UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

BTT-GO: UM AGENTE JOGADOR DE GO COM BUSCA MONTE-CARLO APRIMORADA COM TABELA DE TRANSPOSIÇÃO E MODELO BRADLEY-TERRY

ELDANE VIEIRA JÚNIOR

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELDANE VIEIRA JÚNIOR

BTT-GO: UM AGENTE JOGADOR DE GO COM BUSCA MONTE-CARLO APRIMORADA COM TABELA DE TRANSPOSIÇÃO E MODELO BRADLEY-TERRY

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Inteligência Artificial.

Orientadora:
Prof[a]. Dr[a]. Rita Maria da Silva Julia

Uberlândia, Minas Gerais
2014

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU

V658b Vieira Júnior, Eldane, 1984-
BTT-Go : um agente jogador de Go com busca Monte-Carlo aprimorada com tabela de transposição e modelo Bradley-Terry / Eldane Vieira Júnior. - 2014.
91 p. : il.

Orientadora: Rita Maria da Silva Julia.
Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Jogos eletrônicos - Teses. 3. Jogos por computador - Teses. 4. Monte Carlo, Método de - Teses. I. Julia, Rita Maria da Silva. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada "**BTT-Go: um Agente Jogador de Go com Busca Monte-Carlo Aprimorada com Tabela de Transposição e Modelo Bradley-Terry**" por **Eldane Vieira Júnior** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 21 de Março de 2014

Orientadora: _____

Profª. Drª. Rita Maria da Silva Julia
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Profª. Drª. Sandra Aparecida de Amo
Universidade Federal de Uberlândia

_____

Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho
Universidade de São Paulo - São Carlos

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Março de 2014

Autor:	**Eldane Vieira Júnior**
Título:	**BTT-Go: um Agente Jogador de Go com Busca Monte-Carlo Aprimorada com Tabela de Transposição e Modelo Bradley-Terry**
Faculdade:	**Faculdade de Ciência da Computação**
Grau:	**Mestrado**

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão de cópias deste documento para propósitos exclusivamente acadêmicos, desde que o autor seja devidamente informado.

_____

Autor

O AUTOR RESERVA PARA SI QUALQUER OUTRO DIREITO DE PUBLICAÇÃO DESTE DOCUMENTO, NÃO PODENDO O MESMO SER IMPRESSO OU REPRODUZIDO, SEJA NA TOTALIDADE OU EM PARTES, SEM A PERMISSÃO ESCRITA DO AUTOR.

©Todos os direitos reservados a Eldane Vieira Júnior

Dedicatória

Ao meu pai Eldane, minha mãe Verginia e minhas irmã Mirian e Maísa.

Agradecimentos

Agradeço...

Aos meus pais por todo o apoio que me oferecem na minha formação tanto profissional quanto como ser humano.

Aos meus amigos e familiares que durante todo este percurso de formação acadêmica, tornaram momentos difíceis em momentos mais agradáveis. Obrigado Júlio, Fidel, Mirian, Maísa, Virgínia, Aninha, Regina ...

A todos os professores que participaram da minha formação e que serviram de motivação para cumprir mais esta etapa na minha formação profissional.

Principalmente, à Professora Rita por toda paciência e confiança a mim destinadas, compartilhando comigo todo seu profissionalismo que possibilitou o sucesso deste Mestrado.

A CAPES, pelo apoio financeiro concedido para realização deste trabalho.

De um modo geral agradeço a todos que contribuíram, de alguma forma, para a concretização deste trabalho.

"Para obter algo que nunca teve, é preciso fazer algo que nunca fez"
(Chico Xavier)

Resumo

O jogo de Go é, atualmente, um dos grandes desafios para a área de Inteligência Artificial, pois este reúne uma série de características que impedem o sucesso de técnicas que foram bem sucedidas em outros jogos. Entre as características desafiadoras do jogo, está o alto nível de complexidade que inviabiliza o uso de técnicas que necessitam explorar, ao máximo, seu espaço de busca.

Diante deste desafio, neste trabalho de Mestrado foi criado o agente jogador de Go denominado BTT-Go. Este agente foi criado a partir de outro jogador, chamado Fuego, que utiliza uma das poucas técnicas que proporcionam bons ganhos aos jogadores automáticos de Go: o algoritmo de busca Monte-Carlo. O Fuego possui uma aprendizagem essencialmente supervisionada, uma vez que seu processo de busca pelo melhor movimento baseado, exclusivamente, nas simulações Monte-Carlo, em avaliações heurísticas de tabuleiros e em bases de dados de início de jogo (*opening book*). Assim sendo, o objetivo do presente trabalho é produzir um agente inspirado no Fuego que se mantenha bastante competitivo apesar de apresentar um nível de supervisão inferior ao do citado jogador automático. Para atingir este objetivo, o BTT-Go foi desenvolvido em três versões: na primeira delas, foi usada uma Tabela de Transposição, que serve como um repositório de dados previamente processados. Desta maneira, é possível reduzir a supervisão da quantidade de simulações efetuadas pelo algoritmo de busca Monte-Carlo, avaliação que permite substituir, em alguns casos, a avaliação *prior-knowledge* herdada do Fuego. A segunda versão do BTT-Go consiste na aplicação, durante a etapa final da busca Monte-Carlo, de uma técnica bayesiana inspirada no modelo Bradley-Terry. Esta técnica permite predizer a melhor jogada através da avaliação do tabuleiro. Esta avaliação é feita em função de alguns atributos, que servem para dizer o quanto uma jogada é boa. No Fuego esta etapa tem os movimentos gerados, unicamente, por políticas. Na terceira versão é feita a associação da primeira com a segunda versão para o funcionamento em conjunto das técnicas aplicadas.

Uma vez concluídas as três versões do agente BTT-Go, testes foram realizados em tabuleiros de tamanho 9x9, 13x13 e 19x19. Nestes testes observou-se que com a aplicação da Tabela de Transposição reduziu-se a supervisão no agente. Contudo, ocorreu uma leve queda no percentual de vitórias em tabuleiros maiores (13x13 e 19x19), quando comparado ao jogador Fuego, mas mesmo assim o agente se manteve competitivo. Contudo, com a aplicação da técnica inspirada no modelo Bradley-Terry observou-se um aumentou na competitividade do jogador mesmo em tabuleiros maiores (13x13 e 19x19), chegando em alguns casos ser melhor que o agente Fuego.

Portanto, a criação do jogador BTT-Go proporcionou a redução da supervisão, obtida tanto pelo uso da Tabela de Transposição quanto pela técnica bayesiana inspirada no modelo Bradley-Terry. Também proporcionou ao agente um aumento da acuidade na geração de movimentos no processo de busca.

Palavras chave: go, busca monte-carlo, modelo bradley-terry, simulações monte-carlo.

Abstract

The game of Go is, nowadays, one of the greatest challenge in the Artificial Intelligence area, since this game has a set of characteristics that prevents the success application of techniques, which has been very successful in other games. In this set of characteristics there is the high level of complexity, which prevents it from the use of techniques that require the maximum exploration of its search state-space.

In this thesis is described the development of a player agent for the game of Go named BTT-Go. This agent was created from another one named Fuego, which uses one of the few techniques that had provided improvement to the automatic players of Go: the Monte-Carlo Tree Search algorithm. The player Fuego uses a supervised learning, once its search method is based, exclusively, on Monte-Carlo simulations, heuristics board evaluations and database, which contains data about the game start (*opening book*).This way, the objective of this thesis is to produce a competitive agent in spite of the supervision reduction, which is much less then the supervision used by the agent Fuego. To achieve this objective, BTT-Go was developed in three versions: in the first, the agent uses a Transposition Table, which is a repository of data processed previously. This way, it is possible to reduce the simulation supervision by its reduction, and in some situations, the agent uses the data from the table instead of using the Fuego *prior knowledge* evaluation. The second version of BTT-Go consists in the application, in the final stage of the Monte-Carlo search algorithm, of a bayesian technique inspired on Bradley-Terry model. This technique predicts the best move by a board evaluation. This evaluation is done considering some features that describes how good a move is. In this stage, the agent Fuego uses policies to indicate which move should be played. The BTT-Go third version was created by the combination of the first and the second versions, in a way that the techniques used can work together without any loss.

Once the development of the three version was completed, it was performed some experiments in different board sizes (9x9, 13x13 and 19x19). In these experiments, it was observed that the use of Transposition Table reduced the agent supervision. Although, there was a little reduction in its winning rate in large boards (13x13 and 19x19), comparing it to Fuego, nevertheless BTT-Go is still a competitive player. It was also observed that the technique inspired on Bradley-Terry model increased the competitiveness of the agent in large boards (13x13 and 19x19), and in some situation it was better than the agent Fuego.

Therefore, the development of the player BTT-Go has provided a supervision reduction by the use of Transposition Table and by the use of bayesian technique inspired on Bradley-Terry model, and also a increase of the acuity in the moves generation during the search process.

Keywords: go, monte-carlo tree search, bradley-terry model, monte-carlo simulations.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BR	Brasil
BT	Bradley Terry
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
FLAIRS	*Florida Artificial Intelligence Research Society*
MCTS	Monte Carlo Tree Search
MG	Minas gerais
TT	Tabela de transposição

Capítulo 1

Introdução

Jogos têm sido uma das principais áreas de pesquisa em Inteligência Artificial por estarem ligados a problemas do mundo real que geram situações inesperadas e imprevisíveis. Alguns exemplos destes problemas consistem naqueles provenientes da interação homem-máquina como os atendentes eletrônicos de empresas de prestação de serviços e o problema referente ao controle de tráfego urbano visando diminuir os congestionamentos [Walker 2000]. Dessa maneira, a criação de jogadores automáticos possuem desafios tanto técnicos quanto práticos para o tratamento de problemas como os apresentados anteriormente.

A criação de jogadores automáticos tem sido uma realidade em diferentes jogos com o propósito de estudo de diferentes técnicas. Entre esses estudos está o trabalho feito com o jogo de Xadrez em que foi estudada a aplicação da força bruta no jogo e o uso de conhecimento específico do mesmo, com o propósito de explorar parte do espaço de busca do jogo de Xadrez [Shannon and Hsu 1949]. Um outro exemplo está ligado ao jogo de Gamão, em que foi estudada a aplicação da técnica de Diferenças Temporais [Tesauro 1995]. No jogo de Damas, por sua vez, dentre outras técnicas foi aplicado Algoritmos Genéticos [Goldberg and Holland 1988] para gerar um conjunto de características do jogo [Castro Neto 2007].

O jogo de Go tem sido um grande desafio na área de Inteligência Artificial devido a elevada dimensão do seu espaço de estados, conforme pode ser observado na Figura 1.1 [Allis 1994], onde a complexidade é medida de duas maneiras, pelo espaço de estados e pelo tamanho da árvore de busca do jogo. Para que um jogador lide com eficiência com tal dimensão, ele precisa ser capaz de definir complexas estratégias de jogo, fato que torna o jogo de Go um importante campo para pesquisas em agentes automáticos.

O trabalho apresentado nesta dissertação relata a criação do programa jogador de Go, chamado BTT-Go, que foi criado através de modificações no jogador chamado Fuego [Enzenberger et al. 2010]. O Fuego utiliza o algoritmo de busca baseado em simulações Monte-Carlo (MC) [Ross 2006] de jogos efetuadas de acordo com a política *Upper Confidence Trees* (UCT) [Coulom 2007b]. Tal política é combinada com os algoritmos de

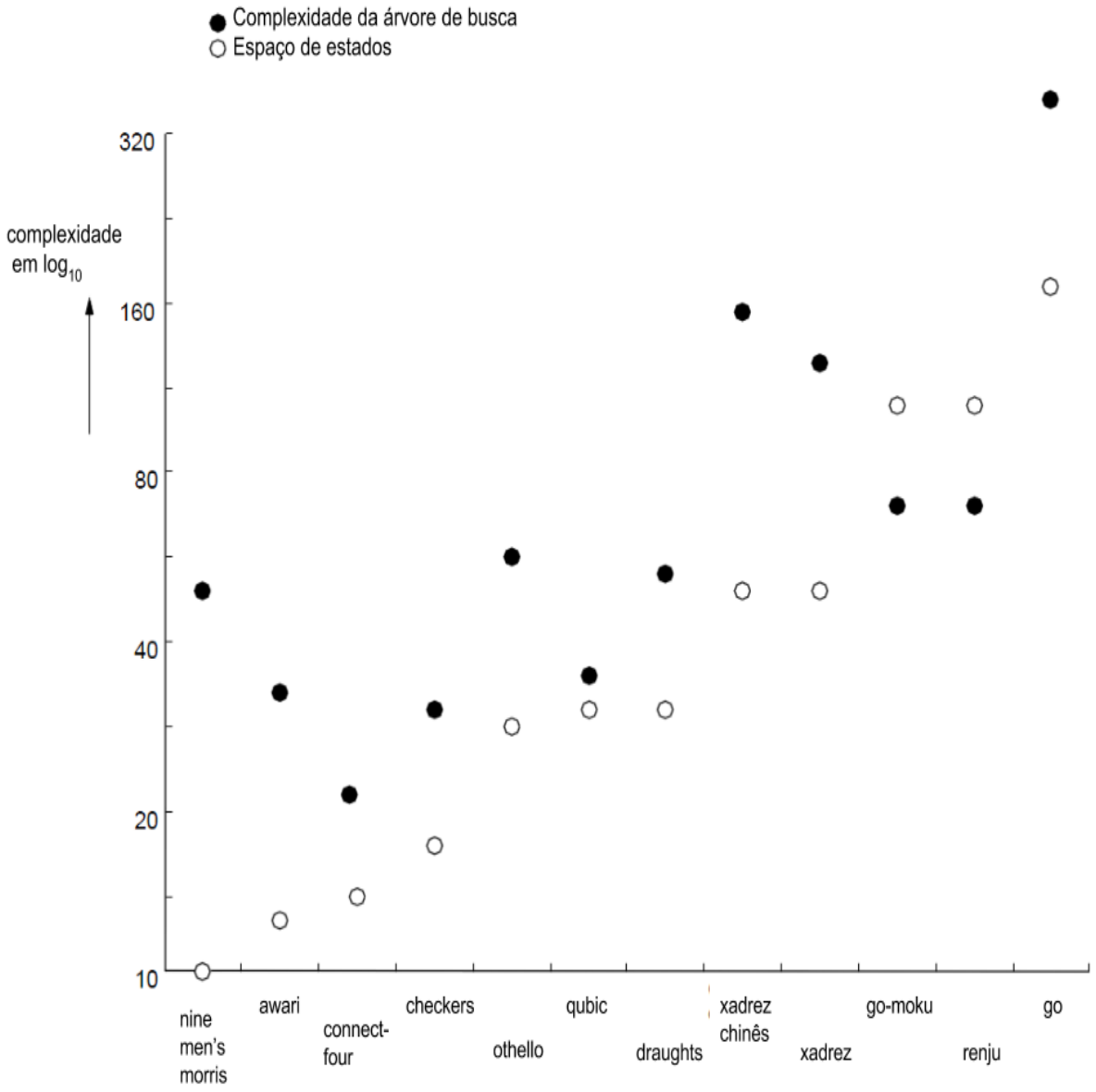

Figura 1.1: Complexidade de jogos.

atualização de valores dos nós da árvore de busca chamados *Rapid Action Value Estimation* (RAVE) [Gelly and Silver 2011] e *Monte-Carlo Tree Search* (MCTS) [Brügmann 1993]. Os movimentos de início de jogo do Fuego são definidos através de bases de dados chamadas de *opening book*. A partir daí, os movimentos são escolhidos pela busca MC baseada na política UCT. Os nós candidatos a integrar a árvore de busca concorrem entre si contando com valores iniciais que são providos por heurísticas de *prior knowledge*. As simulações de jogadas efetuadas durante a busca pelo melhor movimento do Fuego dividem-se nas etapas de *construção da árvore*, que é baseada na avaliação dos estados, e na fase de *play-out*, baseada em regras heurísticas conhecidas como *política de play-out*. Dessa forma, o jogador Fuego, base para este trabalho, tem sido concebido de forma extremamente supervisionada, uma vez que seu processo de busca pelo melhor movimento é baseado em simulações e heurísticas diversas. O BTT-Go tem dois objetivos princi-

pais com relação ao Fuego: otimizar o seu processo de busca, melhorando a acuidade das avaliações dos tabuleiros; reduzir seu caráter supervisionado, reduzindo o número de simulações. Ambas as metas são obtidas através do uso de uma Tabela de Transposição (TT) e do modelo Bradley-Terry (BT), sendo que a TT é utilizada como repositório dos estados previamente avaliados e o modelo BT é usado para substituir a *política de play-out*. Dessa forma, o BTT-Go consegue aumentar a precisão da avaliação dos estados utilizando o histórico de avaliações armazenados na TT e utilizando a heurística mais precisa do modelo BT durante as fases de *play-out*. O uso da TT também atenua o nível de supervisão do Fuego, uma vez que suas informações permitem reduzir a frequência de uso da heurística do *prior-knowledge* durante a construção da árvore, além de permitir a interrupção do *play-out*, fato que reduz a quantidade de simulações MC relativas a tal fase.

Pela análise do fator de ramificação e do espaço de estados do jogo de Go, respectivamente ± 360 e 10^{160}, disponíveis na Tabela 1 [Campos and Langlois 2003], fica evidente que aplicar alguma técnica que traga benefícios como a redução de processamento, terá que lidar com o problema da complexidade. Sendo assim, torna-se interessante o uso de uma Tabela de Transposição (TT) como repositório de estados já avaliados, uma vez que a árvore de busca MC pode ter múltiplos nós para uma mesma posição. Convém ressaltar que o uso de uma TT apresenta-se como uma alternativa interessante a ser aplicada em agentes para Go, uma vez que a ocorrência de transposição (reaparição de um mesmo estado durante a busca) é bastante elevada na dinâmica do jogo [van der Werf 2005].

Tabela 1.1: Comparação do fator de ramificação e espaço de estados de alguns jogos.

Game	**Fator de ramificação**	**Espaço de estados**
Xadrez	30 – 40	10^5
Damas	8 – 10	10^{17}
Gamo	± 420	10^{20}
Othello	± 5	$< 10^{30}$
Go 19x19	± 360	10^{160}
Abalone	± 80	$< 3^{61}$

A técnica bayesiana que é aplicada neste trabalho de mestrado é inspirada no modelo BT [Coulom 2007a] que visa incorporar "inteligência" na etapa de *play-outs* do algoritmo MCTS. Dessa forma, ao invés de usar as políticas de *play-out* do Fuego, que definem padrões de tabuleiro, o BTT-Go vai utilizar o modelo BT. A técnica BT permite predizer o melhor movimento a ser escolhido, isso é possível devido a uma análise de atributos que avaliam o estado do tabuleiro quando um movimento é realizado, gerando assim uma nota que descreve a qualidade do movimento. É importante salientar que uma variação da técnica BT foi aplicada no jogador de Go Crazy Stone [Coulom 2007a], que é um programa jogador com código fechado, distinto do jogador Fuego. Contudo, a técnica inspirada no modelo BT será usada no BTT-Go diferentemente da forma como foi aplicada no Crazy

Stone, visto que este aplica a técnica para substituir a seleção das simulações feitas pelo algoritmo MCTS, enquanto o BTT-Go aplica a técnica na etapa de *play-outs*. Outra diferença está na forma de avaliar o movimento, uma vez que o Crazy Stone gera as avaliações em função de padrões, enquanto o BTT-Go gera as avaliações independente do que for apresentado ao módulo avaliador. Para a aplicação do modelo BT no BTT-Go, o agente utiliza um conjunto de atributos que visam extrair conhecimento a ser aplicado durante a etapa de *play-out* do algoritmo MCTS.

A título comparativo, o agente BTT-Go foi concebido em três versões: a primeira associa uma TT ao algoritmo MCTS durante toda a busca MC; a segunda substitui a *política de play-out* pelo modelo BT ; já a terceira versão corresponde à uma associação das duas versões anteriores, usando uma TT durante toda a busca MC e usando o modelo BT durante o *play-out*.

1.1 Objetivos e contribuição

Diante da complexidade referente ao jogo de Go, este trabalho tem o objetivo de incorporar melhorias ao algoritmo de busca Monte-Carlo do agente Fuego, além de reduzir a supervisão do jogador.

Para tanto, a criação do jogador BTT-Go, como foi apresentado, insere no referido processo de busca dois recursos adicionais: a TT e a técnica de avaliação de estados baseada no modelo BT.

O objetivo com o uso da TT é otimizar o processo de busca do jogador, uma vez que estados avaliados anteriormente, e que tenham suas avaliações armazenadas na TT, não serão processados novamente, possibilitando a redução da supervisão do agente jogador. O modo de uso da TT, que visa atuar em conformidade com os algoritmos MCTS e RAVE, foi criado e aplicado, exclusivamente, no BTT-Go, ou seja, não existe nenhum outro jogador automático de Go que utiliza TT com a estratégia de interromper uma etapa com alto custo computacional que é a etapa de simulações. Com a referida estratégia, foi comprovado por testes (apresentados no Capítulo 5) que a interrupção de simulações trouxe ganhos ao agente quanto a redução de sua supervisão.

Em relação a segunda técnica aplicada (modelo BT) usada para predição de movimento, especificamente, na seleção de um movimento durante a etapa de *play-out*, o modelo BT introduziu "inteligência" ao algoritmo de busca MCTS, através da avaliação do estado do tabuleiro que acontece via atributos. A conquista deste objetivo foi comprovada através de experimentos (Capítulo 5) que consistem em jogos realizados contra outros agentes e que mostram ganho de competitividade. Na aplicação do modelo BT foi criado um atributo chamado *Elemento Tático*, usado exclusivamente no BTT-Go, que visa emular o comportamento imprevisível de um jogador humano, especialmente no que se refere a mudança de estratégias. Esse atributo foi avaliado por testes e apresentou

importantes ganhos ao agente BTT-Go.

Portanto, a criação do jogador BTT-Go com o uso das duas técnicas apresentadas conseguiu mais acuidade das avaliações e redução da supervisão, decorrentes da redução de simulações MC, pelo uso do histórico de avaliações da TT (substitui o uso do *prior knowledge*) e pela aplicação do modelo BT.

Os resultados deste trabalho de Mestrado originaram na publicação de um artigo [Junior and Julia 2014] na conferência *The 27th International Conference of the Florida Artificial Intelligence Research Society* (FLAIRS-27) listado no Qualis da CAPES com classificação B1.

1.2 Organização da dissertação

Esta dissertação está organizada da seguinte maneira:

- Capítulo 2, apresenta o referencial teórico utilizado na criação do agente BTT-Go.
- Capítulo 3, relata trabalhos vinculados ao tema de criação de agentes jogadores de Go e relacionados ao trabalho apresentado nesta dissertação.
- Capítulo 4, apresenta os detalhes da criação do agente BTT-Go que ocorreu em três versões, apresentando a arquitetura do agente que ilustra o modo de funcionamento e fluxo de sua execução.
- Capítulo 5, apresenta os resultados obtidos em jogos realizados entre o BTT-Go e outros jogadores automáticos de Go. Neste capítulo, também é feita uma análise comparativa que mostra os ganhos proporcionados com o trabalho apresentado nesta dissertação.
- Capítulo 6, apresenta as conclusões diante dos resultados obtidos e as próximas atividades a serem desenvolvidas.

Capítulo 2

Fundamentos Teóricos

Este capítulo apresenta o referencial teórico utilizado na criação do agente BTT-Go. As seções neste capítulo apresentam conceitos relacionados ao jogo do Go, à busca MCTS, à transposição e ao modelo BT.

2.1 Jogo de Go

O jogo de Go foi criado há aproximadamente 4000 anos na China, sendo também conhecido como Igo no Japão. Antigamente, na China, o jogo de Go era tratado como uma arte em que os senhores da sociedade o aprendiam como também aprendiam pintura e poesia. Dessa maneira, o jogo de Go se tornou bastante popular na China, desenvolvendo jogadores muito habilidosos [Harrison 2010]. Com o passar do tempo, o jogo de Go se tornou muito popular em diversos outros países.

O jogo de Go é jogado em diversos tamanhos de tabuleiros, sendo que a dimensão máxima é 19x19, como é ilustrado pela Figura 2.1. Participam do jogo dois competidores, sendo que um joga com as peças pretas e o outro com as peças brancas. As jogadas são realizadas alternadamente, e consistem em colocar uma peça em alguma intersecção livre do tabuleiro, sendo que na fase inicial do jogo o tabuleiro está completamente vazio. As peças são removidas do tabuleiro somente quando são capturadas pelo adversário, ou seja, depois de efetuada uma jogada não é possível mover uma peça para outra posição no tabuleiro.

A captura de uma peça no jogo de Go consiste em remover todas as liberdades da mesma, sendo que liberdade consiste em uma posição livre e adjacente a referida peça. A Figura 2.2 ilustra um exemplo em que a peça está quase sendo capturada, pois a sua liberdade tem valor 1. Um jogador pode capturar um grupo de peças do adversário, cercando completamente com suas peças um conjunto de peças adjacentes do adversário, como mostra a Figura 2.3.

Outros conceitos importantes que retratam situações particulares do jogo de Go são apresentados a seguir:

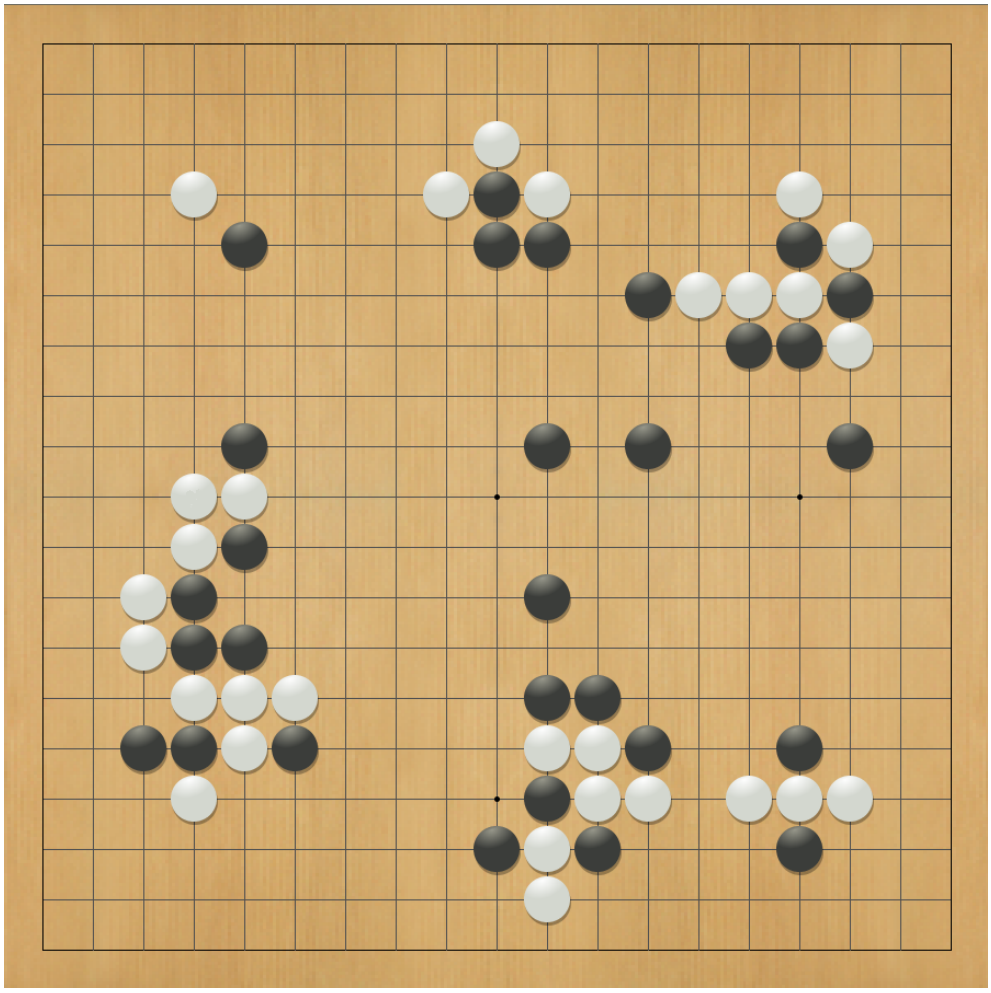

Figura 2.1: Tabuleiro do jogo de Go 19x19.

- *Handicap*: O termo *Handicap* no jogo de Go se refere ao número de peças colocadas no tabuleiro antes do início do jogo. O *Handicap* é tratado como uma vantagem dada ao jogador considerado mais fraco.
- *Ko*: O termo *Ko* se refere a uma sequência de capturas que se repetem em ciclos, como ilustra a Figura 2.4.
- *Komi*: No jogo de Go, o jogador com as peças pretas é quem inicia o jogo, então para compensar o jogador com peças brancas utiliza-se o *komi*, que é uma pontuação adicional dada ao jogador com peças brancas. O objetivo desta pontuação adicional é compensar a desvantagem do jogador que não iniciou o jogo.
- *Eye*: Um ponto vazio no tabuleiro e que esteja cercado de peças de um mesmo grupo.
- *Suicídio*: No jogo de Go, o termo *Suicídio* refere-se a situação em que um jogador realiza uma jogada que ocasiona a imediata captura de sua peça.
- *Território*: O território conquistado por um jogador consiste da quantidade de intersecções livres que estão cercadas por peças do jogador em questão.

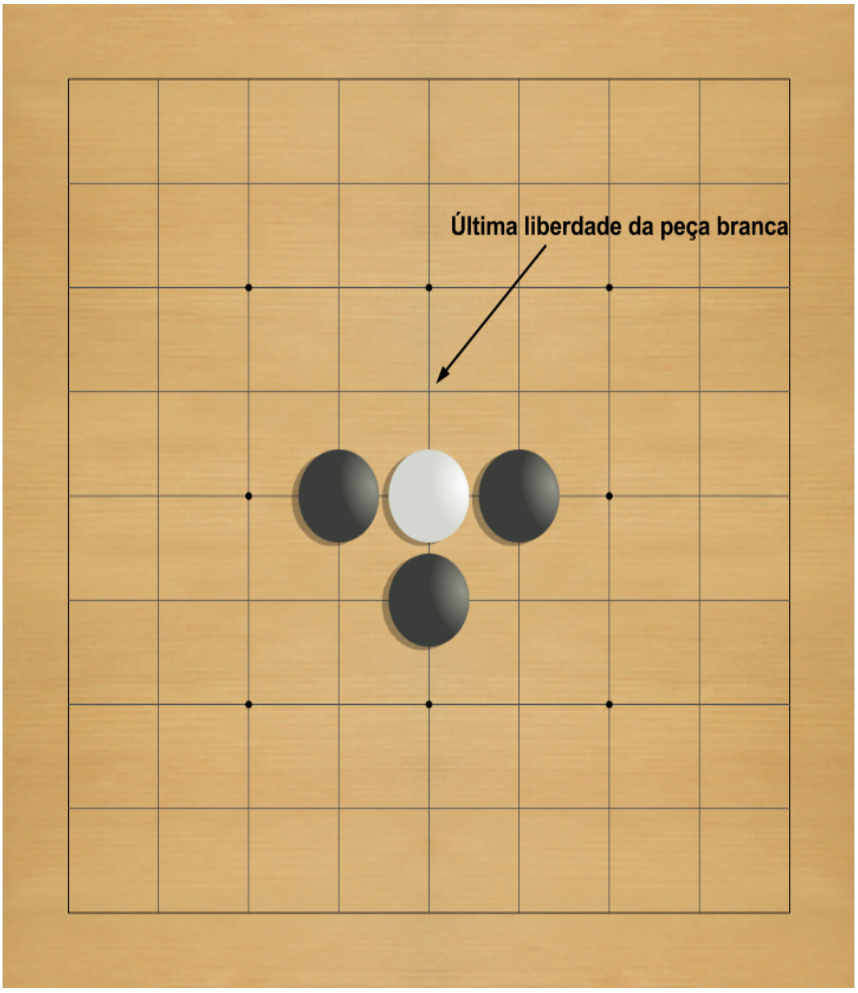

Figura 2.2: Exemplo de peça quase capturada.

O objetivo no jogo de Go é conquistar o maior território possível no tabuleiro e capturar as peças do adversário. O fim de jogo no Go ocorre quando ambos os jogadores decidem por não realizar mais jogadas, efetuando a ação de passar a sua vez de jogada. Uma vez encerrado o jogo, o cálculo da pontuação é iniciado, sendo que uma forma de efetuar o cálculo dos pontos consiste em somar os seguintes valores: o valor do território conquistado, a quantidade de peças capturadas e o *Komi* (no caso do jogador com peças brancas).

Pelo apresentado nesta seção, fica claro que o desafio do jogo de Go é justificado por uma série de características além da complexidade do jogo, como a simplicidade das regras e por particularidades, como o fato de que algum ganho vinculado a um dos jogadores também significa perda para o adversário, ou seja, não há formas de cooperação no jogo.

A próxima seção apresenta conceitos relativos ao algoritmo MCTS, bem como uma descrição do seu funcionamento.

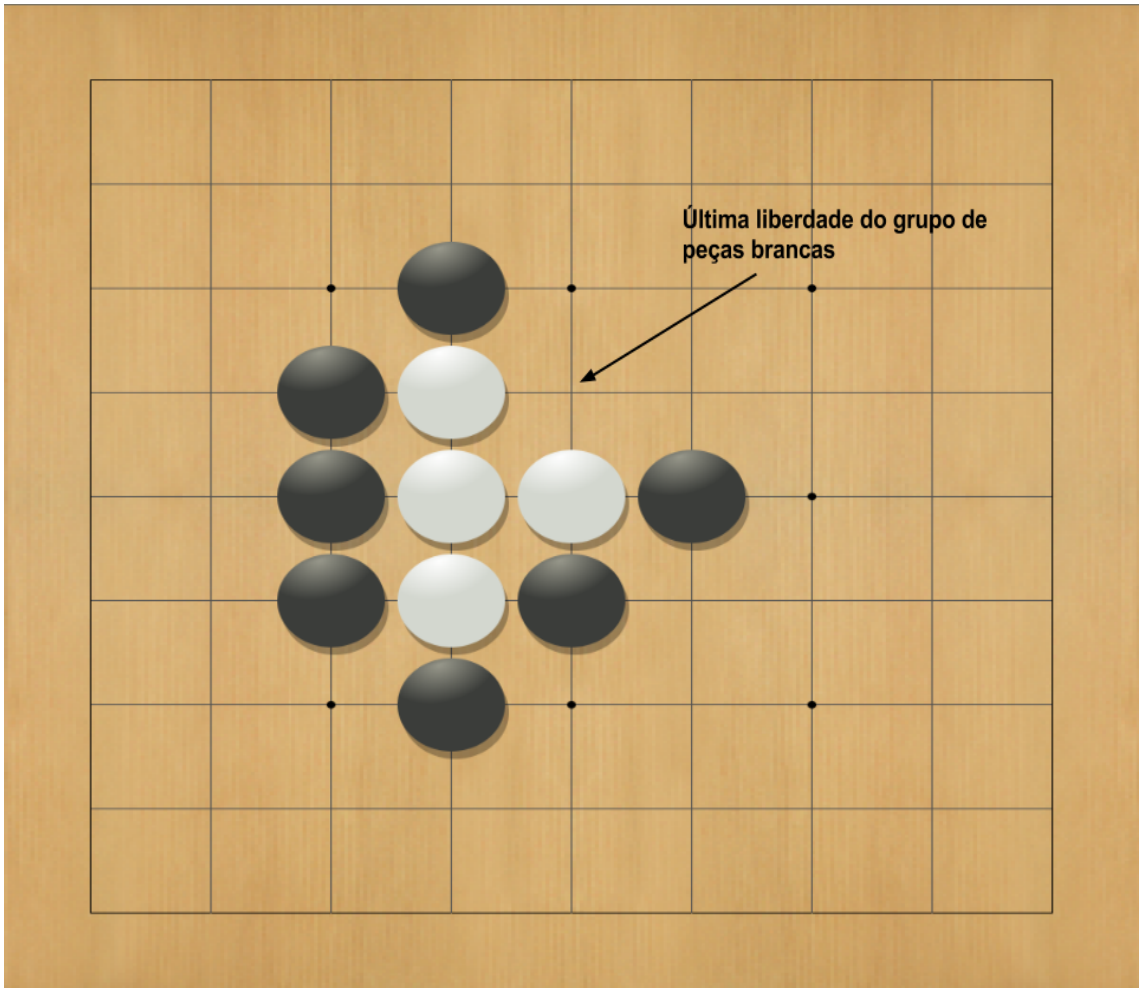

Figura 2.3: Exemplo de um grupo de peça quase capturado.

2.2 Busca em árvore Monte-Carlo

O algoritmo de busca MCTS tem contribuído para uma melhora considerável no jogo de Go [Gelly and Silver 2008]. A busca MCTS consiste em um algoritmo baseado em árvore, em que a escolha do melhor movimento ocorre através de uma sequência de simulações de jogos completos efetuadas a partir do estado corrente [Brügmann 1993]. Cada simulação de jogo corresponde a um caminho definido na árvore. Tal caminho é traçado através de políticas baseadas na escolha de nós (movimentos) com maior probabilidade de levar à vitória com mais frequência. Cada uma dessas simulações é denominada episódio. A quantidade de episódios de uma busca é definida em função de um tempo pré-estabelecido.

Um episódio de busca do algoritmo MCTS é realizado em quatro passos: seleção, expansão, *play-out* e retropropagação, sendo que a seleção e a expansão compõem a chamada etapa de *construção da árvore*. A seguir é descrito cada passo realizado em um episódio do algoritmo MCTS:

- *Seleção*: Caminho definido pelos melhores nós da árvore a partir da raiz. Tal defi-

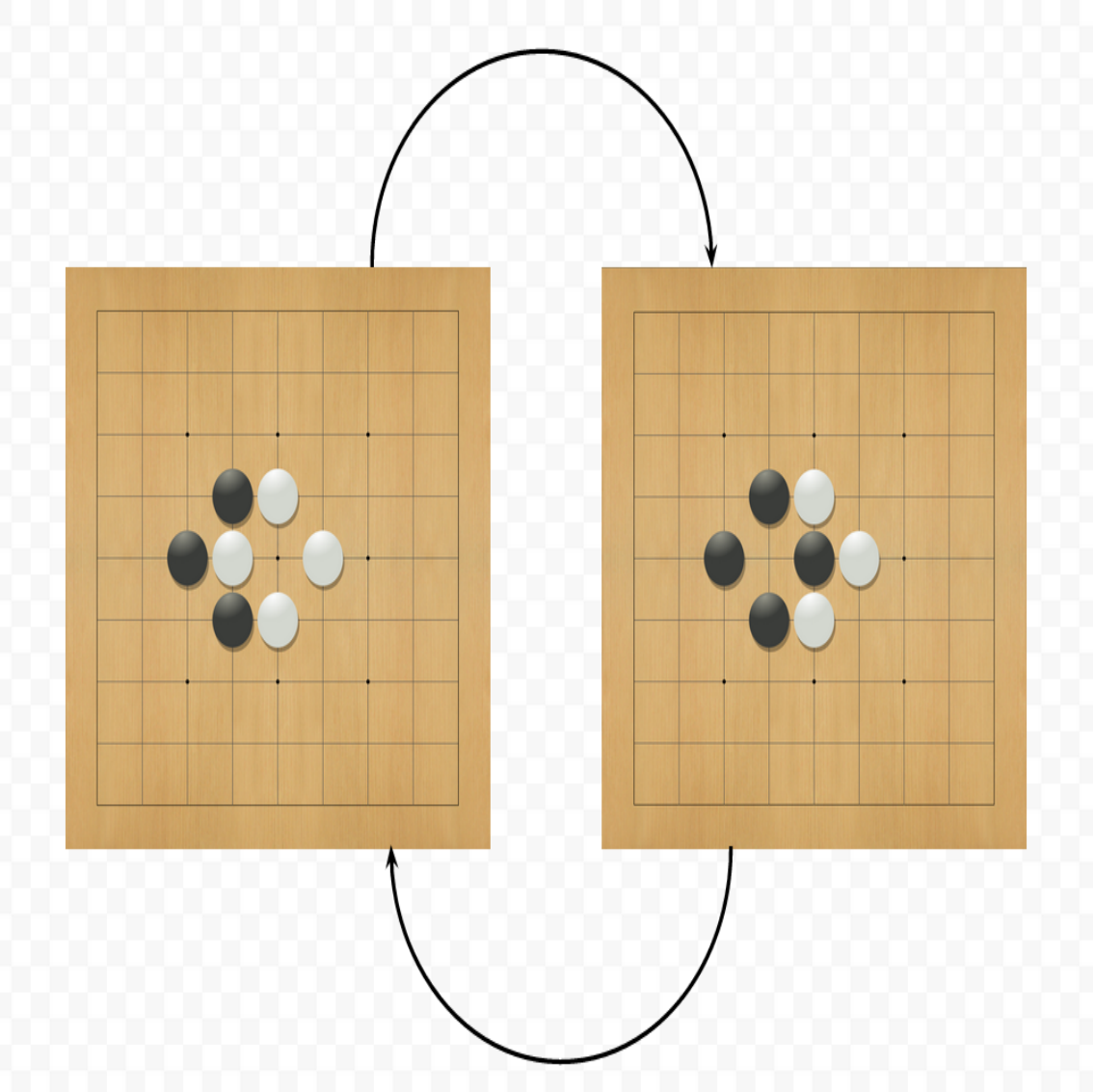

Figura 2.4: Exemplo de uma ocorrência de Ko.

nição é estabelecida por uma estratégia conhecida como *política de árvore*. A fase da seleção somente é concluída quando é encontrado um nó ainda não pertencente a árvore.

- *Expansão*: No final da *seleção*, é inserido na árvore o melhor filho da folha definida pela *seleção*.
- *Play-out*: Simulação sucessiva de movimentos a partir do nó inserido na expansão até que se atinja um estado de final de jogo. Tais movimentos são indicados por heurísticas chamadas *políticas de play-out*.
- *Retropropagação*: Uma vez concluída uma simulação de jogo (*play-out*), o resultado (vitória ou derrota) é retropropagado no caminho simulado da árvore e utilizado como parâmetro para atualizar os valores dos nós daquele caminho (o valor 0 indica derrota e 1 indica vitória).

A Figura 2.5 ilustra a construção de uma árvore MC.

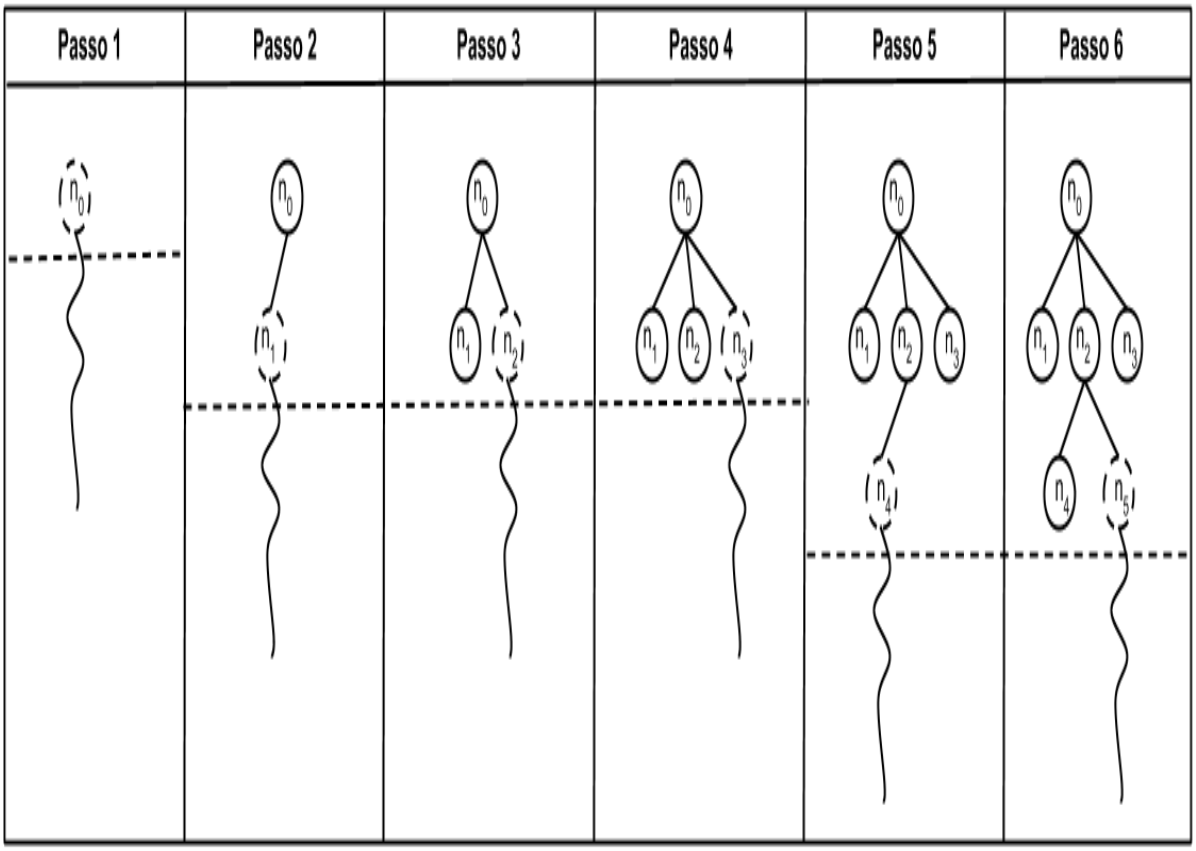

Figura 2.5: Exemplo de construção de uma árvore MC.

Pelo que foi apresentado na Figura 2.5, a formação de uma árvore de busca MC ocorre da seguinte maneira, sendo que os filhos do nodo n_0 são os seguintes, $F_{n_0} = \{n_1, n_2, n_3\}$:

- Insere: nó n_0
  - Árvore: $\{n_0\}$
  - 1^a simulação começa a partir do nodo n_0
- Insere: nó de F_{n_0} de melhor avaliação (suponha que seja o nodo n_1)
  - Árvore: $\{n_0, n_1\}$
  - 2^a simulação começa a partir do nodo n_1
- Insere: nó de F_{n_0} de melhor avaliação (suponha que seja o nodo n_2)
  - Árvore: $\{n_0, n_1, n_2\}$
  - 3^a simulação começa pelo n_2
- Insere: nó de F_{n_0} de melhor avaliação (suponha que seja o nodo n_3)

  - Árvore: $\{n_0, n_1, n_2, n_3\}$
  - 4^a simulação começa pelo n_3

- Insere: suponha que o n_2 é o nó com melhor avaliação, como este nó já faz parte da árvore, insere o n_4 que é filho do n_2

  - Árvore: $\{n_0, n_1, n_2, n_3, n_4\}$
  - 5^a simulação começa em n_4

- Insere: suponha que n_2 possui novamente melhor avaliação e insere o nó n_5 que é filho do n_2 e não está presente na árvore

  - Árvore: $\{n_0, n_1, n_2, n_3, n_4, n_5\}$
  - 6^a simulação começa em n_5

Dentre as políticas de árvore usadas com maior êxito em agentes para Go, destaca-se o UCT. A fim de implementar tal política, a estrutura dos nós da árvore MC precisa conter as seguintes informações [Gelly and Silver 2007]:

- $N(s)$: representa o número de jogos simulados na busca corrente;
- $Q(s, a)$: o valor estimado para o estado de tabuleiro s', que foi gerado pela execução da ação a a partir do estado s.
- $N(s, a)$: número de vezes que a ação a foi executada a partir do estado s, durante a busca corrente.

Paralelamente, podem-se citar como *políticas de play-out* de sucesso aquelas baseadas em regras (tal como as usadas no Fuego [Enzenberger et al. 2010]) e as baseadas no modelo Bradley-Terry (usadas no presente trabalho). As *políticas de árvore* são resumidas nas seções subsequentes.

Para ilustrar a atuação da políticas de árvore *play-out* durante toda a construção de uma árvore MC, apresenta-se na Figura 2.6 [Gelly and Silver 2011] as regiões de atuação de cada política na árvore MC.

2.2.1 Política UCT

O algoritmo UCT é um método de definição do caminho de simulação durante a etapa de *construção da árvore* MC [Childs et al. 2008], baseando-se, para tanto, nos nós que têm melhor avaliação corrente.

Tais avaliações procuram balancear a taxa de simulação dos nós com a qualidade dos mesmos, considerando como mais promissores os nós menos simulados e com maior qualidade. Desta maneira, a dinâmica de avaliação dos nós não se baseia apenas nos

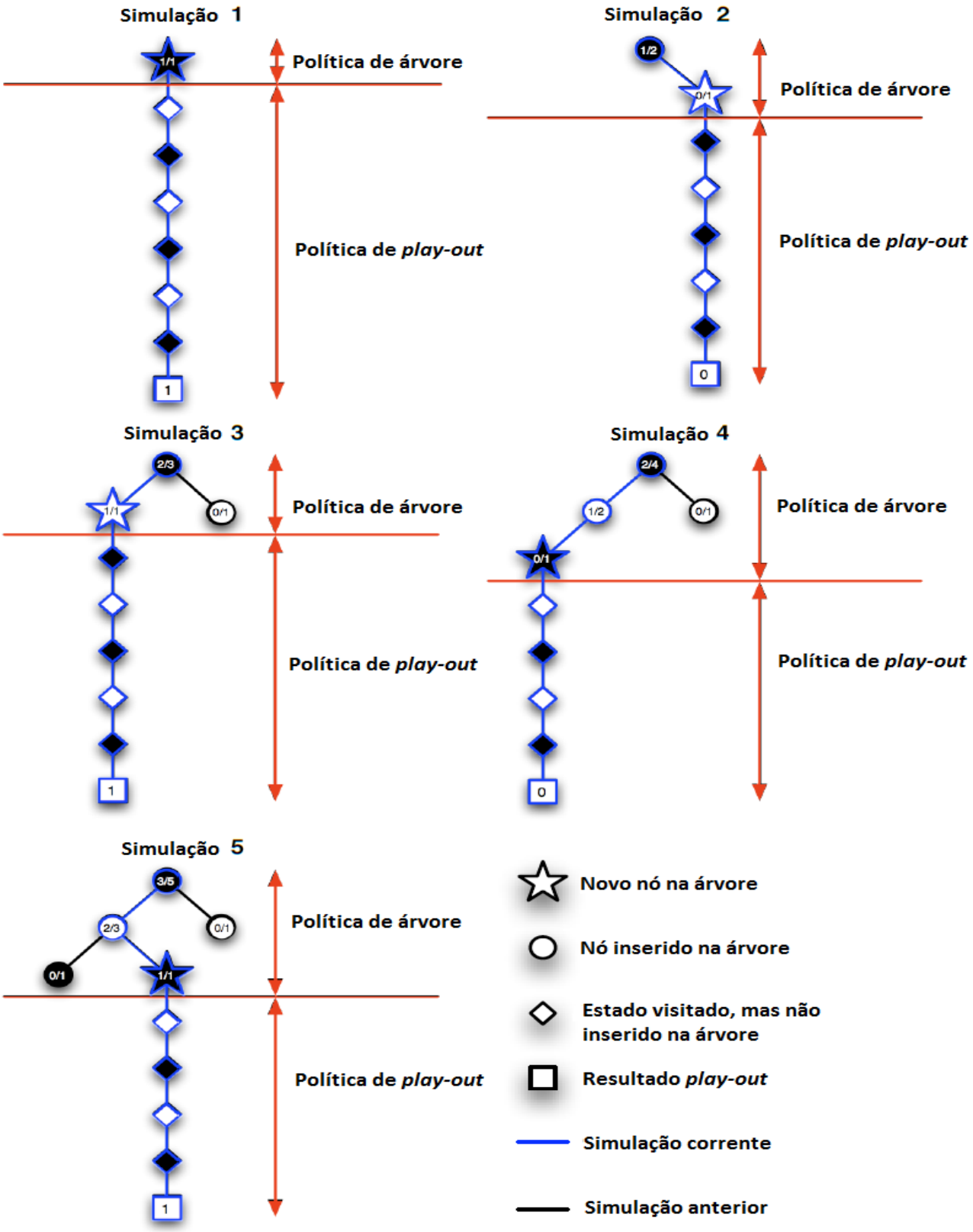

Figura 2.6: Exemplo da atuação das políticas de árvore e *play-out*.

movimentos avaliados como melhores até o momento, pois esta avaliação é dependente das simulações previamente realizadas, mas, também, na quantidade de vezes que os nós

foram simulados, favorecendo nós com menor quantidade de simulação. Resumindo, o valor de avaliação UCT refletirá, sempre, um balanceamento entre qualidade do nó e a frequência com que foi simulado, refletindo as políticas do "*exploration*"(prioriza a exploração de ramos da árvore que tenham sido pouco visitados) e do "*exploitation*" (prioriza a exploração de ramos da árvore que têm produzido bons resultados), respectivamente.

A política UCT através de simulações MC em Go é usada em conjunto com algoritmos responsáveis pelas atualizações de valores dos nós durante a *retropropagação*. Dentre esses algoritmos, destacam-se o MCTS e o RAVE resumidos a seguir.

2.2.1.1 Algoritmo MCTS

Durante a *retropropagação*, o algoritmo MCTS efetua a atualização dos valores de uma ação a executada a partir de um estado s de acordo com a equação 2.1.

$$Q_{(s,a)}^{updated} = Q_{(s,a)}^{corrente} + Estat.Vitoria_{(s,a)} + Estat.Simulacao_{(a)} \tag{2.1}$$

onde,

- $Q_{(s,a)}^{corrente}$: Valor da ação antes da atualização.
- $Q_{(s,a)}^{updated}$: Novo valor da ação após a atualização de valores da *retropropagação*.
- $Estat.Vitoria_{(s,a)}$: Proporção de vitórias obtidas em episódios, cujos caminhos a ação a tenha sido executada a partir da raiz na busca corrente.
- $Estat.Simulacao_{(a)}$: Número de vezes em que a ação a foi simulada na árvore de busca corrente a partir de qualquer estado.

A **parcela de reajuste** corresponde ao seguinte termo da equação 2.1: $Estat.Vitoria_{(s,a)}$ + $Estat.Simulacao_{(a)}$. Obviamente, o cálculo desse reajuste exige a atualização contínua dos valores associados aos nós, tal como mostrado a seguir:

$$N(s) = N(s) + 1 \tag{2.2}$$

$$N(s,a) = N(s,a) + 1 \tag{2.3}$$

As equações 2.2 e 2.3 atualizam, respectivamente, o número de simulações totais da busca corrente e a contagem do número de vezes que a ação a foi executada a partir do estado s.

2.2.1.2 RAVE

O algoritmo RAVE [Gelly and Silver 2011] é uma heurística responsável por definir uma relação entre as avaliações em diferentes posições da árvore, ou seja, ele compartilha

as avaliações de cada jogada pelas subárvores da árvore de busca. O RAVE realiza os cálculos das avaliações de maneira rápida, mas não muito precisa, porém sua avaliação se torna mais confiável quando aplicado junto ao algoritmo de busca MCTS [Gelly and Silver 2007]. O reajuste de valores no RAVE também é feito de acordo com a equação 2.1 de reajuste de valores. Contudo, o RAVE estende a abrangência do cálculo da parcela ($Estat.Vitoria_{(s,a)}$) da referida equação de modo que ele reflita a proporção do número de vezes (comparado com o número total de simulações na busca corrente) em que a ação a levou a uma vitória, independentemente do nó a partir do qual tenha sido simulada.

A Figura 2.7 ilustra a diferença no cálculo do termo $Estat.Vitoria_{(s,a)}$ feito pelo algoritmo MCTS e pelo RAVE.

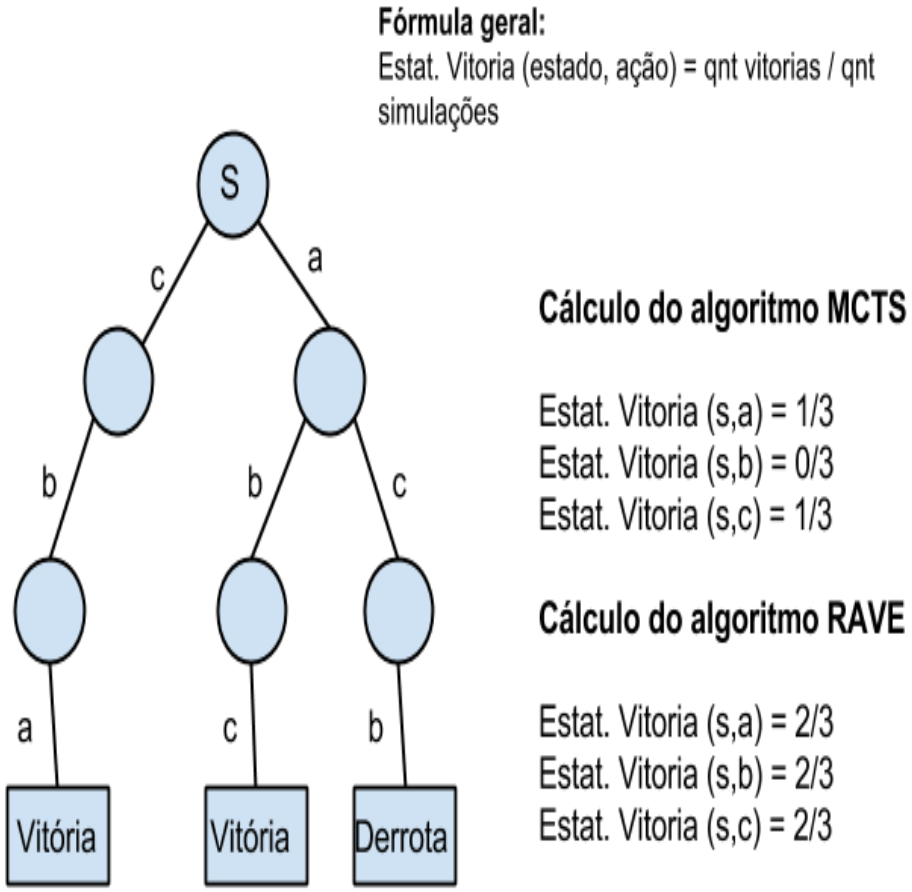

Figura 2.7: Comparação entre MCTS e RAVE.

2.3 Conceito de transposição em Go

Uma transposição é uma nova ocorrência de um estado anteriormente processado durante a execução do algoritmo de busca. Os casos de transposição no jogo de Go podem ocorrer devido a situações em que peças são capturadas e o estado do tabuleiro volta a

uma configuração anterior. Como no jogo de Go pode ocorrer a captura de uma única peça ou de um grupo de peças, o estado do tabuleiro pode retornar a configurações que diferenciam do estado atual por uma, duas ou muito mais peças, dependendo unicamente do tamanho do tabuleiro e do estágio do jogo.

A Figura 2.8 ilustra uma outra forma de ocorrência de transposição, em que é possível atingir uma mesma configuração de tabuleiro através de diferentes sequências de jogadas.

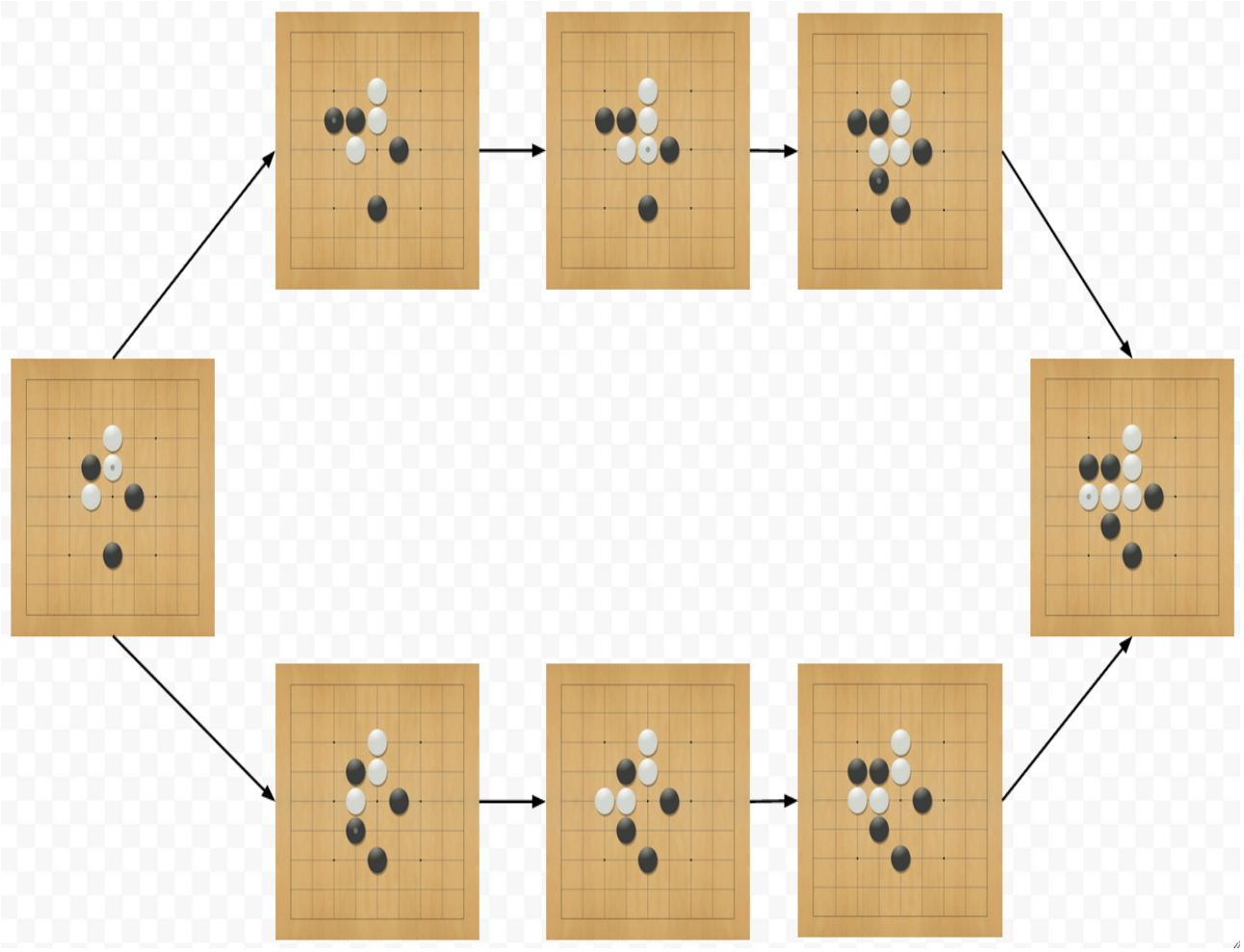

Figura 2.8: Exemplo de transposição por diferentes sequências de jogadas.

2.4 Modelo Bradley-Terry

O modelo BT permite realizar predições sobre resultados de competições. Seu princípio consiste em avaliar a força de cada indivíduo i, com um valor positivo denominado γ_i. As predições são feitas de acordo com a equação 2.4 que estima a probabilidade de um indivíduo vencer o outro [Hunter 2004].

$$P(i \text{ vencer } j) = \frac{\gamma_i}{\gamma_i + \gamma_j} \tag{2.4}$$

A aplicação do modelo BT pode ocorrer então para predizer a solução de um problema qualquer, sendo que para isto é necessário cumprir os seguintes passos:

- Formar um conjunto de soluções candidatas ao problema;
- Avaliar cada solução candidata;
- Realizar uma competição entre as soluções candidatas, gerando um valor que indica a probabilidade de cada solução candidata vencer as demais;
- Selecionar a solução candidata que obteve a maior probabilidade na competição como solução do problema.

No contexto do jogo de Go, o modelo BT pode ser aplicado para predizer a melhor jogada a ser realizada. Dessa maneira, é possível aplicar esta técnica para indicar os movimentos a serem executados na etapa de *play-out*, como é feito no BTT-Go.

2.4.1 Generalizações do modelo BT

O modelo BT pode ser generalizado para lidar com competições envolvendo mais de dois indivíduos. Por exemplo, para n jogadores [Hunter 2004], como mostra a equação 2.5:

$$\forall i \in \{1, \cdots, n\}, P(i \text{ vencer }) = \frac{\gamma_i}{\gamma_1 + \gamma_2 + \cdots + \gamma_n} \tag{2.5}$$

Outra generalização consiste em considerar não apenas indivíduos, mas também grupos. Neste caso, a força de um grupo é calculada pelo produto da força de seus membros. Por exemplo, considere os seguintes grupos (1-2-3), (4-2) e (1-5-6-7):

$$P(1-2-3 \text{ vencer } 4-2 \text{ e } 1-5-6-7) = \frac{\gamma_1\gamma_2\gamma_3}{\gamma_1\gamma_2\gamma_3 + \gamma_4\gamma_2 + \gamma_1\gamma_5\gamma_6\gamma_7} \tag{2.6}$$

Note na equação 2.6 que um mesmo γ não aparece mais de uma vez em um grupo, mas não impede que um γ apareça em mais de um grupo.

2.4.2 Inferência Bayesiana

Modelos BT fornecem uma distribuição probabilística sobre os resultados de futuras competições, uma vez conhecidas as forças dos competidores. Porém, na maioria das vezes, o valor exato da força dos competidores é desconhecido, precisando ser estimado por resultados de competições passadas [Bouzy and Chaslot 2005]. Esta estimativa pode ser feita com Inferência Bayesiana, onde γ é o vetor com a força dos competidores e *Res*

contém os resultados passados, como mostra a equação 2.7. Um conhecimento a priori é utilizado atribuindo uma vitória e uma derrota a cada γ.

$$P(\gamma|Res) = \frac{P(Res|\gamma)P(\gamma)}{P(Res)} \tag{2.7}$$

Da maneira como foi aplicado o modelo BT no BTT-Go não houve a necessidade de estimar a força de movimentos em função de resultados de competições passadas, visto que os dados usados na avaliação de cada movimento estão sempre disponíveis.

No próximo capítulo são apresentados alguns trabalhos relacionados ao que foi desenvolvido nesta dissertação, sendo que alguns deles utilizam técnicas, como o modelo BT, apresentadas no presente capítulo de fundamentação teórica.

Capítulo 3

Estado da Arte

Neste capítulo são apresentadas algumas pesquisas relacionadas com o trabalho apresentado nesta dissertação, destacando as diferenças entre os mesmos. Contudo, é importante destacar que existe entre os jogadores automáticos de Go aqueles destinados à competições e não somente à pesquisa e desenvolvimento de um jogador inteligente com aprendizagem não supervisionada. Os agentes apresentados nesta seção foram escolhidos por estarem ligados ao trabalho e por terem sido uma base de pesquisa para a realização do trabalho.

3.1 Migos

O programa jogador de Go Migos (Mini Go Solver) é dedicado a lidar com tabuleiros menores (a partir de 3x3). Quando aplicado a tabuleiros maiores (como 19x19), este programa lida com pequenos quadrados dentro do tabuleiro, como se fossem tabuleiros pequenos. Este programa utiliza o algoritmo de busca alfa-beta (diferentemente do BTT-Go, que utiliza busca MCTS + RAVE) e recursos de aprimoramento de busca, como a pesquisa por simetria e TT [van der Werf 2005].

Com a TT, o Migos faz uma análise de todos os sucessores de um nó para saber se contém uma transposição que leva a uma poda antes de retomar a busca em maior profundidade [Plaat et al. 1996]. Contudo, devido ao elevado custo computacional, somente a poda beta é aplicada até três ou mais níveis de distância do nó folha (a poda alfa é descartada) [van der Werf 2005].

O uso dessa melhoria na TT proporcionou ao jogador Migos resultados bons, uma vez que em experimentos realizados em tabuleiros 3x3, 4x4 e 5x5, a redução de nós com o uso da TT foi de 6%, 35% e 40% respectivamente, deixando evidente que com o aumento do tabuleiro o uso da TT é mais efetivo, especialmente se na TT estiver guardada a grande maioria dos casos de transposições como aconteceu nos experimentos mencionados.

O programa Migos com o uso da TT armazena as seguintes informações:

- Avaliação da posição;
- O melhor movimento;
- A profundidade da sub-árvore já explorada.

O efeito do uso da TT é ampliado com uma busca que envolve a análise de tabuleiros contendo posições simétricas que já foram pesquisadas. Isso ocorre devido ao fato de que tabuleiros correntes que apresentam simetria total ou parcial com relação a outros previamente simulados e presentes na TT não precisam ser recalculados, desde que tenham profundidade no máximo igual ao dos valores armazenados na TT. Neste caso, serão usados tais valores. A TT utilizada no BTT-Go apenas utiliza simetria total, sendo que a cada registro guardado na TT outros três também são inseridos, correspondendo a três rotações do tabuleiro de 90° cada.

Entenda-se por simetria no jogo de Go como sendo uma configuração de tabuleiro que é repetida mantendo as suas características, inclusive na avaliação. A simetria pode ser total ou parcial. Dentre as simetrias totais que podem ocorrer no tabuleiro estão a reflexão, a rotação e a translação, além da inversão da cor de todas as peças presentes no tabuleiro, pois a avaliação neste caso é a mesma, só diferindo o jogador. As simetrias parciais podem ocorrer de maneira semelhante às totais, mas sem envolver todas as peças do tabuleiro, como mostra a Figura 3.1.

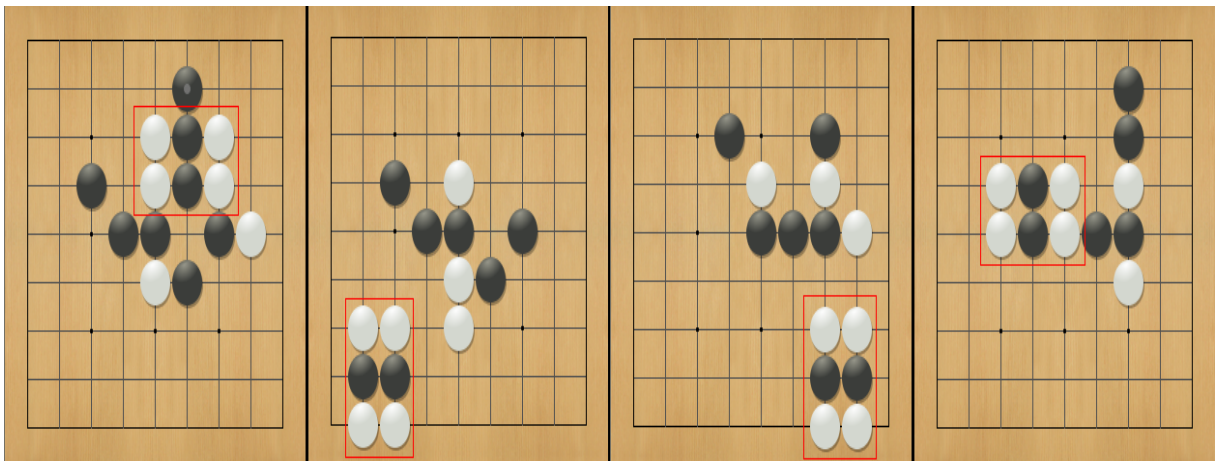

Figura 3.1: Exemplo de simetria parcial.

Na ocorrência de múltiplas simetrias em uma mesma configuração de tabuleiro, todas elas são usadas para definir limites estreitos na avaliação, ou seja, o valor guardado na TT não é referente a uma única simetria já identificada, mas sim um novo valor que visa avaliar, juntamente, as diferentes simetrias presentes no tabuleiro.

Dessa forma, a análise de simetria combinada com a TT, reduz ainda mais o tempo de processamento.

3.2 MOGO

MOGO foi o primeiro programa jogador de Go a ganhar oficialmente de um jogador humano em um tabuleiro 19x19 [Gelly et al. 2006].

O MOGO possui a característica de utilizar poucas informações sobre o jogo de Go, nada além das regras básicas do jogo. Dessa maneira, o MOGO consegue desenvolver métodos um pouco mais gerais que não são específicos para o jogo de Go, permitindo compreender o poder, e as limitações da técnica de busca em árvore MC. Para os tabuleiro 9x9 o MOGO utiliza um grande *opening book* que lhe proporciona importantes ganhos, especialmente, em tempo de processamento, pois os movimentos já estão disponíveis nessa base de dados, permitindo que eles sejam realizados imediatamente [Rimmel et al. 2010].

Dentre as principais contribuições do jogador MOGO estão, o uso de reconhecimento de padrões durante as simulações do algoritmo MCTS, o uso de uma estrutura de árvore dinâmica e paralelização do programa, o que não é feito pelo BTT-Go.

Basicamente, usa-se reconhecimento de padrões para criar sequências significativas em simulações, encontrando respostas locais, pois os movimentos realizados não são necessariamente as melhores jogadas em uma visão global [Gelly et al. 2006]. Os padrões usados na política de *play-out* do Fuego são semelhantes às utilizadas no MOGO, sendo esta uma das mudanças feitas pelo BTT-Go através do uso do modelo BT.

Os experimentos realizados com o MOGO mostraram que a precisão sobre a próxima jogada diminui, se o mesmo padrão for usado para encontrar movimentos interessantes em qualquer lugar no tabuleiro, ao invés de um lugar próximo ao movimento anterior [Gelly et al. 2006].

3.3 Crazy Stone

O Crazy Stone é um programa jogador do jogo de Go desenvolvido por Rémi Coulom que faz uso da busca em árvore MC, juntamente com uma técnica Bayesiana de aprendizagem baseada no modelo BT [Coulom 2007a]. Nesta técnica, cada movimento é representado por um conjunto fixo de *features*, que são heurísticas que descrevem uma propriedade do movimento candidato que definirá a próxima jogada.

Diferentemente do BTT-Go e do Fuego, o Crazy Stone não utiliza *prior knowledge* na fase de *construção da árvore* do algoritmo de busca MCTS, mas sim em conjunto com a técnica Bayesiana que define o caminho da seleção da árvore baseada nas avaliações BT [Coulom 2007a], sendo que inicialmente são atribuídas uma vitória e uma derrota aos movimentos candidatos a seleção.

O modelo BT no Crazy Stone foi aplicado de duas maneiras: primeiro, o modelo BT foi utilizado como método de seleção na construção da árvore do algoritmo MC, e segundo, o modelo BT foi aplicado como padrões que serviam para que o processo de

busca somente considera-se movimentos que atendam algum dos padrões. O agente BTT-Go, diferentemente do Crazy Stone, utiliza o método BT, exclusivamente, para seleção dos movimentos durantes os *play-outs*, além de aplicar uma diferente forma de avaliação.

O módulo avaliador BT opera conjuntamente com uma TT [Enzenberger and Müller 2010] que serve de repositório para nós já avaliados anteriormente, enquanto que o BTT-Go utiliza TT para guardar as avaliações do algoritmo MC. Os estados avaliados pelo módulo BT são representados por *features*, que são funções matemáticas que expressam informações relevantes sobre os tabuleiros [Coulom 2007a].

O Crazy Stone utiliza as *features* para avaliar um movimento e gerar o grau de força do mesmo, sendo que este grau de força serve para indicar o quanto um movimento é bom. Uma diferença da aplicação do modelo BT no Crazy Stone em relação ao BTT-Go, está na forma de cálculo deste grau de fora e da avaliação feita pelas *features*, que no BTT-Go são chamados de atributos, sendo que entre os atributos há um criado exclusivamente para o BTT-Go.

No Crazy Stone é utilizada uma tabela que serve para associar valores aos resultados obtidos pelas avaliações das *features*, então o cálculo do grau de força do movimento é feito pelo produto destes valores oriundos da tabela. No BTT-Go o cálculo do grau de força de um movimento ocorre de um modo diferente, em que a avaliação do movimento, feita por atributos, gera um valor usado, diretamente, no cálculo do grau de força do movimento, ou seja, não é preciso consultar uma tabela.

Em [Coulom 2007a] são apresentados resultados de experimentos em que avalia-se o agente através de comparações entre o melhor movimento indicado pelo método Bayesiano e o movimento que, na mesma situação, seria indicado por mestres humanos (disponível em banco de dados). Nestes testes, o Crazy Stone, comparado a trabalhos relacionados, mostrou ter boa predição de movimento a partir do estágio de meio de jogo. Em outros testes, o Crazy Stone jogou contra o agente GNU Go 3.6 [Bump 2003], obtendo uma taxa de vitória de 90.6% e 57.1%, respectivamente, nos tabuleiros com dimensões 9x9 e 19x19.

3.4 Érica

O jogador automático de Go Érica [Huang et al. 2011] possui entre seus desenvolvedores o responsável pelo desenvolvimento do jogador Crazy Stone, sendo que este jogador pode ser visto como a versão anterior ao Érica. Sendo assim, o Érica herdou algumas técnicas do Crazy Stone como a busca MCTS e a forma de processar padrões de tabuleiros para geração de movimentos. Em comparação ao BTT-Go, a forma de geração dos movimentos pelo modelo BT também é diferente em relação a este jogador, visto que o BTT-Go criou sua própria forma de avaliação do movimentos.

O jogador Érica, diferentemente de sua versão anterior, utiliza uma técnica de balanceamento de simulações para ajustar os parâmetros da *política de play-out* na busca

MCTS, atribuindo valores de importância aos padrões de tabuleiro.

3.5 Gnu Go

O jogador Gnu Go [Bump 2003] utiliza a busca MCTS da mesma maneira como é feita pelo MOGO, ou seja, o jogador utiliza aprendizagem supervisionada através de padrões de tabuleiros utilizados durantes as simulações MC.

O jogador Gnu Go foi utilizado neste trabalho em jogos realizados contra o BTT-Go. O Gnu Go já foi também utilizado em experimentos com outros agentes como o Crazy Stone. Sendo assim, devido ao registro de competições, o Gnu Go assume um importante papel para realizar a comparação do BTT-Go com outros agentes.

3.6 Fuego

A plataforma Fuego foi construída a partir de dois projetos anteriores, o Smart Game Board [Kierulf 1990] e o Explorer [Müller 1995], em que o primeiro consiste em uma coleção de ferramentas para desenvolvimento de jogadores automáticos e o segundo um jogador de Go desenvolvido com os recursos oferecidos pelo Smart Game Board.

Motivado pelo sucesso alcançado pelos programas jogadores de Go, Crazy Stone e MOGO, que utilizam o método MC para busca em árvore, em 2007, Enzenberger começou a desenvolver um programa que implementava tal técnica. Inicialmente chamado apenas de UCT (Upper Confidence Tree), este programa desenvolvido por Enzenberger foi renomeado para Fuego, e se tornou um projeto com código-aberto em 2008.

A plataforma Fuego conta também com um jogador de Go de mesmo nome, com aprendizagem supervisionada. Este jogador usa a expansão de árvore MC, processada através do algoritmo de busca UCT (Upper Confidence Tree) combinado com RAVE. Nas jogadas iniciais, o agente Fuego utiliza um *opening book* e nas etapas iniciais da *construção da árvore*, o Fuego usa uma avaliação baseada em *priori knowledge*, ou seja, os valores iniciais do $Q_{(s,a)}$ (avaliação do nó) são definidos pelo *prior knowledge*. Nas jogadas posteriores, os nós da árvore terão suas avaliações alteradas para uma combinação de resultados UCT + RAVE + simulação. Salientando que o Fuego não utiliza nenhum tipo de repositório de dados já processados, como faz o BTT-Go.

No jogador Fuego a política UCT orienta a *construção da árvore*, balanceando a exploração dos nós pouco simulados com os nós que possuem melhor avaliação ($Q_{(s,a)}$). Contudo, as primeiras expansões do algoritmo MCTS são orientadas pelo *prior knowledge*, que avaliam os nós gerados na expansão, mas sem dispor de simulações executadas a partir destes nós. Diferentemente no BTT-Go, com a TT é possível utilizar uma avaliação mais precisa do que o *prior knowledge*, desde que exista um registro da configuração do tabuleiro na TT.

A etapa de *play-outs* no jogador Fuego é orientada por uma política definida por regras que estabelecem o movimento a ser simulado em função do tipo do tabuleiro corrente definido por políticas (*Nakade Heuristic*, *Atari Capture*, *Atari Defense*, *Low Liberty*, *Pattern* e *Capture Move*). Ressalta-se que o tabuleiro corrente pode encaixar-se em mais de um dentre estes padrões. Neste caso, a política de *play-out* define o movimento a ser simulado a partir de uma hierarquização entre esses padrões, isto é, ser escolhido o movimento sugerido pelo padrão de maior hierarquia. Sempre que o tabuleiro corrente não se encaixa em um dos padrões de tipo de tabuleiro, é estabelecido um movimento aleatório. Estas políticas geram movimentos que são executados dentro da etapa de *play-outs*, diferentemente do BTT-Go que aplica nesta etapa o método BT para realizar as jogadas.

O processo de busca do algoritmo MCTS, aplicado no jogador Fuego, pode ser resumido da seguinte maneira: repetem-se N vezes uma sequência de episódios, onde cada um deles é composto por duas fases: *construção de árvore* e simulação MC. Durante a *construção da árvore* MC, ocorre o processo de *expansão* de busca, isto é, a inserção de um novo nó. O novo nó é escolhido imediatamente após o processo chamado de *seleção* que consiste em se traçar um caminho a partir do nó raiz definido pelos nós da árvore que têm melhor avaliação. Assim sendo, o processo de seleção de cada episódio é sucedido pela inserção do novo nó na árvore, caso exista. Convém salientar que sempre que um novo nó é inserido no mesmo ramo em que tinha sido efetuada a inserção anterior, o processo de expansão é denominado *exploitation*. Caso o novo nó seja inserido em outro ramo alternativo (ou seja, a seleção corrente refere-se a um caminho distinto daquele da anterior), o processo de expansão é denominado *exploration*. A partir daí, começa os *play-outs*, em que se define um caminho que começa pelo último nó inserido na árvore e que chega a um estado de final de jogo, retornando um valor que indicará derrota ou vitória. Salienta-se que os nós do caminho percorrido durante o *play-out* não são inseridos na árvore.

3.7 Tabela comparativa dos jogadores

Para uma melhor comparação entre os agentes apresentados neste capítulo e o agente desenvolvido neste trabalho (BTT-Go), é apresentada na tabela 3.1 uma relação entre os jogadores e as técnicas utilizadas.

Salienta-se que mesmo utilizando uma mesma técnica existem, em alguns casos, diferenças entre suas aplicações, além de funcionarem associadas com técnicas diferentes. Os detalhes de aplicação das técnicas e as diferenças foram apresentadas nas seções anteriores.

Os campos na tabela assinalados com "**X**" indicam que o jogador utiliza a referida técnica, enquanto que os campos assinalados com "–" indicam a ausência da referida técnica.

Uma vez que neste capítulo uma série de trabalhos relacionados com o que foi desenvolvido nesta dissertação foram apresentados, enfatizando as semelhanças e diferenças

Tabela 3.1: Tabela comparativa entre jogadores

Jogadores	**MCTS**	**TT**	**Modelo BT**
Migos	X	X	–
MOGO	X	–	–
Crazy Stone	X	X	X
Érica	X	–	X
Gnu Go	X	–	–
Fuego	X	–	–
BTT-Go	X	X	X

entre os trabalhos, então é propício a apresentação dos detalhes da criação do agente BTT-Go, sendo que isto acontece no capítulo a seguir.

Capítulo 4

BTT-Go

O agente BTT-Go foi criado a partir do jogador Fuego [Enzenberger et al. 2010] e tem o objetivo de reduzir a supervisão inerente a tal agente, comprometendo o mínimo possível, seu desempenho. Para atingir tal meta, o BTT-Go insere na busca MCTS + RAVE do Fuego duas ferramentas: TT e uma técnica inspirada no modelo BT. Para fins comparativos, foram implementadas três versões, cada uma apresenta uma alternativa distinta de utilização de tais recursos, conforme apresentado nas seções 4.1, 4.2 e 4.3.

A motivação da criação deste agente está relacionado com o desafio da criação de um jogador de Go competitivo e sem utilizar muita supervisão, além da ampla possibilidade de estudo e aplicação de técnicas que contribuíram na criação de um jogador inteligente.

4.1 Versão 1 do BTT-Go: Redução das simulações através da TT

Sendo a TT uma estrutura que armazena dados previamente processados, o seu histórico de avaliações pode ser utilizado como recurso de redução da supervisão na aprendizagem. Para alcançar a referida redução de supervisão, foram adotadas as seguintes medidas:

- Uso da informação disponível na TT para substituir o uso da heurística *prior knowledge* na *expansão* que ocorre na fase de *construção da árvore*.
- Interrupção da etapa de *play-outs* sempre que ocorrer uma transposição;
- Uso da informação armazenada na TT para atualizar os valores dos nós que estão presentes no caminho percorrido pelo algoritmo de busca durante a *retropropagação*.

A fim de impedir que os efeitos da redução de supervisão introduzidos pelo uso dos dados da TT comprometam a qualidade da busca, foi adotada a estratégia em que a TT somente começa a ser preenchida quando 50% das posições do tabuleiro já estiverem

ocupadas (tanto por peças do BTT-Go quanto por peças do adversário). Com essa estratégia, os dados armazenados na TT terão maior acuidade, pois terão sido gerados depois de concluída uma quantidade maior de simulações de busca, gerando assim dados mais precisos.

A maneira como foi implementada a TT e o modo de funcionamento da mesma são descritos na seção a seguir.

4.1.1 Arquitetura geral da 1^a versão do BTT-Go

Como descrito anteriormente, o agente BTT-Go visa reduzir a supervisão existente no algoritmo de busca do Fuego. Sendo assim, o BTT-Go utiliza informações vinculadas ao histórico de avaliações dos nós disponíveis na TT.

A Figura 4.1 ilustra a arquitetura do agente BTT-Go utilizando a TT. Para escolher um movimento, o tabuleiro corrente é apresentado ao algoritmo de busca, #1 (seta 1 da figura). Então, este algoritmo constrói a árvore MC executando recursivamente n episódios do algoritmo de busca em um intervalo de tempo pré-estabelecido. Para cada nó s envolvido nestes episódios, o algoritmo apresenta tal estado (tabuleiro) ao módulo gerador de chave *hash*, #2. Posteriormente, a chave *hash* correspondente ao nó s é calculada, verificando-se a existência de algum registro, associado a esta chave, na TT, #3. Caso não exista este registro, significa que s ainda não foi visitado e também não pertence à árvore de busca, ou seja, s é um candidato a etapa de expansão, #4. Neste caso, o algoritmo de busca segue sua execução normalmente, considerando a avaliação indicada pelo *prior knowledge* como sendo a avaliação de s (tal como no Fuego). Assim que a fase da *retropropagação* é concluída, as avaliações dos nós (pertencentes ao episódio corrente de busca e já presentes na TT) são atualizadas. Caso contrário, os nós que ainda não pertencem à TT, terão um registro inserido na mesma, seguindo todo o processo de geração de chave *hash* já mencionado. Na seta #5 é representada a situação em que s pertence à TT. Neste caso, o valor disponível na TT é utilizado na etapa de *construção da árvore*. Se esta situação ocorrer com um nó s durante a fase de *play-outs*, então a simulação do episódio é interrompida e utiliza-se o valor disponível na TT, sendo esta uma das estratégias aplicadas para reduzir supervisão. Finalmente, assim que os n episódios são concluídos, o algoritmo de busca indica qual o melhor movimento m a ser executado, #6. Então, o tabuleiro é alterado, devido a execução da jogada m, assumindo uma nova configuração, #7.

A próxima seção descreve detalhadamente cada módulo da arquitetura apresentada pela Figura 4.1.

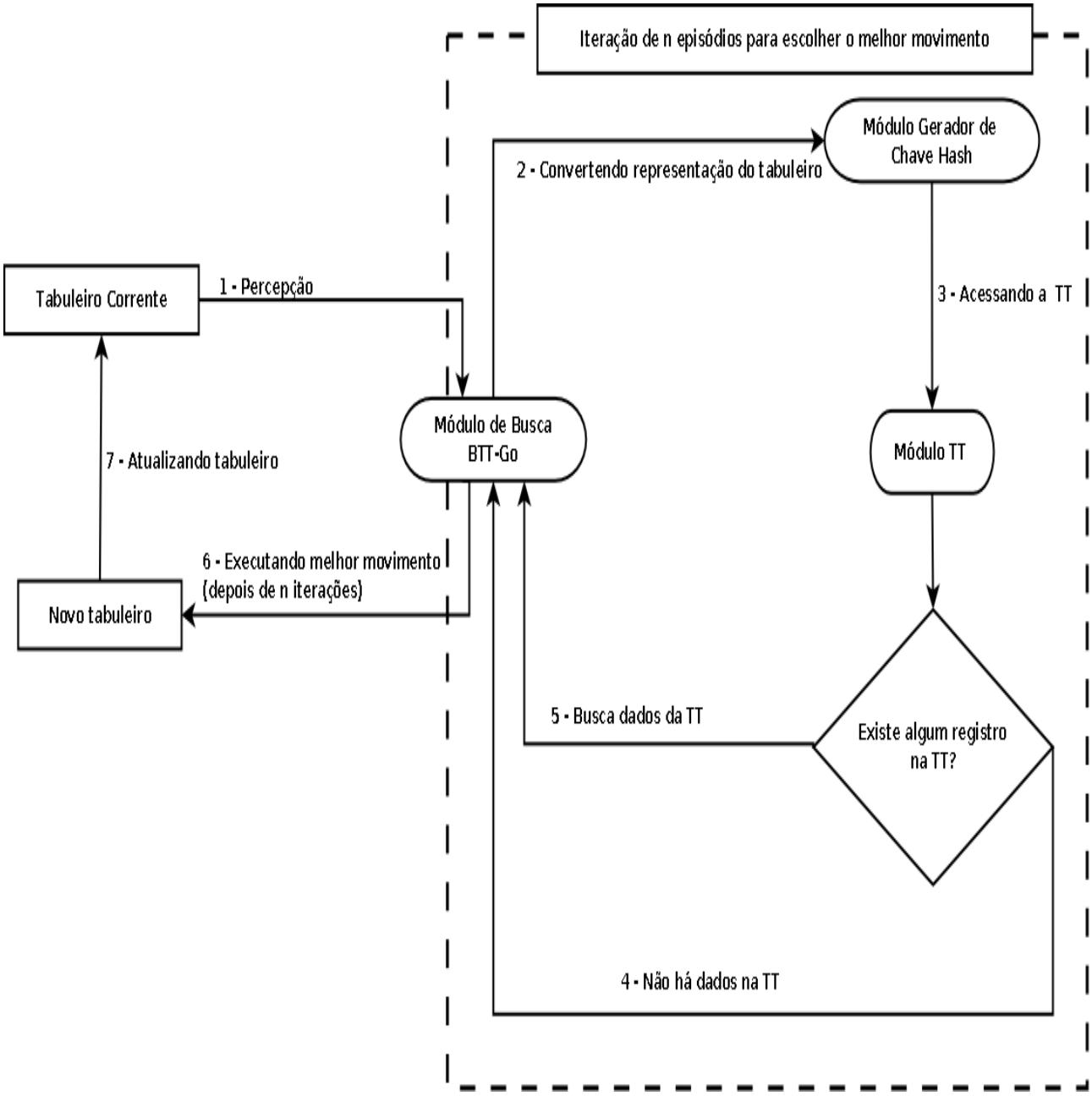

Figura 4.1: Arquitetura da TT no BTT-Go.

4.1.2 Módulos da arquitetura de uso da TT

A seção 4.1.1 apresentou uma arquitetura geral do uso da TT no BTT-Go, descrevendo a interação entre os módulos da arquitetura. Nesta seção há mais detalhes sobre o funcionamento de cada módulo e sobre como cada um foi implementado.

4.1.2.1 Módulo Gerador da Chave *Hash*

No BTT-Go o endereço de cada estado s na TT é indicado por duas chaves *hash*, uma de 64 bits e outra de 32 bits (conforme seção 4.1.2.2). Ambas as chaves são criadas pelo módulo gerador de chave *hash*, como é apresentado pelo pseudo-código abaixo:

GeracaoChaveHash(tabuleiro)

```
1   VetorPosicaoID = IdentificaPosicoesOcupadas(tabuleiro);
2   ChaveHash = VetorPosicaoID[1];
3   Para i = 2 até Tamanho do VetorPosicaoID faça
4       id = VetorPosicaoID[i];
5       ChaveHash = XOR(ChaveHash, id);
6   Fim Para;
7   retorna ChaveHash;
```

O algoritmo apresentado no pseudo-código foi inspirado na técnica de Zobrist [Zobrist 1970] que se fundamenta nas seguintes propriedades do operador XOR (*ou exclusivo*): a associatividade (o resultado da operação XOR não muda se for alterada a ordem das operações entre os bits) e a comutatividade (a ordem dos operandos não altera o resultado da operação). Assim, o método descrito pelo pseudo-código recebe a configuração do tabuleiro como parâmetro, o que serve para identificar as posições já ocupadas por peças no tabuleiro. Essas posições recebem identificadores únicos (números inteiros) e são guardados em um vetor (*VetorPosicaoID*), linha 1. Posteriormente, para cada identificador guardado no *VetorPosicaoID* (linha 3) é aplicado o operador XOR, sendo que depois da primeira aplicação, o resultado é guardado na variável *ChaveHash*. Assim, esse resultado será usado, posteriormente, em uma nova aplicação do operador XOR com outro membro do *VetorPosicaoID*, como pode ser observado na linha 5. Depois de aplicado o operador XOR em todos os identificadores do *VetorPosicaoID*, obtém-se, enfim, a chave *hash* utilizada como endereço na TT (linha 7).

Salienta-se que o operador **XOR** (ou exclusivo), representado por $\otimes$, resulta em *verdadeiro* se, e somente se, apenas um dos operandos também for *verdadeiro*.

O operador XOR pode ser aplicado sobre dois operandos numéricos, como descrito a seguir [Zobrist 1970]:

- Operandos na base binária: o XOR aplicado sobre dois bits quaisquer resulta em *1* se, e somente se, um dos operandos tiver o valor *1*. Assim, considere a sequência binária de n bits, $seq_1 = b_1, b_2, \cdots, b_n$, e a sequência binária, também de n bits, $seq_2 = r_1, r_2, \cdots, r_n$. Para encontrar uma terceira sequência, resultante da aplicação do operador XOR em seq_1 e seq_2, basta aplicar XOR nos bits das posições correspondentes, $seq_3 = b_1 \otimes r_1, b_2 \otimes r_2, ..., b_n \otimes r_n$;
- Operandos na base decimal: para aplicar o operador XOR em inteiros na base decimal, basta primeiramente convertê-los para a base binária, e realizar o procedimento já descrito sobre a aplicação do operador XOR sobre operandos na base binária.

Serão consideradas as seguintes propriedades para o operador XOR que será aplicado sobre as sequências aleatórias com n bits:

- $r_i \otimes (r_j \otimes r_k) = (r_i \otimes r_j) \otimes r_k$;
- $r_i \otimes r_j = r_j \otimes r_i$;
- $r_i \otimes r_i = 0$;
- Se $S_i = r_1 \otimes r_2 \otimes \cdots \otimes r_n$, então S_i é uma sequência aleatória de n bits;
- S_i é uniformemente distribuída, sendo que uma variável é dita uniformemente distribuída quando assume qualquer um dos valores possíveis com a mesma probabilidade.

Para geração de uma chave *hash*, primeiramente, atribui-se um número pseudo-aleatório para cada possibilidade de ocupação de uma posição do tabuleiro. Por exemplo, uma peça branca na posição [0,0] possui um valor, ao passo que uma peça preta, na mesma posição possui outro valor. A Figura 4.1 ilustra a atribuição de identificadores únicos, condicionados a cor da peça que ocupa a posição.

Tabela 4.1: Tabela de identificadores associados às cores das peças (Tabuleiro 9x9).

Identificador	**Cor da Peça**	N^o **intersecção**
1534826452842130000	Branca	1
2156482356482410000	Preta	
1423652215688410000	Branca	2
1241156845235680000	Preta	
2258648264123590000	Branca	3
1478956234580150000	Preta	
1365481236049780000	Branca	4
2564120519841230000	Preta	
2145784692350450000	Branca	5
2158496541236580000	Preta	
1548236498745210658	Branca	6
1250487963201478441	Preta	
2256487015479568412	Branca	7
1478402541369852014	Preta	
...	...	...
1658741235018946250	Branca	79
2345785065847126985	Preta	
3584125478526589452	Branca	80
2569840235741014587	Preta	
4152894235655012488	Branca	81
3158496512048703659	Preta	

A seguir descreve-se um exemplo de obtenção de chave *hash* para o tabuleiro ilustrado na Figura 4.2 de acordo com a técnica de Zobrist.

Pelo que foi apresentado na Figura 4.2, é possível criar uma chave *hash* pelos identificadores apresentados na figura da seguinte maneira: 17903615704209920410 $\otimes$ 145072576720

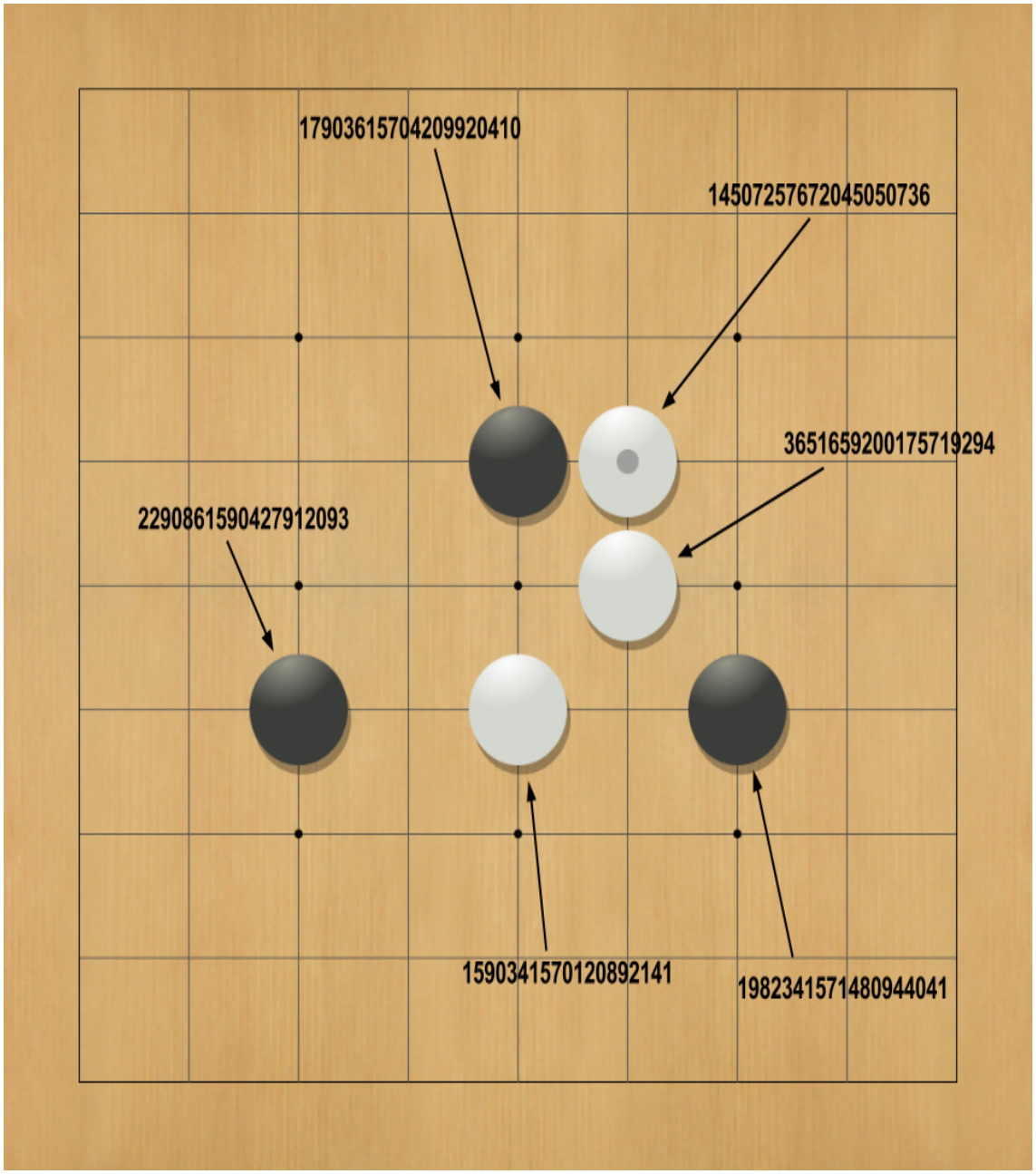

Figura 4.2: Peças no tabuleiro e seus identificadores.

45050736 ⊗ 3651659200175719294 ⊗ 1982341571480944041 ⊗ 1590341570120892141 ⊗ 2290861590427912093 = 1285063583604174157.

4.1.2.2 Módulo TT

No BTT-Go o módulo TT utiliza a classe *TableEntry* que serve para definir os dados que compõem um registro na TT, ou seja, toda entrada de dados na TT deve conter essas informações, como mostrado a seguir:

```
class TableEntry {
        long long int chaveHash1;
        long int chaveHash2;
        int treeDepth;
        double evaluation;
}
```

Os atributos *chaveHash1* e *chaveHash2* representam, respectivamente, as chaves *hash* de 64 e 32 bits, que correspondem aos endereços nos quais um registro é guardado na TT. A chave de 32 bits serve para reduzir o número de colisões na TT. O atributo *treeDepth* indica a profundidade já explorada na árvore de busca, considerando a atual configuração do tabuleiro. O atributo *evaluation* ($Q_{(s,a)}$) é o valor retornado ao algoritmo de busca caso ocorra uma transposição. Este atributo também está sujeito a sofrer atualizações na medida em que evolui o algoritmo de busca, pois seu valor corresponde a avaliação do nó.

Para cada tabuleiro s inserido na TT, outros três registros também são inseridos na tabela. Esses três registros adicionais representam configurações simétricas do tabuleiro, correspondendo a rotações de 90°, 180° e 270°, o que enriquece ainda mais o conhecimento armazenado na TT, uma vez que as informações armazenadas também poderão ser utilizadas para esses tabuleiros simétricos. A Figura 4.3 ilustra um exemplo de configurações simétricas que são armazenadas na TT.

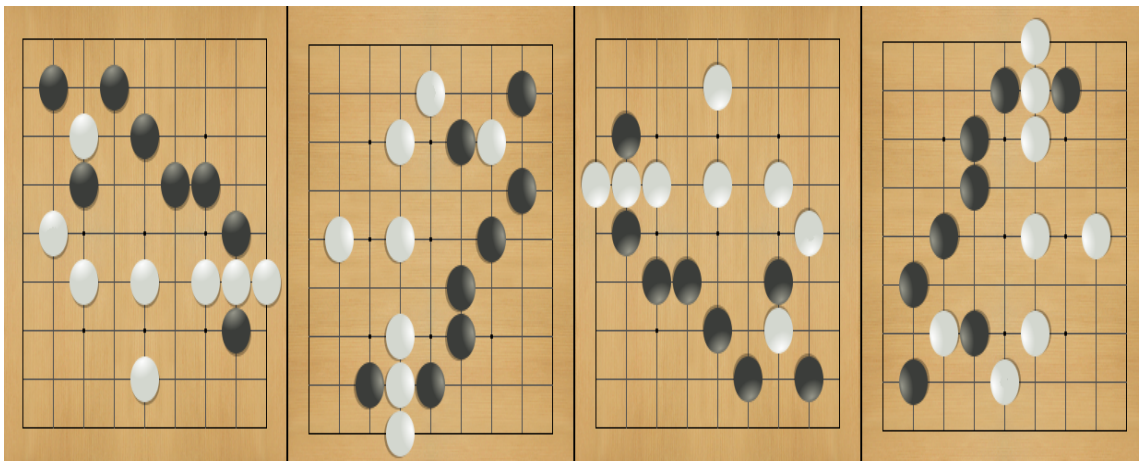

Figura 4.3: Exemplo de configurações simétricas do tabuleiro.

A TT foi implementada como uma tabela *hash* que é uma estrutura de dados que associa chaves a valores. Cada chave representa um estado do tabuleiro do jogo de Go e é associada a informações como avaliação do tabuleiro e profundidade já explorada na árvore. A representação do estado do tabuleiro na forma de chave *hash* é feita utilizando a técnica descrita por Zobrist [Zobrist 1970].

O uso da tabela *hash* permite um acesso rápido aos dados da TT, com complexidade média por operação de *O(1)* e no pior caso *O(N)* (N é o tamanho da TT) [Szwarcfiter and

Markenzon 1994], que pode ocorrer quando não é possível concluir a operação de inserção de dados, pois todas as tentativas colidiram, ou seja, o endereço indicado pela função *hash* está ocupado, como ilustra com um exemplo de colisão a Figura 4.4.

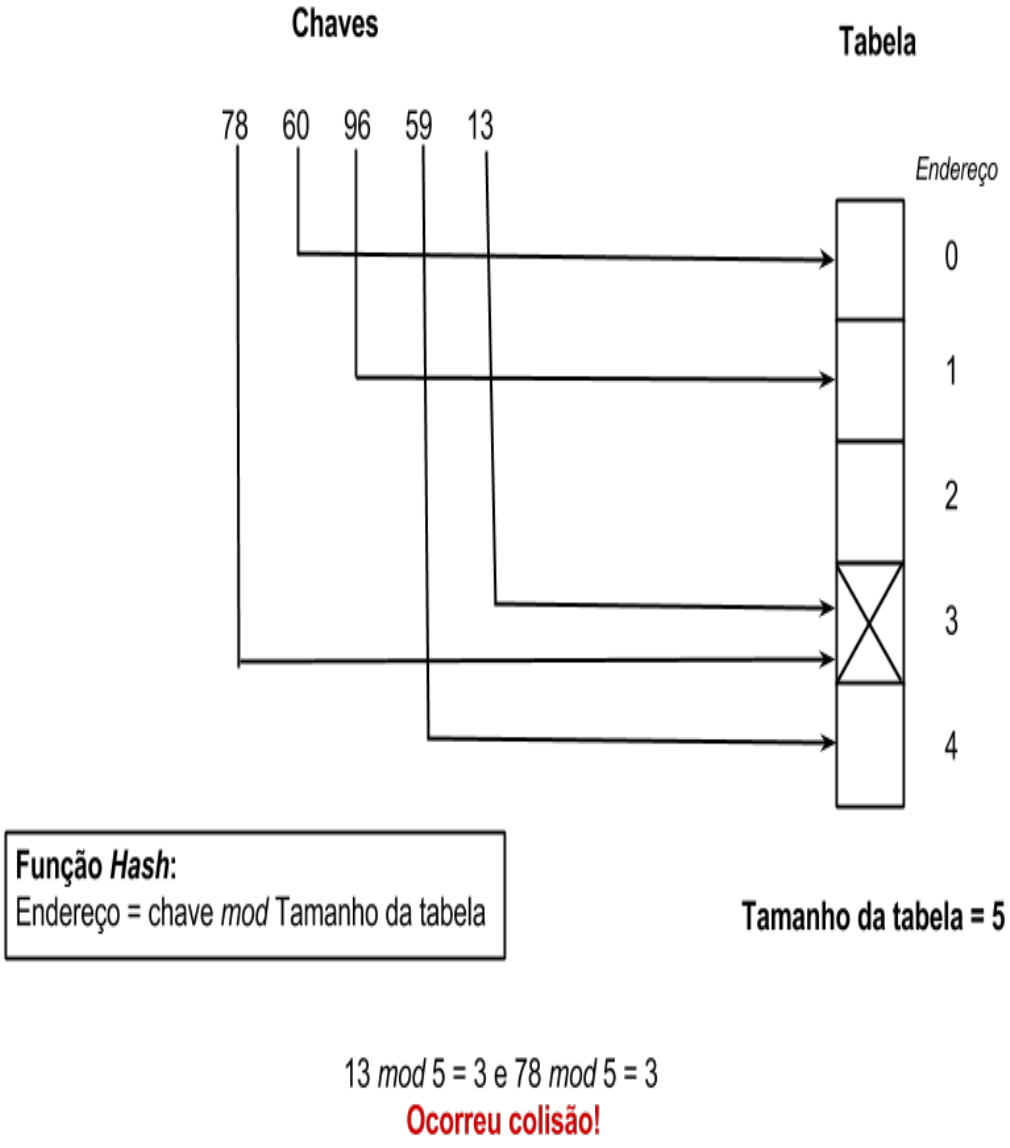

Figura 4.4: Exemplo de colisão.

Uma boa função *hash* deve ser facilmente computável e deve produzir um número baixo de colisões. Uma opção de funçao *hash* funciona da seguinte maneira: divide-se a chave *hash* (x) pelo tamanho da tabela (N) e o resto da divisão (h) é usado como endereço para a TT, como mostra a equação 4.1:

$$h = x \bmod N. \tag{4.1}$$

A TT implementada no agente BTT-Go trata duas formas de colisão: a primeira ocorre quando duas configurações distintas do tabuleiro são mapeadas para a mesma chave *hash*,

o que pode ocorrer em função do operador *mod*. Tal tipo é denominado colisão *Tipo 1*. Para tratar colisões, o agente BTT-Go utiliza duas chaves *hash* (64 e 32 bits), sendo que ambas são geradas pela aplicação do método de Zobrist em identificadores gerados aleatoriamente, conforme explicado a seguir. Neste caso, a função *hash* calcula o endereço H_s de um estado s através da chave de 64 bits. Ao se tentar inserir s na TT, se o endereço H_s está vazio, insere-se s em tal posição na TT. Caso contrário, é checado se a chave *hash* de 32 bits de s coincide com a chave de 32 bits do estado *s'* que ocupa a posição H_s na TT. Se coincidir, isto é, s é igual a *s'* (ocorrência de transposição), permanecerão em H_s as informações referentes à ocorrência mais profunda de s; Se não coincidir, calcula-se um segundo valor de H'_s pela função *hash* para s baseado na sua chave de 32 bits. Neste último caso, se H'_s estiver vazio, s é armazenado em tal endereço; Se não estiver, tal situação representa uma ocorrência de colisão do *Tipo 2*, cuja frequência é tão maior quanto menor for o dimensionamento da TT. O BTT-Go resolve tal tipo de colisão do seguinte modo: se s coincidir com o estado *s"* armazenado em H'_s (transposição), prevalecerão em tal endereço as informações relativas à ocorrência mais profunda de s; Se não coincidir, *s"* permanecerá em H'_s e s será abortado [Caexêta 2008].

4.1.2.3 Módulo de Busca do BTT-Go

O módulo de busca do BTT-Go é responsvel por criar uma eficiente e dinâmica interação entre o algoritmo de busca e a TT.

Uma primeira adaptação feita na busca do jogador Fuego para permitir a redução de supervisão no BTT-Go foi a interrupção da etapa de *play-outs* sempre que ele atinge um estado que esteja armazenado na TT. Neste caso, o valor da TT é retornado como resultado do *play-out*. Caso contrário, quando nenhum nó do *play-out* pertence à TT, o processo de busca prossegue normalmente, e o resultado do *play-out* corresponde àquele do jogo simulado. A partir do fim do *play-out*, o seu resultado é usado para reajustar os valores dos nós da árvore durante a *retropropagação*. Dentre os novos valores produzidos, aquele que corresponde ao nó de *expansão* é inserido na TT. Os demais, correspondentes aos nós das fases de *seleção*, são usados para atualizar os respectivos valores da TT. A segunda adaptação efetuada no BTT-Go para atenuar a supervisão do Fuego pode ser resumida do seguinte modo: durante o reajuste de valores citado acima, sempre que o nó a ser reajustado estiver na TT, a parcela $Q_{(s,a)}^{corrente}$ de reajuste corresponderá ao valor da TT, o qual representa o histórico de avaliação desse nó durante todo o jogo. Tal procedimento difere da busca Fuego, que usa como valor de parcela $Q_{(s,a)}^{corrente}$ de reajuste, a heurística de *prior knowledge* (caso o nó reajustado seja de expansão) ou, então, o valor do nó que retrata o histórico apenas da busca corrente (caso o nó reajustado seja de seleção).

A parcela $Estat.Simulacao_{(a)}$ da equação 2.1 de reajuste é calculada de modo análogo ao do Fuego, exceto quando ocorre uma interrupção do *play-out*, se o nó visitado pelo BTT-Go (nó filho de s obtido pela execução da ação a a partir de s) pertencer à TT,

haverá uma interrupção do *play-out* e tal parcela será desconsiderada. Por outro lado, se o nó visitado não pertencer à TT, ela será calculada tal como é feito no Fuego.

No agente BTT-Go (como no Fuego) é evitada a existência de ciclos, que consistem na existência de *Ko* (repetição consecutiva de dois movimentos em ciclo). Impedir a ocorrência de ciclos é uma medida importante, pois estes podem impossibilitar que se atinja um estado de fim de jogo.

A seguir é apresentado um pseudo-código da busca MC, onde são destacadas as etapas de *seleção*, *expansão*, *play-out* e *retropropagação*.

BuscaMC()

```
1    nodoRaiz = InicializaArvore();
2    Para i = 1 at NroSimulacoes faa
3        nodo = ExecutaSelecao(nodoRaiz);
4        ExecutaExpansao(nodo);
5        resultado = ExecutaPlayOut(nodo);
6        Retropropagacao(resultado);
7    Fim Para;
8    retorna melhorFilho(nodoRaiz);
```

No pseudo-código do algoritmo de busca MC as linhas de instrução definem os seguintes passos:

- **Linha 1:** A árvore de busca é inicializada e é atribuída a configuração do tabuleiro ao *nodoRaiz*;
- **Linha 2:** Nesta linha é definida a estrutura de repetição que tem como limite o número de simulações;
- **Linha 3:** Executa-se a etapa de *seleção*, que escolhe um nó ainda não pertencente a àrvore;
- **Linha 4:** Executa-se a etapa de *expansão*, que insere na árvore o nó filho definido pela etapa de seleção;
- **Linha 5:** Executa-se a etapa de *play-out*, que simula movimentos, sucessivamente, até que se atinja um estado de final de jogo, retornando o resultado que indica vitória ou derrota;
- **Linha 6:** Executa-se a etapa de *retropropagação*, que retropropaga o resultado obtido na etapa de *play-out* pelo caminho simulado na árvore, atualizando os valores dos nós neste caminho.

- **Linha 7:** Indica o fim da estrutura de repetição orientada pelo número de simulações;
- **Linha 8:** Por fim, retorna o nó filho com melhor avaliação.

A próxima seção descreve a 2^a versão do BTT-Go, bem como os detalhes para a aplicação do modelo BT.

4.2 Versão 2 do BTT-Go: Inserção do modelo BT no *play-out*

A segunda versão do BTT-Go consiste na aplicação de uma técnica bayesiana (modelo BT) na etapa de *play-outs* do algoritmo de busca em substituição da *política de play-out* usada no Fuego e na primeira versão do BTT-Go.

Como a TT utilizada na primeira versão atua em todo o processo da busca, guardando as avaliações geradas pelo algoritmo MCTS, foi decidido que esta tabela não será utilizada na segunda versão do BTT-Go, visto que a proposta dessa segunda versão é avaliar o desempenho do agente com a redução de supervisão feita, exclusivamente, durante a simulação de movimentos na etapa de *play-outs*. Assim sendo, uma outra TT é usada unicamente com a finalidade de armazenar o histórico dos movimentos a serem simulados durante tal etapa, os quais são apontados pelo modelo BT. Esta TT usada na segunda versão segue os mesmos processos para geração de chaves e de tratamento de colisões descritos na seção 4.1.2.2 referente à primeira versão.

4.2.1 Adaptação do modelo BT para o jogo de Go

O modelo BT [Hunter 2004] é uma maneira de se escolher uma solução para um problema baseada na maior probabilidade de sucesso alcançada pelas soluções candidatas em torneios efetuados entre si. Para realizar a comparação entre os candidatos, é necessário que se estabeleça um valor de medida que avalie cada candidato. Feito isso, torna-se possível realizar uma competição para saber qual é o melhor.

No BTT-Go, o modelo BT é aplicado para definir o caminho de *play-out*, isto é, definir, dentre os movimentos candidatos possíveis da etapa do *play-out*, aquele que é o mais apropriado para ser simulado. Dessa forma, o valor de medida possui um valor positivo chamado de *grau de força* (γ) do movimento. Por exemplo, considerando os movimentos i e j como sendo candidatos à simulação durante o *play-out*, então aos candidatos i e j estão associados os graus de força γ_i e γ_j, respectivamente. É realizada, então, uma competição entre i e j de modo a identificar qual deles é o mais indicado para simulação. A competição entre os candidatos ocorrerá da seguinte maneira: são calculadas as probabilidades de i vencer j e de j vencer i. Aquele que tiver maior probabilidade de vitória é declarado como

vencedor a ser simulado no *play-out*. Na prática a competição pode ser realizada com um número maior de competidores, sendo que todos irão competir entre si.

Conforme discutido até aqui, no contexto do jogo de Go o problema a ser resolvido pela busca baseada em simulações MC é saber qual jogada realizar a cada momento, sendo candidatos à solução de tal problema os movimentos disponíveis a partir do estado corrente. Assim sendo, é imprescindível definir a maneira na qual será representado um movimento. No contexto do modelo BT, o BTT-Go representa um movimento como um vetor onde cada posição contém a avaliação de um dado atributo presente (ou não) no tabuleiro que resulta da simulação desse movimento a partir do estado corrente. O grau de força de um movimento é calculado pelo produto das avaliações existentes no vetor que o representa.

A seção 4.2.2 descreve com mais detalhes a representação de uma jogada para ser submetida à avaliação do modelo BT.

4.2.2 Representação de movimentos no *play-out*

Uma maneira simples de representar algo é através da descrição de seus atributos, permitindo assim a sua identificação, sendo que em alguns casos é possível compreender o contexto no qual o mesmo está inserido. Dessa maneira, as informações geradas pelos atributos descritos possibilitam uma análise que gerará um conhecimento contextualizado do elemento. No caso em que o contexto é o jogo de Go, um elemento que pode ser representado através dos seus atributos é o movimento, sendo este avaliado em função dos seus atributos considerados importantes. Logo, os atributos de um movimento precisam ser passíveis de serem quantificados. O processo de representação de um movimento no jogo de Go através de atributos requer, necessariamente, que sua execução seja simulada, de modo a se ter ciência do tabuleiro subsequente que ele origina. Depois de concluída esta etapa, a simulação do movimento será desfeita e a configuração do tabuleiro retorna ao estado original.

A maioria dos atributos usados no BTT-Go foram inspirados, principalmente, nas heurísticas do Crazy Stone [Coulom 2007a], conforme visto na seção 3, porém a forma de avaliação dos mesmos são exclusivas do BTT-Go. A fim de tornar factível a representação dos movimentos, o BTT-Go limita o escopo de movimentos candidatos, aos quais será aplicada a avaliação do modelo BT, àqueles que se encaixam em pelo menos um dos padrões das regras da *política de play-out* do Fuego (conforme descrito na seção 3). Para os demais estados, a escolha do movimento será aleatória (como no Fuego). Observa-se, assim, que o BTT-Go insere conhecimento na etapa do *play-out* através das avaliações BT. Os atributos utilizados para definir uma jogada no BTT-Go são:

- *Atari*;
- *Self-atari*;

- *Distância até a borda*;
- *Distância do movimento anterior*;
- *Pass*;
- *Elemento Tático*.

Na próxima seção será explicada a maneira como esses atributos são avaliados e a função de cada um deles.

4.2.3 Avaliações dos atributos

Uma vez que as avaliações dos atributos serão feitas sistematicamente durante cada simulação de jogada do *play-out*, é fundamental que os atributos sejam definidos de uma maneira simples, de modo a não gerar grande impacto no processamento, conforme descrito nas seções seguintes.

Uma vez concluído o cálculo dos atributos relativos a um movimento, o próximo passo é utilizá-los para gerar a avaliação final do movimento, chamada de *grau de força* do movimento. Com tal valor, é possível decidir, entre as jogadas candidatas, qual é a melhor para ser simulada. O cálculo do grau de força é feito de acordo com a equação 4.2.

$$Força = Atributo_1 * Atributo_2 * Atributo_3 \ldots * Atributo_n \tag{4.2}$$

Os atributos *Atari*, *Self-atari*, *Distância até a borda*, *Distância do movimento anterior* e *Pass* tiveram inspiração nas heurísticas utilizadas pelo Crazy Stone, mas suas avaliações são únicas e criadas especificamente para o BTT-Go. O atributo *Elemento Tático*, por sua vez, é um atributo original e criado neste trabalho, tendo sua primeira aplicação feita no BTT-Go. Maiores detalhes sobre os atributos são apresentados nas seções a seguir.

4.2.3.1 Atari

O atributo *Atari* (situação em que uma peça possui liberdade igual à 1), conforme ilustrado na Figura4.5 contabiliza a quantidade de peças do adversário em situação de atari.

O valor da avaliação corresponde ao número de peças na situação de atari, porém para não gerar um valor muito grande para este atributo e consequentemente tornar este atributo mais importante em relação aos demais, foi definida como avaliação máxima a obtida por 4 ocorrências de atari, que é uma quantidade difícil de ocorrer com grande frequência.

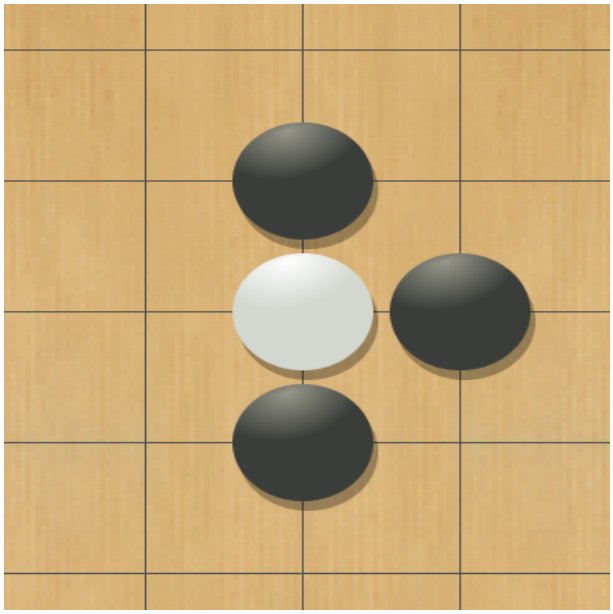

Figura 4.5: Exemplo com peça branca em situação de atari.

4.2.3.2 Self-atari

O atributo *Self-Atari* contabiliza a quantidade de peças do jogador BTT-Go em situação de atari.

O processo de contagem das peças e a avaliação ocorrem de maneira semelhante ao descrito em 4.2.3.1.

4.2.3.3 Distância até a borda

O atributo *Distância até a borda* calcula a distância do movimento que está sendo avaliado até a borda do tabuleiro. Neste caso, a avaliação atribui valores baixos quando o movimento está muito próximo da borda. O atributo atinge valor máximo para distâncias de 3 posições da borda. A partir dessa distância, o valor do atributo começa a decair.

Este atributo permite ao agente se defender de estratégias que dominam completamente regiões livres no tabuleiro, que por estarem mais próximas da borda ficam completamente cercadas mais facilmente, como é apresentado na Figura 4.6.

O adversário não poderá jogar nas posições A e B destacadas na Figura 4.6, uma vez que tais intersecções já pertencem ao jogador de peças pretas. Diz-se, nesta situação, que há dois *eyes* nas posições A e B. Devido ao fato dos *eyes* terem sido criados próximos a borda, foi possível dominar aquela região com facilidade, apesar de não ser uma estratégia com aplicação exclusiva em bordas.

O cálculo da distância até a borda é feito dividindo o tabuleiro em quadrantes e calculando a distância da peça em relação às bordas lateral, superior ou inferior, dependendo do quadrante onde se encontra a peça, como mostra a Figura 4.7.

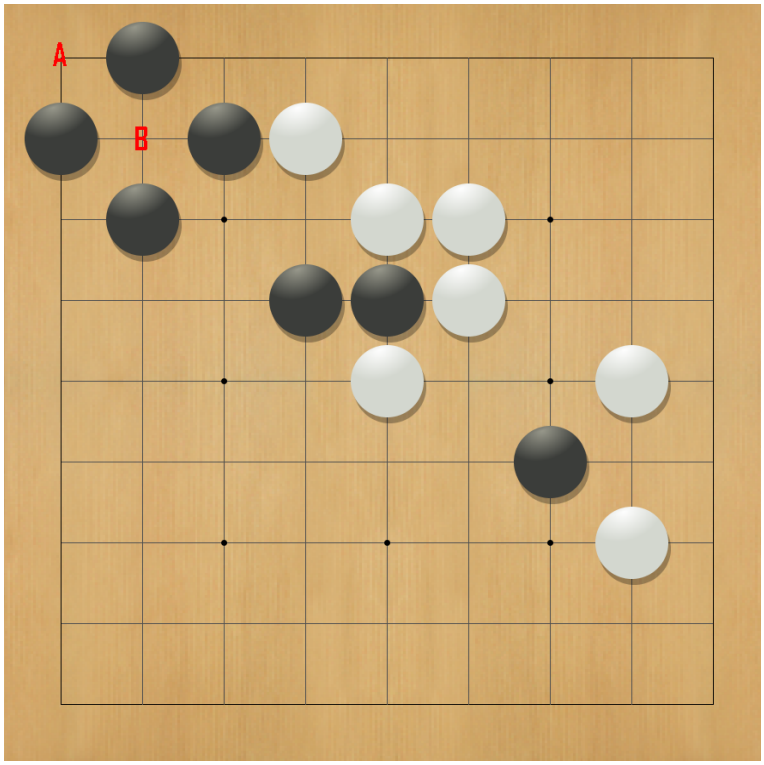

Figura 4.6: *Eyes* próximos a borda do tabuleiro.

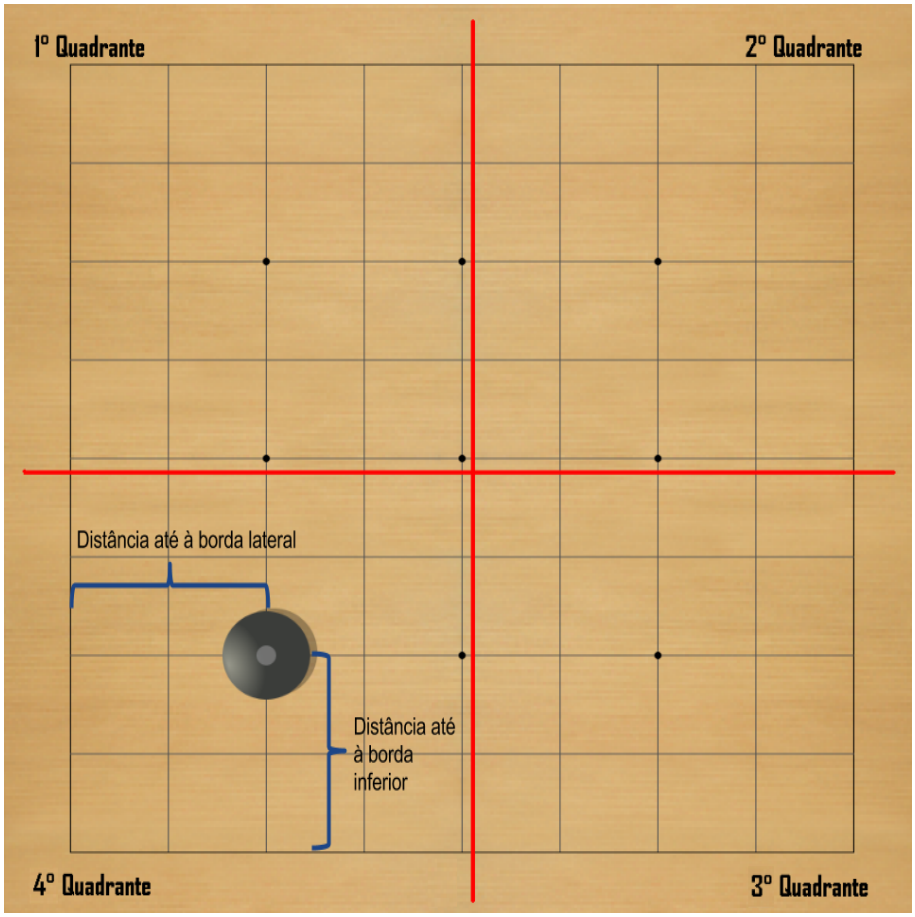

Figura 4.7: Divisão do tabuleiro em quadrantes

4.2.3.4 Distância do movimento anterior

O atributo *Distância do movimento anterior* calcula a distância do movimento que está sendo avaliado em relação ao último movimento executado pelo adversário. Este atributo é importante, pois evita que o agente realize jogadas longe da área onde o adversário está ocupando território no tabuleiro e definindo sua estratégia, algo que pode acontecer facilmente em tabuleiros maiores como o 13x13 e o 19x19.

O cálculo da distância em relação ao movimento anterior é feito pela equação 4.3, sendo que cada movimento passa a ser representado por um par de coordenadas (x, y) e o termo da equação δx calcula a diferença entre os pontos x dos movimentos M_1 e M_2 usados no cálculo, e δy calcula a diferença entre os pontos y. A Figura 4.8 ilustra o cálculo da distância em relação ao movimento anterior.

$$d(M_1, M_2) = |\delta x| + |\delta y| + max(|\delta x|, |\delta y|) \tag{4.3}$$

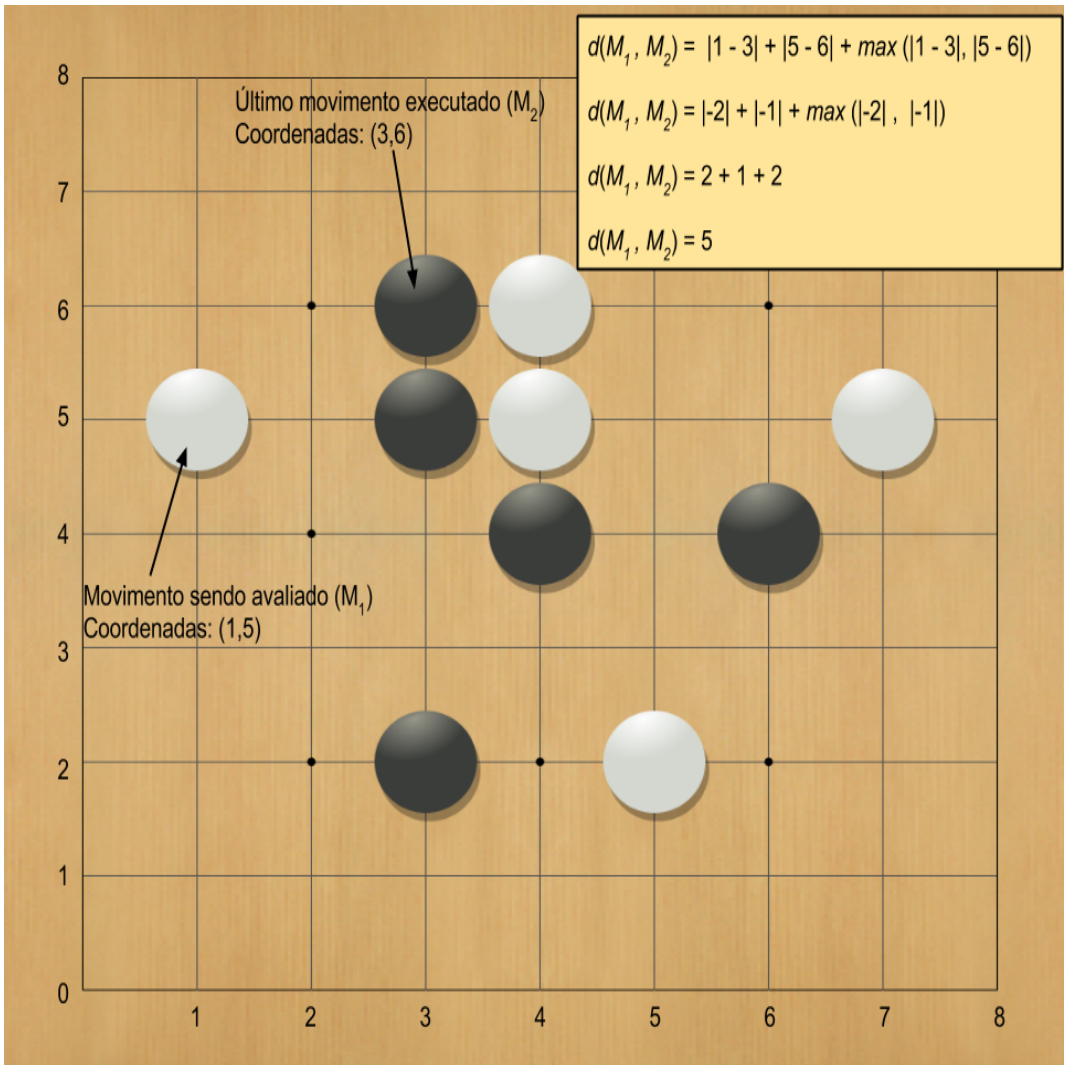

Figura 4.8: Exemplo de cálculo da distância entre movimentos

Neste caso, quanto mais distante do movimento anterior pior será a avaliação.

4.2.3.5 Pass

O atributo *Pass* indica se o movimento anterior foi uma desistência de jogada do adversário. Este é um atributo que fornece informação sobre o contexto do jogo, uma vez que um jogador só realiza o movimento "Pass" se ele não conseguiu encontrar uma jogada para aquele estágio do jogo, que possivelmente é o estágio de fim de jogo.

4.2.3.6 Elemento Tático

O atributo *Elemento Tático* foi criado no contexto do BTT-Go. Ele busca emular o comportamento imprevisível do ser humano presente durante os jogos de tabuleiro, característica que pode ser notada através das mudanças de estratégia e pelas tentativas de enganar o adversário quanto aos propósitos de suas jogadas. No BTT-Go, o valor deste atributo é calculado por uma função chamada *função de imprevisibilidade*, que funciona em conjunto com as avaliações dos demais atributos. A fim de ponderar o impacto do *Elemento Tático*, o BTT-Go efetua um teste de ativação que limita a probabilidade do atributo ser habilitado, em cada filho candidato do nó corrente, a 20%. Caso tal teste estabeleça que o *Elemento Tático* deva ser habilitado em um dado filho candidato, então é feito um sorteio para definir qual dentre os demais atributos será utilizado como base para o cálculo do valor do *Elemento Tático* naquele filho. Tal valor corresponderá à máxima avaliação possível definida para o atributo sorteado.

O atributo *Elemento Tático* também tem o importante papel de incorporar variabilidade no processo de busca pelo melhor movimento, resgatando alguma opção de movimento que poderia ser uma boa alternativa devido a alguma característica que os atributos usados pelo BTT-Go não estão avaliando.

4.2.4 Aplicação do modelo BT

O modelo BT serve como uma política que substitui a *política de play-out* na tarefa de traçar o caminho de simulação durante a etapa de *play-out*. Portanto, tendo que eleger o melhor movimento entre muitos, configura-se um cenário de competição guiado pelo critério do *grau de força* (γ_{m_i}) de um movimento m_i.

As predições são feitas através de uma fórmula que estima a probabilidade de um movimento qualquer vencer os demais e é uma variação da equação 2.4, que mostra a probabilidade de um indivíduo vencer um adversário. Neste caso, calcula-se a probabilidade de um indivíduo m_i vencer um conjunto de movimentos adversários CMA de N elementos, que são todos os demais movimentos disponíveis a partir do nó corrente, como mostra a equação 4.4.

$$P(m_i \text{ vencer } \forall m \in CMA) = \frac{\gamma_{m_i}}{\sum_{j=0}^{N} \gamma_{m_j} + \gamma_{m_i}} \quad (4.4)$$

A seguir é apresentado um pseudo-código que descreve o cálculo apresentado na equação 4.2 para definir os movimentos a serem simulados durante um *play-out*.

CalculoProbabilidadeVitoria(m_i, CMA)

```
1   ForçaConjunto = 0;
2   Para j = 1 até N faça
3       ForçaConjunto = ForçaConjunto + γ_j;
4   Fim Para;
5   retorna γ_mi / (ForçaConjunto + γ_mi);
```

A seguir é apresentada a arquitetura definida para a aplicação do modelo BT, descrevendo as interações entre os módulos e a ordem de processamento.

4.2.5 Arquitetura geral da segunda versão do BTT-Go

O modelo BT visa incorporar conhecimento na etapa de *play-out*. Sendo assim, todas as avaliações, seleções de movimentos e dados utilizados da TT são apresentados pela arquitetura ilustrada na Figura 4.9.

Uma vez iniciada a etapa de *play-out*, é feita uma análise do tabuleiro corrente, #1 (seta 1 da Figura 4.9) (que a partir de agora passar a ser chamado de tabuleiro *play-out*). Por esta análise, identificam-se as posições ocupadas no tabuleiro *play-out*. Tal informação é fornecida ao módulo TT que verificará se existe algum registro armazenado na tabela e se ele está ligado à configuração atual do tabuleiro. Se existir dados na TT, #3, então a jogada indicada na tabela é simulada, o *play-out* corrente é interrompido (sendo efetuados os devidos reajustes da *retropropagação*) e o tabuleiro *play-out* é atualizado, #4, para que o próximo passo de simulação de jogada *play-out* seja iniciado. Caso não existam dados na TT vinculados à configuração corrente do tabuleiro *play-out*, #2, então a execução prossegue normalmente com o módulo de definição de movimentos. Tal módulo é responsável por definir se, a partir daí, o próximo movimento será definido aleatoriamente (#11 e #12) ou por avaliação BT, #5, conforme descrito na seção 4.2.2. Caso seja por BT, o fluxo entra em um processo iterativo representado na parte hachurada da figura, que visa calcular o *grau de força* dos movimentos envolvidos, conforme definido nos seguintes

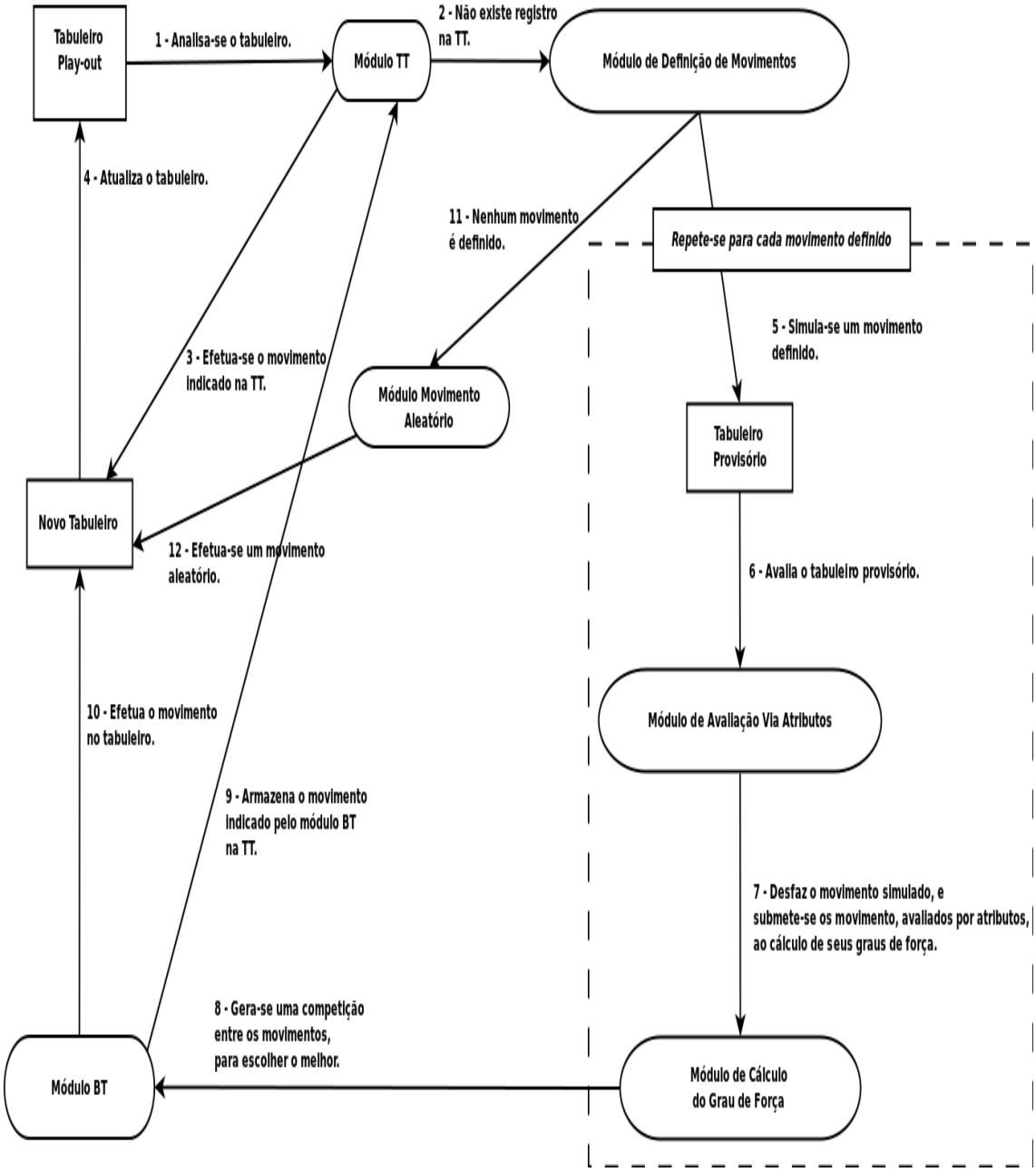

Figura 4.9: Arquitetura da segunda versão do BTT-Go.

passos:

- Simula-se um movimento selecionado em um tabuleiro provisório, que possui a mesma configuração do tabuleiro *play-out*, #5;
- O tabuleiro provisório é submetido a uma avaliação, #6, sendo que esta avaliação é feita pelo módulo de avaliação via atributos, que é responsável por gerar para cada movimento selecionado uma nota em função de um determinado atributo;
- O movimento simulado no tabuleiro provisório é desfeito, para que este tabuleiro esteja disponível com a configuração correta (igual a do tabuleiro *play-out*) para uma posterior simulação e avaliação de movimento, #7;

- O movimento que acabou de ser avaliado em função de atributos tem, em seguida, o valor de seu grau de força calculado pelo módulo de cálculo do grau de força, #7.
- Agora repetem-se os passos anteriores para outro movimento selecionado, até que todos esses movimentos já tenham seu grau de força calculado.

Concluído o cálculo do grau de força de todos os movimentos candidatos, terá incio o processo de competição que determinará o movimento a ser simulado no *play-out* corrente, #8. O resultado é obtido pelo módulo BT, que calcula a probabilidade de todos os movimentos candidatos vencer os demais, sendo que aquele com maior probabilidade é o escolhido e simulado no tabuleiro, #10. Logo em seguida, um registro, que indica o movimento selecionado pelo módulo BT é criado e este é armazenado na TT, #9. Por fim, atualiza-se o tabuleiro *play-out* com uma nova peça inserida no tabuleiro pela simulação do movimento. Salienta-se que o ciclo descrito na Figura 4.9 se repete a cada passo do *play-out* em que se deve estabelecer o próximo movimento a ser simulado.

As seções seguintes descrevem detalhadamente cada módulo da arquitetura da segunda versão do BTT-Go.

4.2.5.1 Módulo TT

Para armazenar a informação processada pelo modelo BT durante o *play-out* nesta segunda versão do agente, é utilizada uma TT implementada com a estrutura semelhante à da primeira versão.

A seguir resume-se as informações relevantes relativas à tal TT.

- Geram-se duas chaves *hash* a partir das posições ocupadas no tabuleiro, uma de 64 bits e outra de 32 bits;
- Com a chave *hash* de 64 bits, acessa-se o endereço na TT calculado pela equação 4.1;
- Utiliza-se a chave *hash* de 32 bits para mais uma verificação do endereço na TT e para tratar colisões;
- Verifica-se se o endereço na TT já está ocupado;
  - Se a posição está ocupada, efetua o movimento indicado na TT e retorna à etapa de *play-out* (dessa forma, nesta segunda versão do agente, não há interrupção de *play-out*);
  - Se a posição não está ocupada, continua a execução passando pelos seguintes módulos: Definição de Movimentos, Avaliação Via Atributos, Cálculo do Grau de Força, e, por fim, o Módulo BT que indica o movimento a ser efetuado;
- Cria-se um registro a ser armazenado na TT, o qual indica o movimento indicado pelo módulo BT;

- Calcula-se o endereço de entrada na TT a partir das chaves *hash*;
- Armazenam-se as informações pertinentes na TT;
- Por fim, retorna-se a execução do programa para a etapa de *play-outs*.

É importante salientar que devido ao atributo *Elemento Tático* não é armazenado na tabela registros das configurações simétricas do tabuleiro, pois como esse atributo é ativado em função de uma probabilidade, então uma transposição pode ou não ter este atributo ativado.

4.2.5.2 Módulo de Definição de Movimentos

O módulo de definição de movimentos é a primeira "filtragem" realizada entre os candidatos a movimentos que podem ser efetuados em um *play-out*. Este módulo gera um conjunto de movimentos baseados nas políticas de *play-out*, que consistem em:

- Nakade heuristic: *Nakade* pode ser, literalmente, entendido como um movimento interno, pois se refere a uma jogada dentro de um *eye* criado pelo adversário, impedindo que se crie dois *eyes* a partir de um [Hoock et al. 2010], como mostra a Figura 4.10.

  Pela figura, observa-se que a posição representada por A é a posição onde se realiza um movimento *nakade*, pois para o jogador com peças brancas uma jogada nessa posição impede a formação de dois *eyes* a partir do que já existe;
- Atari capture: A política *Atari Capture* visa gerar um movimento que capture uma peça do adversário que esteja na situação de atari;
- Atari defense: A política *Atari Defense* tem o objetivo de gerar um movimento que proteja alguma peça do jogador que esteja na situação de atari;
- Low liberty: A política *Low Liberty* gera movimentos que provocam redução da liberdade das peças do adversário;
- Pattern move: A política *Pattern move* tem o objetivo de gerar movimentos que criam padrões já definidos com dimensão 3x3 [Enzenberger et al. 2010];
- Capture move: A política *Capture move* gera movimentos que provoquem a captura de uma peça adversária;

Quando nenhum movimento é gerado pelas políticas apresentadas, então gera-se um movimento aleatório ou efetua-se um movimento *Pass*, este movimento é usado quando o jogador vai deixar de realizar uma jogada na rodada corrente. Isso ocorre quando o jogador não encontra mais alternativas de jogadas, caracterizando um provável fim de jogo.

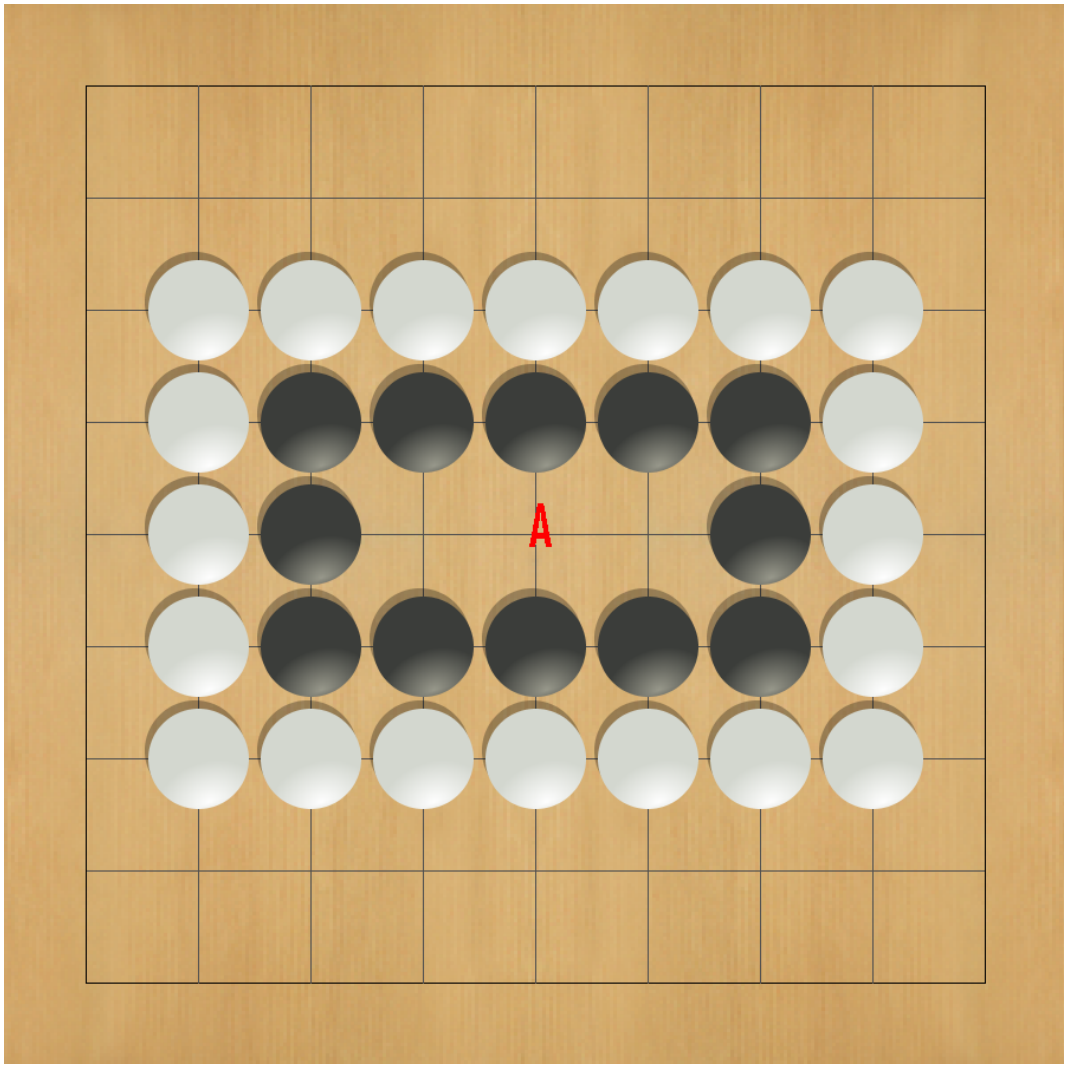

Figura 4.10: Exemplo de um movimento nakade.

4.2.5.3 Módulo de Avaliação via Atributos

O módulo de avaliação via atributos atua sobre o conjunto de movimentos gerado pelo módulo de definição. Neste módulo cada movimento é simulado e avaliado de acordo com atributos previamente definidos e apresentados na seção 4.2.2, com suas formas de avaliação descritas na seção 4.2.3.

4.2.5.4 Módulo do Cálculo do Grau de Força

O módulo do cálculo do grau de força utiliza o resultado da avaliação feita pelo módulo de avaliação via atributos e aplica, a cada movimento candidato, a equação 4.2 geradora de seu *grau de força*.

4.2.5.5 Módulo BT

O módulo BT é responsável por gerar uma competição entre os movimentos candidatos. Através dessa competição é gerada uma probabilidade que retrata a chance de cada

um deles vencer os demais (considerando a sua força).

Quando concluída a competição entre todos os movimentos, então analisam-se os dados probabilísticos gerados, sendo escolhido o movimento com maior probabilidade de vencer os demais.

4.2.6 Eventual inconveniente do modelo BT

O modelo BT realiza uma suposição relativa as vitórias do movimentos que pode não ser verdadeira em toda situação. Tal suposição refere-se à transitividade nas vitórias, ou seja, considerando que o movimento m_1 vence o movimento m_2, e que o movimento m_2 vence o movimento m_3, então o movimento m_1 vence o movimento m_3. Tal afirmação pode não ser verdadeira sempre, pois o modelo BT toma sua decisão considerando a probabilidade de um movimento vencer todos os demais. Portanto, o modelo não consegue considerar a situação em que o movimento m_1 vence o movimento m_2, o movimento m_2 vence o movimento m_3, e o movimento m_3 vence o movimento m_1, visto que pode existir uma probabilidade, mesmo que pequena, de que isto aconteça.

4.3 Versão 3 do BTT-Go: Aplicação conjunta de TT e BT

A terceira versão do BTT-Go combina a primeira e a segunda versões do agente, permitindo avaliar o impacto de se utilizar ambas estratégias em conjunto.

Assim sendo, nesta nova versão do BTT-Go utilizam-se duas TTs: a primeira TT (a ser chamada de TT_1 na versão três) exerce o mesmo papel daquela descrita na primeira versão do agente (seção 4.1); a segunda TT (aqui denominada TT_2) opera em conjunto com a política BT, conforme descrito na segunda versão do agente (seção 4.2).

O motivo para se usarem duas TTs na terceira versão do BTT-Go consiste no fato de que existem duas avaliações feitas pelo agente: uma delas diretamente vinculada a etapa de construção da árvore MC (baseada no MCTS + RAVE + histórico) e, a segunda, vinculada ao modelo BT da etapa de *play-out* (baseada no *grau de força* do movimento). Portanto, cada uma das TTs é responsável por guardar informações distintas, além de serem usadas em etapas diferentes da busca. Salienta-se que o uso de uma única TT para ambas avaliações, poderia trazer situações de inconsistência nos dados armazenados na mesma, pois, em casos de transposição em que um mesmo estado tenha aparecido durante a *construção da árvore* e durante o *play-out*, não haveria como armazenar, no mesmo endereço da TT, seus respectivos valores calculados em ambas as etapas, por estes cálculos ocorrerem em momentos distintos.

É importante salientar que as situações previstas para interrupção de *play-out* na primeira versão do agente também são mantidas nesta sua terceira versão.

Uma vez apresentadas as três versões do BTT-Go, no próximo capítulo cada uma destas versões será submetida a experimentos e comparada a outros jogadores de Go.

Capítulo 5

Experimentos e Resultados

Para avaliar o agente BTT-Go em suas três versões, foram realizados testes que consistem em torneios de 100 jogos realizados contra os jogadores Fuego e Gnu Go [Bump 2003], visando analisar suas taxas de vitória em tabuleiros de tamanho 9x9, 13x13 e 19x19. Cada torneio foi realizado em um cenário de teste diferente, sendo que em 50 dos jogos o agente BTT-Go joga com peças pretas e nos outros 50 o agente joga com peças brancas. O objetivo de tal dinâmica é avaliar o agente nas duas situações, visto que as peças pretas dão a vantagem de início de jogo. Convém salientar que, mesmo com o uso do *komi* (pontos dados ao jogador com peças brancas para compensar a desvantagem de não ter iniciado o jogo), a desvantagem do branco não é plenamente compensada. No presente trabalho foi usado um valor de *komi* igual a 6,5 pontos, que é o padrão definido pelo GoGui que é a interface gráfica usada no trabalho [Enzenberger et al. 2010]. A configuração do computador usado nos experimentos é: *Intel Core 2 Quad* 2.4 GHz com 8 GB de RAM.

Este capítulo está organizado da seguinte maneira:

- Na seção 5.1 são apresentados os resultados dos cenários de teste realizados com a primeira versão do BTT-Go;
- Na seção 5.2 são apresentados os resultados dos cenários de teste realizados com a segunda versão do BTT-Go;
- Na seção 5.3 são apresentados os resultados dos cenários de teste realizados com a terceira versão do BTT-Go;

Os cenários de teste seguem uma sequência de numeração e cada cenário está descrito na seção correspondente a versão testada do agente.

5.1 Testes com a 1^a versão do BTT-Go

A primeira versão do BTT-Go foi explicada na seção 4.1, sendo que os testes feitos nesta versão, além de avaliar o agente em função da taxa de vitória, também o avalia

em função do tempo de processamento do algoritmo de busca. Com o objetivo de focar a análise apenas no efeito produzido pelo uso dos dados oriundos da TT, nos testes da primeira versão do BTT-Go, nenhum dos agentes usa *opening book*.

Nos testes efetuados, o tempo médio gasto pelo BTT-Go para realizar uma jogada, calculado em função de 20 movimentos, foi em torno de 30% inferior ao tempo gasto pelo agente Fuego. Analisando os dados, o BTT-Go teve um tempo médio igual à 5,34 segundos, considerando tabuleiros de tamanho 9x9, 13x13 e 19x19. Enquanto que o Fuego teve tempo médio, nos referidos tamanhos de tabuleiro, igual à 7,645 segundos, uma diferença de 2,305 (Fuego gasta 30% a mais de tempo). Isso se deve, à redução do número de *play-outs* decorrente da interrupção dos episódios de simulação quando ocorre alguma transposição, implicando no uso dos dados da TT. A quantidade de simulações do BTT-Go foi, em média, inferior a quantidade do Fuego em 30% (tabela 5.1). Como esta redução está vinculada ao uso da TT, este ganho também reflete a frequência de uso de dados da tabela.

Tabela 5.1: Comparação quantidade média de simulações

	9x9		**13x13**		**19x19**	
Movimentos	**BTT-Go**	**Fuego**	**BTT-Go**	**Fuego**	**BTT-Go**	**Fuego**
mov 1	66647	67899	40218	50611	12089	28312
mov 2	45974	84202	40301	51419	18135	18785
mov 3	59743	89796	41198	55835	13383	24023
mov 4	56863	90692	30013	59584	10206	26322
mov 5	31817	67749	37551	59856	13684	25429
mov 6	80067	86377	30230	49965	19128	20402
mov 7	42932	125743	29739	39815	16920	12629
mov 8	41941	65115	36916	43035	14333	16323
mov 9	59484	51998	30108	46323	18141	18620
mov 10	49455	51878	26740	45167	17046	15448
mov 11	40516	34461	17928	68625	18145	23405
mov 12	27963	55945	21869	54495	13101	24843
mov 13	32835	70117	23697	43226	8333	24340
mov 14	20854	69515	19755	22676	17822	19355
mov 15	27313	49223	42137	21581	16484	26666
mov 16	34052	67819	46925	19819	19686	20983
mov 17	33466	71916	19974	29968	20161	16315
mov 18	34893	41641	9940	28430	19920	18324
mov 19	37708	69572	20247	25450	15575	20468
mov 20	43139	73604	10025	17021	11450	23661
Média Simulações	**43383,1**	**69263,1**	**2877,55**	**41644,05**	**15687,1**	**21232,65**

Pelos dados apresentados na tabela 5.1 calcula-se que no tabuleiro 9x9 o BTT-Go realizou menos 25.880 simulações em média que o Fuego, o que corresponde à aproximadamente 37% a menos de simulações. No tabuleiro 13x13 observa-se que o BTT-Go

realizou menos 12.869,5 simulações em média que o Fuego, o que corresponde à aproximadamente 30% a menos de simulações. No tabuleiro 19x19 observa-se que o BTT-Go realizou menos 15.687,1 simulações em média que o Fuego, o que corresponde a aproximadamente 26% a menos de simulações. Analisando os dados, observa-se que em média o BTT-Go realiza 31% a menos de simulações que o Fuego considerando os três tamanhos de tabuleiro.

Com o propósito de analisar as taxas de vitória contra os jogadores Fuego e Gnu Go, foram elaborados três cenários de teste para avaliar a primeira versão do BTT-Go.

O primeiro cenário consiste em um torneiro de jogos realizados entre a primeira versão do BTT-Go e o jogador Gnu Go em cada dimensão de tabuleiro. A Tabela 5.2 mostra a taxa de vitória contra seu adversário, em que o BTT-Go, jogando com peças pretas, venceu 98%, 98% e 77,6%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Gnu Go em que o BTT-Go jogava com peças brancas, as taxas de vitória foram 100%, 88% e 58%, respectivamente, para os tabuleiros 9x9, 13x13 e 19x19. Estes resultados confirmam um desempenho significativamente superior, em todas as situações, em relação ao jogador Gnu Go.

Tabela 5.2: Cenário 1 – Taxa de vitória BTT-Go(versão 1) X Gnu Go

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	98% (± 2)	100% (± 0)
13x13	98% (± 2)	88% (± 4,6)
19x19	77,6% (± 6)	58% (± 7)

O cenário 2 consiste no torneio entre o BTT-Go (versão 1) e o Fuego, sendo que a Tabela 5.3 mostra a taxa de vitórias do BTT-Go em tal cenário. Conforme mostrado na tabela, o BTT-Go, jogando com peças pretas, venceu 44%, 42% e 42%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Por outro lado, jogando com peças brancas, o BTT-Go venceu 34,7%, 34% e 26%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Os resultados mostram que apesar de todas as medidas para redução da supervisão do agente, o BTT-Go obteve resultados que mostram que o jogador continua competitivo mesmo jogando contra o Fuego que possui um nível de supervisão superior. Para exemplificar esta situação destaca-se a taxa de vitória do BTT-Go jogando com peças pretas no tabuleiro 19x19, sendo que este é o caso mais complexo do jogo de Go, em que a árvore de busca do jogo possui alto nível de ramificação (ver Tabela 1).

Tabela 5.3: Cenário 2 – Taxa de vitória BTT-Go (versão 1) X Fuego

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	44% (± 7)	34,7% (± 6,8)
13x13	42% (± 7)	34% (± 6,7)
19x19	42% (± 7)	26% (± 6,2)

Finalmente, no cenário 3, o principal objetivo é verificar se a informação obtida da TT pelo BTT-Go (versão 1) possui, de fato, "qualidade" quando comparada ao conhecimento expresso pelas *políticas de play-out* herdadas do Fuego. Para realizar essa avaliação o BTT-Go sofre uma pequena modificação que consiste no seguinte: na etapa de *play-out*, cada vez que um movimento, indicado pela avaliação resgatada da TT, não for o mesmo que o movimento indicado pelas *políticas de play-out*, esta avaliação, contida na TT, sofrerá uma redução de 30% em seu valor, caracterizando uma punição aplicada nos movimentos que não coincidirem com o indicado pelas *políticas de play-out*.

Este cenário de teste analisa a importância dos dados armazenados na TT, visto que a punição serve para atribuir maior importância as *políticas de play-out*. A punição utilizada neste teste serve para refletir no jogo a seguinte ideia: se as avaliações armazenadas na TT não são importantes para o agente, então puní-las através da redução de seus valores será indiferente para o agente, que deverá manter o mesmo desempenho.

Os resultados do cenário 3, apresentados na Tabela 5.4, mostram que os dados armazenados na TT realmente possuem uma "qualidade" que permite substituir as *políticas de play-out*, uma vez que o desempenho do BTT-Go neste cenário teve uma considerável redução decorrente da aplicação da punição usada neste cenário. Nos jogos realizados contra o Fuego, quando o BTT-Go jogava com peças pretas, ele venceu 56%, 30% e 20%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19, e quando o BTT-Go jogou com peças brancas venceu 26%, 30% e 20%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Destacando que o BTT-Go teve uma redução na taxa de vitória de 22% em jogos realizados no tabuleiro 19x19, jogando com peças pretas. Em jogos no tabuleiro 9x9, o BTT-Go teve uma melhora na sua taxa de vitória jogando com peças pretas (comparado ao valor obtido no cenário 2, Tabela 5.3), isto é explicado pelo fato de que no tabuleiro 9x9 a quantidade de estados na árvore de busca é bem menor que nos tabuleiros 13x13 e 19x19, então a avaliação armazenada na TT mesmo com a punição continuou satisfatória. Além do mais, neste caso o agente joga com peças pretas que é quem inicia o jogo, podendo assim realizar, inicialmente, ótimos movimentos, caracterizando antecipadamente uma vitória. Salientando também que a política de uso da TT (começar a usar a TT depois de 50% do tabuleiro ocupado) pode ter adiado a aplicação da punição para um momento em que o resultado do jogo já estaria quase definido.

Tabela 5.4: Cenário 3 – Taxa de vitória BTT-Go(versão 1 alterada) X Fuego

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	56% (± 7)	26% (± 6,2)
13x13	30% (± 6,5)	30% (± 6,5)
19x19	20% (± 5,7)	26% (± 6,2)

5.2 Testes com a 2^a versão do BTT-Go

A segunda versão do BTT-Go foi apresentada na seção 4.2, e os testes feitos com esta versão avaliam o agente BTT-Go em função das taxas de vitória obtidas em torneios realizados contra os agentes Fuego e Gnu Go.

O quarto cenário de teste consiste no torneio entre o BTT-Go (versão 2) e o jogador Gnu Go. A Tabela 5.5 mostra a taxa de vitória contra seu adversário, em que o BTT-Go, jogando com peças pretas, venceu 91,7%, 100% e 79,6%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Gnu Go em que o BTT-Go jogava com peças brancas as taxas de vitória foram 98%, 86,3% e 46%, respectivamente, para os tabuleiros 9x9, 13x13 e 19x19. Estes resultados mostram um desempenho superior do BTT-Go em relação ao jogador Gnu Go, apesar de uma pequena redução no percentual de vitórias se comparado com a primeira versão do BTT-Go, por exemplo, no tabuleiro 19x19 ocorreu uma redução de 12% com o agente jogando com peças brancas.

Tabela 5.5: Cenário 4 – Taxa de vitória BTT-Go(versão 2) X Gnu Go

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	91,7% (± 4)	98% (± 2)
13x13	100% (± 0)	86.3% (± 4,9)
19x19	79,6% (± 5,8)	46% (± 7)

O quinto cenário de teste consiste no torneio entre o BTT-Go (versão 2) e o jogador Fuego. A Tabela 5.6 mostra a taxa de vitória, em que o BTT-Go, jogando com peças pretas, venceu 60%, 42% e 44%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Fuego em que o BTT-Go jogava com peças brancas as taxas de vitória foram 24%, 32% e 30%, respectivamente, para os tabuleiros 9x9, 13x13 e 19x19. Pelos resultados dos testes neste cenário observa-se que o modelo BT manteve o agente competitivo, mas as taxas de vitória não foram tão diferentes dos resultados apresentados pela primeira versão.

Tabela 5.6: Cenário 5 – Taxa de vitória BTT-Go (versão 2) X Fuego

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	60% (± 6,9)	24% (± 6)
13x13	42% (± 7)	32% (± 6,6)
19x19	44% (± 7)	30% (± 6,5)

O sexto cenário consiste de jogos realizados entre o Fuego e o BTT-Go (versão 2), mas o BTT-Go joga sem utilizar *prior knowledge* e nem *opening book*, ou seja, o BTT-Go enfrenta o Fuego em desvantagem. A Tabela 5.7 mostra a taxa de vitória, em que o BTT-Go, jogando com peças pretas, venceu 52%, 34% e 44%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Fuego em que o BTT-Go jogava com peças brancas as taxas de vitória foram 28%, 44% e 24%, respectivamente, para os tabuleiros 9x9, 13x13

e 19x19. Pelos resultados dos testes, observa-se que mesmo sem os recursos do *prior knowledge* e do *opening book*, o modelo BT consegue gerar boas avaliações do tabuleiro, agregando qualidade a busca MC e mantendo o agente competitivo mesmo no tabuleiro 19x19. A maior perda neste cenário, comparados com os dados do quinto cenário, foi no tabuleiro 13x13 com o agente jogando com peças pretas, neste caso houve uma redução de 8% na taxa de vitória, contudo neste mesmo tabuleiro o agente proporcionou o importante ganho de 12% em sua taxa de vitória jogando com peças brancas

Tabela 5.7: Cenário 6 – Taxa de vitória BTT-Go(versão 2 em desvantagem) X Fuego

Tabuleiro	Peças Pretas	Peças Brancas
9x9	52% (± 7,1)	28% (± 6,3)
13x13	34% (± 6,7)	44% (± 7)
19x19	44% (± 7)	24% (± 6)

5.3 Testes com a 3^a versão do BTT-Go

A terceira versão do BTT-Go foi descrita na seção 4.3 e consiste na associação da versão 1 com a versão 2 do BTT-Go. Para avaliar esta versão foram realizados jogos contra os jogadores Fuego e Gnu Go.

O sétimo cenário consiste no torneio efetuado entre o BTT-Go (versão 3) e o jogador Gnu Go. A Tabela 5.8 mostra a taxa de vitória contra seu adversário, em que o BTT-Go, jogando com peças pretas, venceu 100%, 100% e 86.7%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Gnu Go em que o BTT-Go jogava com peças brancas as taxas de vitória foram 100%, 93,3% e 60%, respectivamente, para os tabuleiros 9x9, 13x13 e 19x19. Estes resultados mostram um desempenho superior em relação ao jogador Gnu Go e que foram melhores que os obtidos pela segunda versão do agente.

Tabela 5.8: Cenário 7 – Taxa de vitória BTT-Go(versão 3) X Gnu Go

Tabuleiro	Peças Pretas	Peças Brancas
9x9	100% (± 0)	97,9% (± 2,1)
13x13	96% (± 2,8)	92% (± 3,8)
19x19	79,6% (± 5,8)	60% (± 6,9)

O oitavo cenário consiste de jogos realizados entre BTT-Go (versão 3) e o jogador Fuego. A Tabela 5.9 mostra a taxa de vitória, em que o BTT-Go, jogando com peças pretas, venceu 66%, 42% e 54%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Fuego em que o BTT-Go jogava com peças brancas as taxas de vitória foram 32%, 38% e 28%, respectivamente, para os tabuleiros 9x9, 13x13 e 19x19. Pelos resultados dos testes neste cenário, observa-se que a associação da versão 1 com a versão 2 do BTT-Go foi bem sucedida, tendo como principal argumento o ganho na taxa de vitória

do agente no tabuleiro 19x19 jogando com peças pretas (54%). Isso mostra que a leve perda no percentual de vitórias ocasionada pela interrupção de simulações e observada nos testes da primeira versão (ver Tabela 5.2), devido ao uso da TT, foi reduzida com os ganhos proporcionados pelo modelo BT.

Tabela 5.9: Cenário 8 – Taxa de vitória BTT-Go (versão 3) X Fuego

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	66% (± 6.7)	32% (± 6,6)
13x13	42% (± 7)	38% (± 6,9)
19x19	54% (± 7)	28% (± 6,3)

O nono cenário consiste de jogos realizados entre o Fuego e o BTT-Go (versão 3), neste cenário o BTT-Go joga sem utilizar *prior knowledge* e nem *opening book*. A Tabela 5.10 mostra a taxa de vitória, em que o BTT-Go, jogando com peças pretas, venceu 60%, 54% e 50%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Fuego em que o BTT-Go jogava com peças brancas as taxas de vitória foram 30%, 36% e 26%, respectivamente, para os tabuleiros 9x9, 13x13 e 19x19. Pelos resultados dos testes neste cenário, observa-se que mesmo em desvantagem o BTT-Go mantem bons índices de vitória, mesmo nos tabuleiros maiores, isso reflete o sucesso da redução de supervisão obtida pela TT e pela incorporação do modelo BT. Uma vez que, o BTT-Go obteve boas taxas de vitória mesmo abrindo mão de alguns dos recursos que caracterizam a supervisão herdada do jogador Fuego.

Tabela 5.10: Cenário 9 – Taxa de vitória BTT-Go(versão 3 em desvantagem) X Fuego

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	60% (± 6,9)	30% (± 6,5)
13x13	54% (± 7)	36% (± 6,8)
19x19	50% (± 5,7)	26% (± 6,2)

O décimo cenário consiste de jogos realizados entre o Fuego e o BTT-Go (versão 3), neste cenário o BTT-Go joga sem utilizar o atributo *Elemento Tático* (ver seção 4.2.3.6). A Tabela 5.11 mostra a taxa de vitória, em que o BTT-Go, jogando com peças pretas, venceu 64%, 40% e 44%, respectivamente, nos tabuleiros 9x9, 13x13 e 19x19. Nos jogos contra o Fuego em que o BTT-Go jogava com peças brancas as taxas de vitória foram 38,8%, 38% e 24%, respectivamente, para os tabuleiros 9x9, 13x13 e 19x19. Pelos resultados dos testes neste cenário, e comparando-os com os resultados do oitavo cenário (Tabela 5.9) observa-se que o *Elemento Tático* incorporou ganhos ao agente, destacando o ganho de 10% de vitórias no tabuleiro 19x19 com o BTT-Go jogando com peças pretas. Isso é justificado pelo fato do atributo *Elemento Tático* permitir diversificar as jogadas e então realizar um movimento que as avaliações prévias não haviam identificado. Observa-se a maioria das taxas de vitória sofreram redução em relação ao cenário em que o atributo *Elemento Tático* está presente.

Tabela 5.11: Cenário 10 – Taxa de vitória BTT-Go(versão 3 sem elemento tático) X Fuego

Tabuleiro	**Peças Pretas**	**Peças Brancas**
9x9	64% (± 6,8)	38,8% (± 6,5)
13x13	40% (± 7)	38% (± 6,8)
19x19	44% (± 5,7)	24% (± 6,2)

Pelos resultados apresentados neste capítulo, observa-se que os objetivos propostos foram atingidos, e com a implementação do BTT-Go em três versões foi possível analisar o impacto e a melhora correspondente a cada técnica aplicada.

Comparando os teste feitos na primeira versão do BTT-Go com os feitos com a versão final (terceira versão), em que o BTT-Go joga contra o Fuego, observa-se ganhos expressivos no percentual de vitória, especialmente no tabuleiro 19x19, sendo que na primeira versão o BTT-Go possui taxa de vitória de 42% e na terceira versão essa taxa subiu para 54%, o ganho se repetiu em relação a segunda versão em que o percentual de vitória é de 44% (segunda versão).

Uma vez apresentados os resultados dos testes realizados com o agente BTT-Go, no próximo capítulo é feita uma conclusão do trabalho e uma descrição das próximas atividades a serem realizadas.

Capítulo 6

Conclusão e Trabalhos Futuros

Neste trabalho foi apresentado o agente jogador de Go denominado BTT-Go. A criação deste agente pode ser descrita como um processo de aprimoramento do algoritmo de busca MCTS, utilizando para tal TT e o modelo BT.

O agente BTT-Go, com a implementação da primeira versão, conseguiu reduzir a supervisão do bem sucedido jogador automático Fuego, pela incorporação de uma TT, operando em conjunto ao algoritmo de busca. Tal TT serve como ferramenta usada para armazenar e atualizar os valores dos estados ao longo de um jogo, na medida em que eles reaparecem por transposição. Assim sendo, os valores dos nós armazenados na TT contém todo um teor histórico de suas avaliações efetuadas no contexto de um jogo (não apenas o teor informativo da busca corrente ou o teor heurístico que são usados pelo Fuego).

O algoritmo de busca do Fuego tem seu bom desempenho ligado à quantidade de simulações de jogadas realizadas (quanto mais, maior é a acuidade das avaliações dos estados durante a busca) e ao uso de heurísticas de *prior-knowledge* e *play-outs* como ferramentas auxiliares na busca pelo melhor movimento. Nesta direção, o agente BTT-Go, sempre que possível, substitui o uso das heurísticas do Fuego pelas informações providas na TT, o que propicia uma boa acuidade na avaliação dos nós (comprovado pelos resultados de testes apresentados na Tabela 5.4), apesar da redução nos recursos de supervisão do Fuego. Conforme apresentado na seção 5.1, o bom desempenho do BTT-Go em torneios contra o Fuego e o Gnu Go, inclusive em tabuleiros 19x19, comprovam que a técnica aqui utilizada permitiu uma maior autonomia ao agente em termos de supervisão sem comprometer significativamente seu desempenho.

A implementação da segunda versão do BTT-Go proporcionou ganhos ao agente mantendo a competitividade do agente. Dessa maneira, com os testes da segunda versão apresentados na seção 5.2, ficou evidente que a aplicação do modelo BT agregou conhecimento à etapa de *play-out* do algoritmo de busca.

A terceira versão do BTT-Go permitiu avaliar a agregação da primeira com a segunda versão, visto que ambas têm objetivos semelhantes que inclui a redução de supervisão, porém feito de maneira diferente, a primeira versão através da TT aplicada em conjunto

ao algoritmo de busca MC, e a segunda versão pela aplicação do modelo BT na etapa de *play-out*. Já na terceira versão foi possível verificar o sucesso da associação das duas primeiras versões do BTT-Go como mostra os resultados apresentados na Tabela 5.9 e também pelos resultados da Tabela 5.10 em que o BTT-Go consegue bons resultados mesmo não utilizando *prior knowledge* e *opening book*. O atributo *Elemento Tático* foi comprovadamente uma boa estratégia e proporcionou aumento na taxa de vitória do agente. Dessa forma, o BTT-Go conseguiu atingir seus objetivos que são a redução de supervisão e tornar o agente mais competitivo, sendo que esse sucesso foi comprovado pelos testes.

Como trabalhos futuros, considera-se a adaptação do sistema de busca do agente a um ambiente de alto desempenho, o que deve melhorar muito a sua capacidade operacional, principalmente na fase de *play-out*. Essa adaptação é promissora, visto que a busca MCTS tem seu bom desempenho ligado à quantidade de simulações realizadas. Desta maneira, será possível aumentar o número de simulações sem comprometer o tempo de jogadas do agente. Também considera-se o uso de uma única TT para armazenar dados do algoritmo MCTS e do modelo BT. Para tanto é necessário realizar alterações na estrutura da TT e na política de uso da mesma. Nesta proposta será possível incorporar dados na TT em dois momentos, um referente a avaliação da busca MCTS e o outro referente a avaliação do modelo BT. Com isso, também poderá aumentar os casos de colisões a serem tratados, passando a considerar também os casos em que os registros na TT já possuem as duas avaliações preenchidas e os casos em que só possuem uma das avaliações, nesta situação aquele que tiver as duas avaliações preenchidas prevalecerá. Com relação ao atributo *Elemento Tático*, pretende-se analisar o resultado de uso de diferentes taxas de ativação do atributo, o que permitirá observar o comportamento do agente quando este é submetido a uma maior taxa de imprevisibilidade nas jogadas.

Referências Bibliográficas

[Allis 1994] Allis, V. L. (1994). *Searching for solutions in games and artificial intelligence.* PhD thesis, University of Limburg in Maastricht.

[Bouzy and Chaslot 2005] Bouzy, B. and Chaslot, G. (2005). Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 go. In *CIG*.

[Brügmann 1993] Brügmann, B. (1993). Monte carlo go. Citeseer.

[Bump 2003] Bump, D. (2003). Gnugo home page. `http://www.gnu.org/software/gnugo`.

[Caexêta 2008] Caexêta, G. S. (2008). 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.

[Campos and Langlois 2003] Campos, P. and Langlois, T. (2003). Abalearn: Efficient self-play learning of the game abalone. In *INESC-ID, Neural Networks and Signal Processing Group*.

[Castro Neto 2007] Castro Neto, H. d. (2007). Ls-draughts–um sistema de aprendizagem de jogos de damas baseado em algoritmos genéticos, redes neurais e diferenças temporais.

[Childs et al. 2008] Childs, B. E., Brodeur, J. H., and Kocsis, L. (2008). Transpositions and move groups in monte carlo tree search. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 389–395. IEEE.

[Coulom 2007a] Coulom, R. (2007a). Computing elo ratings of move patterns in the game of go. In *Computer games workshop*.

[Coulom 2007b] Coulom, R. (2007b). Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer.

[Enzenberger and Müller 2010] Enzenberger, M. and Müller, M. (2010). A lock-free multithreaded monte-carlo tree search algorithm. In *Advances in Computer Games*, pages 14–20. Springer.

[Enzenberger et al. 2010] Enzenberger, M., Muller, M., Arneson, B., and Segal, R. (2010). Fuego-an open-source framework for board games and go engine based on monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):259–270.

[Gelly and Silver 2007] Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM.

[Gelly and Silver 2008] Gelly, S. and Silver, D. (2008). Achieving master level play in 9 x 9 computer go. In *AAAI*, volume 8, pages 1537–1540.

[Gelly and Silver 2011] Gelly, S. and Silver, D. (2011). Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875.

[Gelly et al. 2006] Gelly, S., Wang, Y., Munos, R., Teytaud, O., et al. (2006). Modification of uct with patterns in monte-carlo go.

[Goldberg and Holland 1988] Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99.

[Harrison 2010] Harrison, B. A. (2010). *Move Prediction in the Game of Go*. PhD thesis, Citeseer.

[Hoock et al. 2010] Hoock, J.-B., Lee, C.-S., Rimmel, A., Teytaud, F., Wang, M.-H., and Teytaud, O. (2010). Intelligent agents for the game of go. *Computational Intelligence Magazine, IEEE*, 5(4):28–42.

[Huang et al. 2011] Huang, S.-C., Coulom, R., and Lin, S.-S. (2011). Monte-carlo simulation balancing in practice. In *Computers and Games*, pages 81–92. Springer.

[Hunter 2004] Hunter, D. R. (2004). Mm algorithms for generalized bradley-terry models. *Annals of Statistics*, pages 384–406.

[Junior and Julia 2014] Junior, E. V. and Julia, R. M. S. (To Be Published 2014). Btt-go: an agent for go that uses a transposition table to reduce the simulations and the supervision in the monte-carlo tree search.

[Kierulf 1990] Kierulf, A. (1990). *Smart Game Board: a workbench for game playing programs, with Go and Othello as case studies*. Verlag der Fachvereine.

[Müller 1995] Müller, M. (1995). *Computer Go as a sum of local games: an application of combinatorial game theory*. PhD thesis, Swiss Federal Institute OF Technology Zürich.

[Plaat et al. 1996] Plaat, A., Schaeffer, J., Pijls, W., and De Bruin, A. (1996). Exploiting graph properties of game trees. In *AAAI/IAAI, Vol. 1*, pages 234–239.

[Rimmel et al. 2010] Rimmel, A., Teytaud, O., Lee, C.-S., Yen, S.-J., Wang, M.-H., and Tsai, S.-R. (2010). Current frontiers in computer go. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):229–238.

[Ross 2006] Ross, S. M. (2006). *Introduction to probability models*. Access Online via Elsevier.

[Shannon and Hsu 1949] Shannon, C. E. and Hsu, T.-s. (1949). Programming a computer for playing chess. In *National IRE Convention*.

[Szwarcfiter and Markenzon 1994] Szwarcfiter, J. L. and Markenzon, L. (1994). *Estruturas de Dados e seus Algoritmos*, volume 2. Livros Técnicos e Científicos.

[Tesauro 1995] Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

[van der Werf 2005] van der Werf, E. C. D. (2005). *AI techniques for the game of Go*. UPM, Universitaire Pers Maastricht.

[Walker 2000] Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email marilyn a. walker walker@ research. att. com at&t shannon laboratory 180 park ave., bldg 103, room e103. *Journal of Artificial Intelligence Research*, 12:387–416.

[Zobrist 1970] Zobrist, A. L. (1970). A new hashing method with application for game playing. *ICCA journal*, 13(2):69–73.