valor crítico ou *p-value* (dado provido pelo teste);

Passo 2: Rejeita-se a hipótese nula H_0 se o teste estatístico $\leq$ valor crítico ou se *p-value* $\leq \alpha$ (nível de significância). Quanto mais baixo é o valor de α , mais forte é a evidência contra a hipótese nula H_0 .

A Tabela 11 apresenta os testes estatísticos de *Wilcoxon* aplicados sobre os resultados dos jogos apresentados na Tabela 10, onde a seguinte pontuação foi definida: 2 pontos para vitória, 1 ponto para empate e 0 ponto para derrota. Analogamente ao estudo de caso 1, os valores R^+ , R^- e *p-value* foram também computados utilizando o software de estatística *SPSS*.

Torneio	R+	R-	p-value
<i>APHID-Draughts</i> x <i>D-VisionDraughts</i>	377	29	0,000006

Tabela 11 – Método *Wilcoxon signed rank test* aplicado aos resultados do torneio entre o *APHID-Draughts* e o *D-VisionDraughts*

Como é possível observar, o *APHID-Draughts* se mostrou significativamente superior ao *D-VisionDraughts* com um alto nível de significância $\alpha = 0.01$ ou 99%. Tal fato representa que há 1% de chance da hipótese nula H_0 (equivalência de desempenho entre o *APHID-Draughts* e o *D-VisionDraughts*) ocorrer na base de dados (amostra) analisada. Em outras palavras, os dados apresentados na Tabela 11 rejeitam fortemente H_0 , o que significa que os resultados alcançados pelo *APHID-Draughts*, nos jogos de torneio apresentados na Tabela 10, são de fato superiores em relação ao *D-VisionDraughts*.

A boa performance do algoritmo *APHID-Draughts* é justificada por suas características que não o limitam em uma arquitetura de memória distribuída e pela definição adequada da janela de busca, que permite controlar a sobrecarga de busca e, consequentemente, melhora o *look-ahead* do jogador quando os jogos são executados em ID.

7.2.2.2 Análise da Qualidade dos Movimentos com relação ao Cake

Assim como efetuado no estudo de caso 1, os experimentos realizados nesta subseção tem como proposta a avaliação do nível da qualidade do processo de tomada de decisão dos jogadores automáticos avaliados - no caso, *APHID-Draughts* e *D-VisionDraughts*. Tal avaliação é baseada na taxa de coincidência entre os movimentos escolhidos por cada agente e os movimentos que o agente *Cake* poderia realizar nas mesmas situações.

Dessa forma, a Tabela 12 mostra os resultados comparativos nas situações em que ambos os agentes operaram com peças pretas (primeira linha da Tabela) e com peças brancas (segunda linha). Coerentemente com os resultados anteriores, estes resultados confirmam a superioridade do *APHID-Draughts* em relação ao *D-VisionDraughts*. De fato, o *APHID-Draughts* jogando com peças pretas obteve 66.53% contra 30,03% do *D-VisionDraughts* e, jogando com peças brancas, apresentou uma taxa de coincidência de 43,9% contra 33,89% do seu oponente.

Posição do agente no jogo	<i>APHID-Draughts</i>	<i>D-VisionDraughts</i>
Peças Pretas	66.53%	30.03%
Peças Brancas	43.9%	33.89%

Tabela 12 – Taxas de coincidência de movimentos com o *Cake*

7.2.3 Considerações finais sobre o estudo de caso 2

Este estudo de caso foi realizado a fim de verificar o desempenho do agente *APHID-Draughts* - proposto no escopo global deste trabalho. Desta forma, foram efetuados experimentos considerando o agente *D-VisionDraughts* que atua em ambiente de alto desempenho utilizando a versão com abordagem síncrona de paralelismo YBWC. Tanto a implementação quanto os experimentos do *APHID-Draughts* e do *D-VisionDraughts* foram realizados em arquitetura de memória distribuída. De fato, a principal intenção foi comparar o desempenho das abordagens assíncrona e síncrona de paralelismo aplicadas a um mesmo problema. Nesta direção, foram considerados os seguintes parâmetros: tempo de treinamento, taxas de vitória e taxa de coincidência entre a escolha de movimentos executados por cada agente (*APHID-Draughts* e *D-VisionDraughts*) e aqueles que o *Cake* realizaria nas mesmas situações. Os resultados obtidos apontaram que a abordagem assíncrona é mais adequada que a versão síncrona para agentes de Damas quando implementadas em arquitetura de memória distribuída.

Neste contexto, salienta-se que a implementação do *APHID-Draughts* além de ter permitido uma análise comparativa prática entre as abordagens assíncrona e síncrona, possibilitou a identificação de pontos fortes e limitações que podem ser trabalhadas no algoritmo de modo a obter maior eficácia. Adicionalmente, foi possível identificar que a metodologia de trabalho do APHID é mais adequada do que a abordagem do YBWC (síncrono), para uma arquitetura de memória distribuída.

Finalmente, os resultados obtidos da análise comparativa realizada nesta seção envolvendo os jogadores *APHID-Draughts* e *D-VisionDraughts*, juntamente com a contribuição da investigação conceitual acerca dos algoritmos YBWC e APHID (detalhada no capítulo 5), permitiram identificar pontos de otimização para a nova versão distribuída do algoritmo Alfa-Beta a partir das percepções dos comportamentos das abordagens síncrona (YBWC) e assíncrona (APHID). Tais fatos culminaram no cumprimento do seguinte objetivo executado no contexto de Doutorado no qual o presente trabalho está inserido: proposta de uma nova versão de paralelização assíncrona do algoritmo Alfa-Beta que tire proveito dos pontos positivos das abordagens síncrona do YBWC e assíncrona do APHID, denominada ADABA.

8 Conclusão

Este trabalho apresentou uma investigação à respeito dos algoritmos de busca distribuídos YBWC e APHID implementados nos jogadores automáticos de Damas produzidos pela equipe do presente trabalho. Nesta direção, primeiramente, focou-se na análise comparativa conceitual entre os principais aspectos propiciados pelas abordagens de ambos algoritmos (síncrona, pelo YBWC; assíncrona, pelo APHID). Todo este estudo realizado - o qual foi uma extensão da análise efetuada no escopo global de Doutorado no qual este trabalho está alocado - permitiu a identificação de pontos fortes e limitações dos dois algoritmos investigados.

Na sequência, os esforços concentraram-se na produção de um sistema automático com a finalidade de propiciar uma avaliação da qualidade da tomada de decisão de agentes jogadores de Damas combinando uma grande agilidade e precisão, de modo a substituir o método manual oneroso utilizado em trabalhos anteriores. Desta forma, foi utilizado um estudo de caso já consolidado na literatura para validar tanto a velocidade de tal sistema quanto as informações estatísticas produzidas por ele. Este estudo de caso envolveu os seguintes agentes: *VisionDraughts* e *D-VisionDraughts*.

A partir do sistema automático implementado e validado, foi possível complementar a comparação prática entre os algoritmos YBWC e APHID realizada no contexto global de doutorado. Para tanto, foram utilizados os sistemas automáticos jogadores de Damas *D-VisionDraughts* - baseado no YBWC - e *APHID-Draughts* - baseado no APHID. Este último jogador foi implementado no escopo global de doutorado. Nesta direção, o presente trabalho contribuiu com a extração das taxas de qualidade de jogadas dos referidos jogadores (baseada na coincidência de movimentos com o agente *Cake*). Os resultados dos experimentos realizados em arquitetura de memória distribuída mostraram que o APHID apresentou melhor desempenho neste tipo de arquitetura.

Portanto, os objetivos propostos pelo presente trabalho foram cumpridos, servindo de base para cumprimento dos objetivos da referida pesquisa de Doutorado cuja meta é propor um novo algoritmo de busca distribuído assíncrono denominado ADABA, o qual resulta da tentativa de eliminação dos gargalos e do aproveitamento dos benefícios dos referidos algoritmos distribuídos. Assim sendo, pretende-se continuar as atividades aqui realizadas em um contexto de mestrado, de forma a aprimorar a versão distribuída do ADABA construída e implementar agentes jogadores de Damas baseados em tal versão e em *Deep Learning*.

Referências

- BARCELOS, A. R. A. *D-VisionDraughts: uma rede neural jogadora de damas que aprende por reforço em um ambiente de computação distribuída*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, 2011. Citado 2 vezes nas páginas 28 e 29.
- BARCELOS, A. R. A.; JULIA, R. M. da S.; MATIAS, R. D-visiondraughts: a draughts player neural network that learns by reinforcement in a high performance environment. In: *ESANN*. [S.l.: s.n.], 2011. Citado 2 vezes nas páginas 21 e 39.
- BROCKINGTON, M. G. *KEYANO Unplugged – The Construction of an Othello Program*. 1997. Citado na página 24.
- BROCKINGTON, M. G. *Asynchronous Parallel Game-Tree Search*. Tese (Doutorado), Edmonton, Alta., Canada, 1998. AAINQ29023. Citado 3 vezes nas páginas 24, 42 e 53.
- BROCKINGTON, M. G.; SCHAEFFER, J. Aphid: Asynchronous parallel game-tree search. *Journal of Parallel and Distributed Computing*, v. 60, n. 2, p. 247 – 273, 2000. ISSN 0743-7315. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0743731599916003>>. Citado 6 vezes nas páginas 7, 20, 21, 23, 30 e 38.
- CAIXETA, G. S. *VisionDraughts - um sistema de aprendizagem de jogos de damas baseado em redes neurais, diferenças temporais, algoritmos eficientes de busca em árvores e informações perfeitas contidas em bases de dados*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, 2008. Citado na página 28.
- CAIXETA, G. S.; JULIA, R. M. da S. A draughts learning system based on neural networks and temporal differences: The impact of an efficient tree-search algorithm. In: _____. *Advances in Artificial Intelligence - SBIA 2008: 19th Brazilian Symposium on Artificial Intelligence Salvador, Brazil, October 26-30, 2008. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 73–82. ISBN 978-3-540-88190-2. Disponível em: <http://dx.doi.org/10.1007/978-3-540-88190-2_13>. Citado 2 vezes nas páginas 17 e 19.
- CAMPOS, P.; LANGLOIS, T. Abalearn: Efficient self-play learning of the game abalone. In: *INESC-ID, Neural Networks and Signal Processing Group*. [S.l.: s.n.], 2003. Citado na página 12.
- DERRAC, J. et al. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, v. 1, n. 1, p. 3 – 18, 2011. ISSN 2210-6502. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2210650211000034>>. Citado 2 vezes nas páginas 64 e 65.
- DUARTE, V. A. R. et al. Mp-draughts: Unsupervised learning multi-agent system based on mlp and adaptive neural networks. In: *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2015. p. 920–927. ISSN 1082-3409. Citado 3 vezes nas páginas 17, 18 e 66.

- FELDMANN, R. et al. Distributed game tree search. In: _____. *Parallel Algorithms for Machine Intelligence and Vision*. New York, NY: Springer New York, 1990. p. 66–101. ISBN 978-1-4612-3390-9. Disponível em: <http://dx.doi.org/10.1007/978-1-4612-3390-9_3>. Citado 3 vezes nas páginas 20, 29 e 53.
- FIERZ, M. C. *Cake informations and Technical report*. [S.l.], 2010. Acessado em 18 de dezembro de 2017. Disponível em: <<http://www.fierz.ch/checkers.htm>>. Citado 4 vezes nas páginas 13, 25, 27 e 28.
- GILBERT, E. *KingsRow Technical report*. [S.l.], 2000. Acessado em 18 de dezembro de 2017. Disponível em: <<http://edgilbert.org/Checkers/KingsRow.htm>>. Citado na página 25.
- HERIK, H. van den; UITERWIJK, J. W.; RIJSWIJCK, J. van. Games solved: Now and in the future. *Artificial Intelligence*, v. 134, n. 1, p. 277 – 311, 2002. ISSN 0004-3702. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0004370201001527>>. Citado na página 12.
- MANOHARARAJAH, V. *Parallel alpha-beta search on shared memory multiprocessors*. Dissertação (Mestrado) — University of Toronto, 2001. Citado na página 20.
- MILLINGTON, I. *Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. ISBN 0124977820. Citado na página 51.
- NETO, H.; JULIA, R.; CAEXETA, G. Ls-visiondraughts: improving the performance of an agent for checkers by integrating computational intelligence, reinforcement learning and a powerful search method. *Appl. Intell.*, vol. 41, no. 2, p. 225–250, 2014. Citado 5 vezes nas páginas 17, 18, 65, 66 e 70.
- NEUMANN, J.; MORGENSTERN, O. *Theory of games and economic behavior*. 3. ed.. ed. Princeton, NJ: Princeton Univ. Press, 1953. Disponível em: <http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+021850372&sourceid=fbw_bibsonomy>. Citado na página 12.
- RUSSELL, S.; NORVIG, P. *Inteligência Artificial - Uma abordagem Moderna*. 2nd. ed. [S.l.]: Campus, 2004. Citado 3 vezes nas páginas 12, 19 e 41.
- SCHAEFFER, J. et al. Checkers is solved. *Science*, 2007. Citado 2 vezes nas páginas 28 e 57.
- SCHAEFFER, J. et al. CHINOOK: The world man-machine checkers champion. *The AI Magazine*, v. 16, n. 1, p. 21–29, 1996. Citado 4 vezes nas páginas 13, 25, 27 e 28.
- SUTTON, R. S. Learning to predict by the methods of temporal differences. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 3, n. 1, p. 9–44, aug 1988. ISSN 0885-6125. Disponível em: <<http://dx.doi.org/10.1023/A:1022633531479>>. Citado na página 29.
- TOMAZ, L. B. P. *D-MA-Draughts: um sistema multiagente jogador de damas automático que atua em um ambiente de alto desempenho*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, 2013. Citado 3 vezes nas páginas 28, 29 e 31.

- TOMAZ, L. B. P.; JULIA, R. M. S.; BARCELOS, A. R. A. Improving the accomplishment of a neural network based agent for draughts that operates in a distributed learning environment. In: *2013 IEEE 14th International Conference on Information Reuse Integration (IRI)*. [S.l.: s.n.], 2013. p. 262–269. Citado 10 vezes nas páginas 7, 8, 12, 17, 28, 29, 62, 63, 65 e 66.
- TOMAZ, L. B. P.; JULIA, R. M. S.; DUARTE, V. A. A multiagent player system composed by expert agents in specific game stages operating in high performance environment. *Applied Intelligence*, p. 1–22, 2017. ISSN 1573-7497. Disponível em: <http://dx.doi.org/10.1007/s10489-017-0952-x>. Citado na página 17.
- URA, A.; TSURUOKA, Y.; CHIKAYAMA, T. Dynamic prediction of minimal trees in large-scale parallel game tree search. *Journal of Information Processing*, v. 23, n. 1, p. 9 – 19, 2015. Citado na página 24.
- URA, A.; YOKOYAMA, D.; CHIKAYAMA, T. Two-level task scheduling for parallel game tree search based on necessity. *Information and Media Technologies*, v. 8, n. 1, p. 32 – 40, 2013. Citado na página 24.
- WALKER, M. A. An application of reinforcement learning to dialogue strategy in a spoken dialogue system for email. *Artificial Intelligence Research* 12, p. 387–416, 2000. Citado na página 12.