UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Renan Motta Goulart

Um Método Kernel para Estimativa de Densidade e sua Aplicação em Jogos de Repetição

Juiz de Fora

2017

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Renan Motta Goulart

Um Método Kernel para Estimativa de Densidade e sua Aplicação em Jogos de Repetição

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Raul Fonseca Neto
Coorientador: Saul de Castro Leite

Juiz de Fora
2017

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Motta Goulart, Renan .
Um Método Kernel para Estimativa de Densidade e sua Aplicação em Jogos de Repetição / Renan Motta Goulart. -- 2017.
58 p.

Orientador: Raul Fonseca Neto
Coorientador: Saul de Castro Leite
- Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2017.

1. Jogos de repetição. 2. Teoria dos jogos. 3. Previsão de sequências. 4. Estimativa de densidade. 5. Descida do gradiente. I. Fonseca Neto, Raul, orient. II. de Castro Leite, Saul, coorient. III. Título.

Renan Motta Goulart

Um Método Kernel para Estimativa de Densidade e sua Aplicação em Jogos de Repetição

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 1 de Setembro de 2017.

BANCA EXAMINADORA

Prof. D.Sc. Raul Fonseca Neto - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Marcelo Costa Pinto e Santos
Instituto Federal Sudeste de Minas Gerais

Prof. D.Sc. Heder Soares Bernadino
Universidade Federal de Juiz de Fora

Dedicado a minha mãe.

AGRADECIMENTOS

Gostaria de começar agradecendo principalmente a minha mãe pelo apoio durante todos estes anos, mesmo antes de eu entrar na faculdade. Gostaria de agradecer especialmente ao meu orientador, Raul Fonseca Neto, e ao meu co-orientador, Saul de Castro Leite, por terem me ajudado durante o mestrado e por possibilitarem que eu pudesse aprender muito durante este período da minha vida. Agradeço também aos professores, tanto da graduação quanto do mestrado, por terem feito nascer em mim uma paixão pela ciência da computação e estarem dispostos a me ajudar quando tive dificuldades.

Gostaria também de agradecer aos amigos que fiz durante o mestrado, tanto do #teamnenc quanto do #teamrapa, por propiciarem boas risadas e boa companhia durante os almoços no R.U., dentro e fora do PGCC.

Por fim gostaria de agradecer a UFJF por ser uma instituição que permitiu que estes últimos seis anos, tanto da graduação quando do mestrado, fossem agradáveis e cheios de recordações boas que levarei para a vida, desde as pessoas diferentes que pude conhecer quanto os tempos realizando trabalhos e jogando com amigos no infocentro. Também agradeço por possibilitar que eu fizesse amizades que ainda durarão por muitos anos, talvez por toda a vida.

"Prepare for worst.... hope for the best."

Anônimo

RESUMO

Jogos de repetição é um ramo de Teoria dos Jogos, em que um jogo é jogado repetidas vezes pelos jogadores. Neste cenário, assume-se que os jogadores nem sempre jogam de modo ótimo ou podem estar dispostos, se possível, a colaborar. Neste contexto é possível um jogador analisar o comportamento dos oponentes para encontrar padrões. Estes padrões podem ser usados para aumentar o lucro obtido pelo jogador ou detectar se o oponente está disposto a realizar uma colaboração mutualmente benéfica.

Nesta dissertação é proposto um novo algoritmo baseado em *kernel* de similaridade capaz de prever as ações de jogadores em jogos de repetição.

A predição não se limita a ação do próximo *round*, podendo prever as ações de uma sequência finita de *rounds* consecutivos. O algoritmo consegue se adaptar rapidamente caso os outros jogadores mudem suas estratégias durante o jogo. É mostrado empiricamente que o algoritmo proposto obtém resultados superiores ao estado da arte atual.

Palavras-chave: Teoria dos Jogos. Jogos de Repetição. Previsão de Sequências. Estimativa de Densidade. Descida do Gradiente.

ABSTRACT

Repeated games is a branch of game theory, where a game can be played several times by the players involved. In this setting, it is assumed that the players do not always play the optimal strategy or that they may be willing to collaborate. In this context it is possible for a player to analyze the opponent's behaviour to find patters. These patterns can be used to maximize the player's profit or to detect if the opponent is willing to collaborate. On this dissertation it is proposed a new algorithm based on similarity *kernel* capable of predicting the opponent's actions on repeated games. The prediction is not limited to the next *round*'s action, being able to predict actions on a finite sequence of *rounds*. It is able to adapt rapidly if the opponents change their strategies during the course of a game. It is shown empirically that the proposed algorithm achieves better results than the current state of the art.

Keywords: Game Theory. Repeated Games. Sequence Prediction. Density Estimate. Gradient Descent.

LISTA DE FIGURAS

LISTA DE SÍMBOLOS

A	Conjunto de ações a serem utilizadas.
a	Ação utilizada pelo jogador. $a \in A$
b	Ação utilizada pelo oponente. $b \in A$
c	Ação utilizada pelo jogador no próximo *round*. $a \in A$
d	Ação utilizada pelo oponente no próximo *round*. $b \in A$
M	Memória Longa.
$M(a)$	Parte da memória longa conténdo apenas sequências que foram sucedidas pela ação a.
m	Memória Curta.
m'	Uma sequência de memória qualquer.
s	Tamanho da memória curta.
P_a	Probabilidade da ação a ser utilizado.
$P_a(m)$	Probabilidade de a ser utilizado com memória curta m.
$m_(a, b)$	Memória curta em que no *round* passado o jogador utilizou a e o oponente b.
$f_a(m)$	Frequência em que a ação a sucedeu a sequência m.
$G(a, b)$	Lucro obtido com o jogador utilizando a ação a e o oponente b.
α	Conjunto de ponderações de subsequências.
α_j	Ponderação da subsequência j.
$\partial H(m, \alpha)$	Entropia calculada utilizando a memória curta m e o conjunto de ponderações α.
F	Função objetivo.

LISTA DE ABREVIATURAS E SIGLAS

ELPH Entrophy Learning Prunning Hypothesys Algorithm

WoLF Win or Learn Fast

AC Always Cooperate

AD Always Defect

BA Brainless Alteration

EBA Evil Brainless Alteration

G Grudger

SG Soft Grudger

TFT Tit for Tat

ETFT Evil Tit for Tat

ATFT Anti Tit for Tat

TF2T Tit for 2 Tats

5TM 5 is Too Much

GC General Cooperator

P Pavlov

2TT 2 To Trust

E2TT Evil 2 To Trust

2TB 2 To Betray

E2TB Evil 2 To Betray

TTP Three Then Punish

TD Trust Distrust

EGET Evil Good Evil Trust

SUMÁRIO

1 INTRODUÇÃO

Estratégias clássicas para jogos não consideram o modo de jogar do oponente, se importando apenas com o jogo em questão. Esta dissertação tem como foco o estudo da aplicação de métodos de aprendizado em jogos de repetição. A utilização destes métodos possibilita obter um lucro maior do que seria possível com os métodos tradicionais. O algoritmo proposto nesta dissertação traz contribuições obtendo resultados melhores que o estado da arte atual.

1.1 MOTIVAÇÃO

É comum encontrar situações similares que acontecem frequentemente em diferentes momentos e que exigem tomada de decisões. Pode ser a simples escolha de um itinerário influenciada pela variação da quantidade de carros em uma certa rua ou até mesmo uma decisão de investimento no mercado acionário influenciada pelas variações do preço de compra e venda das ações. Estes problemas tem como objetivo maximizar ou minimizar alguma função variável, como exemplo o tempo de viajem ou o lucro obtido por uma venda. Em casos mais simples tem-se que as variações são causadas por um agente sem objetivos que não se importa com o valor da variável em questão, porém em casos mais complexos pode ser que este agente também tenha um objetivo que possa o fazer entrar em conflito. Um exemplo disto é um grupo de pessoas que optam simultaneamente pela escolha de um mesmo trajeto ou um vendedor que procura vender pelo maior preço e um comprador que procura gastar o mínimo possível. Em certos casos, o único modo de atingir um objetivo pessoal resulta do detrimento de outros, enquanto em outros casos pode ser que a melhor forma de atingir o objetivo seja encontrar um meio termo que seja considerado bom para ambas as partes.

A Teoria de Jogos (NEUMANN; MORGENSTERN, 1944) procura modelar e estudar situações que envolvem interações de competição e cooperação. Esta dissertação tem seu foco nos jogos de repetição. Eles diferem de outros tipos de jogos devido a possibilitarem que seja feita uma análise do histórico de ações do oponente com o intuito de aumentar o lucro. O histórico de ações é representado como uma sequência. Métodos de inteligência artificial, mais precisamente de aprendizado de máquinas, podem ser usados para

encontrar padrões no histórico e com isto prever quais serão as próximas ações a serem realizadas pelo oponente.

Apesar de existirem abordagens para jogos de repetição, elas não são numerosas e contém pontos fracos como um alto custo de tempo (BOWLING; VELOSO, 2002) para se adaptarem ou terem um custo de memória muito elevado (JENSEN et al., 2005b). O método proposto nesta dissertação busca obter bons resultados promovendo uma mediação entre o custo de tempo e de memória superando os problemas encontrados nos algoritmos atualmente utilizados.

1.2 OBJETIVOS E CONTRIBUIÇÕES

Este trabalho apresenta contribuições para as áreas de predição de sequências e teoria dos jogos. É apresentado um algoritmo para jogos de repetição que é capaz de adaptar seu modo de jogar utilizando o histórico do jogo, que o permite se ajustar à política utilizada pelo oponente para escolher seus ações. Este algoritmo oferece vantagens em relação ao estado de arte atual tanto em relação a qualidade dos resultados obtidos quanto no consumo de memória. Ele também pode ser facilmente estendido para realizar busca de profundidade em relação aos jogos futuros, o que pode ser necessário para obter melhores resultados contra certos tipos de jogos e estratégias de oponentes.

1.3 ORGANIZAÇÃO

Esta dissertação está organizada do seguinte modo, no capítulo *2* será explicado conceitos básicos de Teoria dos Jogos que são necessários para o entendimento do problema abordado, após isto será mostrado no capítulo *3* quais algoritmos são utilizados para atacar o problema proposto junto com suas limitações. Em seguida no capítulo *4* será apresentado o método desenvolvido para jogar jogos de repetição, os resultados empíricos dele serão mostrados no capítulo *5*. Por fim será discutido no capítulo *6* as conclusões deste trabalho e trabalhos futuros.

2 CONCEITOS PRELIMINARES

A seguir será feita uma breve introdução à Teoria dos Jogos e serão apresentados termos essenciais para o melhor entendimento deste trabalho, cujo foco está nos jogos de repetição.

2.1 TEORIA DOS JOGOS

Teoria dos Jogos pode ser interpretada como sendo o estudo sobre cooperação e competição (NEUMANN; MORGENSTERN, 1944). Inicialmente inspirada por interações econômicas como, por exemplo, o estudo sobre monopólios (SELTEN, 1973; ARMSTRONG; PORTER, 1989). Porém com o passar do tempo ela também começou a ser usada em outras áreas e aplicações como biologia (SMITH, 1982), o estudo do ser humano em sociologia, e psicologia (BICCHIERI, 2006). Jogos são definidos formalmente como um conjunto de estados e ações que ao serem usadas resultam em uma recompensa (TUROCY; STENGEL, 2002). Cada mudança de estado pode resultar em um lucro ou prejuízo para cada jogador envolvido, que geralmente possui o objetivo de maximizar seu lucro. Prejuízo pode ser considerado como sendo um lucro negativo.

Jogos podem ser definidos para um ou mais jogadores. Um jogo com apenas um jogador, pode ser interpretado como sendo um problema de decisão, em que o jogador quer encontrar quais são as melhores escolhas para atingir seu objetivo. Geralmente jogos são definidos para dois ou mais jogadores, sendo este o foco deste trabalho. O estudo da interação entre mais de dois jogadores é chamada de sistemas multiagente (TUROCY; STENGEL, 2002).

2.2 FORMAS DE REPRESENTAÇÃO

Jogos podem ser representados de várias formas diferentes, sendo as mais comuns por meio de uma matriz ou por meio de uma árvore.

2.2.1 FORMA NORMAL

A representação em forma de matriz é chamada de forma estratégica ou normal (TUROCY; STENGEL, 2002). Nela todas as ações de cada jogador são listadas junto com o

lucro que ambos obterão para cada opção possível de ações. Esta forma de representação é utilizada frequentemente para representar jogos envolvendo dois oponentes, sendo um deles denominado de jogador das linhas e o outro como jogador das colunas. O jogador das linhas deve escolher uma das linhas da matriz enquanto o das colunas escolhe uma coluna, o elemento resultante da escolha desta linha e coluna contém o lucro que cada jogador receberá. Normalmente o primeiro valor é referente ao jogador das linhas e o segundo ao das colunas, por exemplo, no jogo representado na figura 2.2.1, caso o jogador das linhas escolha a ação T e o jogador das colunas escolha a ação C tem-se que o elemento resultante tem marcado os valores *1* e *-2*. Logo, o jogador das linhas obterá lucro *1* enquanto o jogador das colunas obterá um lucro de *-2*. Em jogos representados na forma normal, os jogadores geralmente fazem suas escolhas simultaneamente sem ter conhecimento prévio da escolha do seu oponente. Este tipo de representação é frequentemente utilizada em jogos de repetição, sendo eles o foco deste estudo. Na Figura 2.2.1, pode-se ver a matriz que representa o jogo *dilema do prisioneiro*.

$$G = \begin{array}{|c|c|c|} \hline & T & C \\ \hline T & -1/-1 & 1/-2 \\ \hline C & -2/1 & 0/0 \\ \hline \end{array}$$

Figura 2.2.1: Matriz do jogo *dilema do prisioneiro*, este jogo será discutido em mais detalhes na sub-seção 2.4.2. O primeiro valor de cada célula indica o lucro do jogador das linhas enquanto o segundo valor indica o lucro do jogador das colunas.

2.2.2 FORMA EXTENSIVA

Outra forma muito utilizada para representar jogos é por meio de árvores, sendo esta forma chamada de forma extensiva (TUROCY; STENGEL, 2002). Esta forma é usualmente utilizada para jogos que possuem mais de uma tomada de decisão por cada jogador para chegar ao fim. Além disto, tem-se que normalmente as escolhas são feitas de forma alternada e cada jogador tem conhecimento das ações passadas dos outros jogadores. Esta forma de representação é mais comum em jogos competitivos, como no jogo de tabuleiro Xadrez. Um exemplo de uma árvore para o *jogo da velha* pode ser vista na Figura 2.2.2.

O estado inicial do jogo é representado pelo nó raiz enquanto que os nós folhas representam os possíveis estados finais do jogo junto com o lucro final que cada jogador

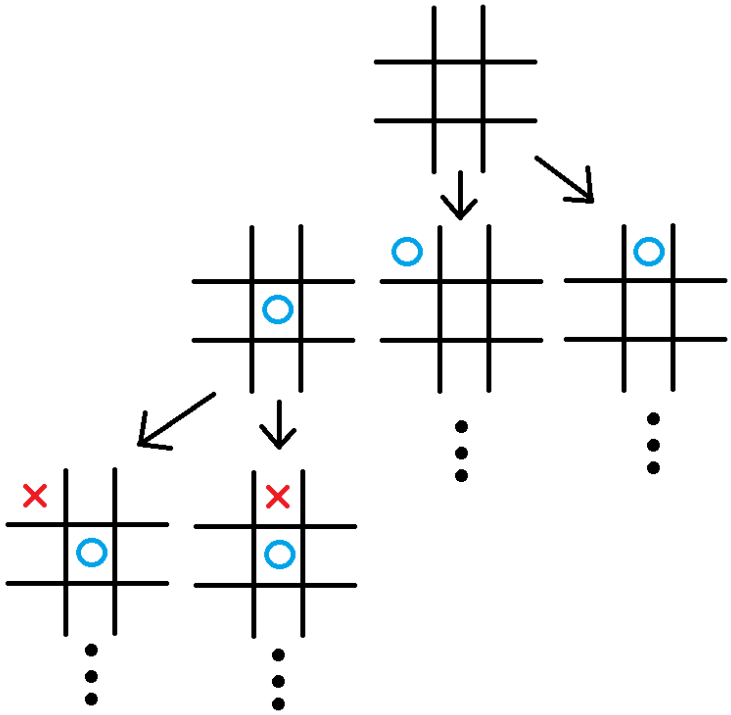

Figura 2.2.2: Árvore dos primeiros níveis do *Jogo da Velha*.

receberá. O caminhamento na árvore funciona do seguinte modo: começando pela raiz, um jogador deve escolher qual será a ação dele; esta ação leva o jogo para um novo estado representado por um dos nós filhos da raiz; após isto, o outro jogador deve escolher qual ação ele irá realizar, que leva o jogo para um novo estado representado por um nó filho do nó atual. Este processo é realizado sequencialmente com a escolha das ações sendo alternada entre os jogadores até se chegar em um nó folha, sendo este o indicador do fim do jogo que contém os valores que cada jogador receberá de lucro.

2.3 TIPOS DE JOGOS

Conforme a interação entre jogadores, os jogos podem ser definidos como cooperativos ou competitivos. A diferença base entre essas duas classes é dada pelo modo como cada jogador alcança seu objetivo. Em jogos cooperativos tem-se que a melhor forma dos jogadores conseguirem maximizar seu lucro é encontrando um modo de cooperarem juntos, um jogador só consegue aumentar seu lucro se os demais jogadores também concordarem em jogar de modo a aumentar o lucro de todos. Enquanto isso, em jogos competitivos, tem-se que cada jogador quer apenas aumentar o seu lucro pessoal, muitas vezes tendo como objetivo lucrar mais que os outros jogadores. Geralmente nestes jogos quando um

jogador obtém lucro o outro obtém prejuízo.

Jogos onde o total de lucro é equivalente ao total de prejuizo são chamados de jogos de soma zero (NEUMANN; MORGENSTERN, 1944). Este nome provém devido ao fato de que o lucro total, obtido pela soma da recompença de cada jogador, sempre será igual a zero ou a alguma constante. Este tipo de jogo é mais comum em cenários competitivos pois o único modo de aumentar o lucro coincide com fazer com que os oponentes sofram um prejuízo. Quando a soma do lucro total de todos os jogadores muda ao decorrer do jogo ele é definido como sendo de soma não zero (TUROCY; STENGEL, 2002). Este tipo de jogo é mais comum em jogos não cooperativos onde geralmente se quer maximizar o lucro total, que depende de todos.

Nem todo jogo precisa ser considerado "justo". Jogadores podem possuir conjuntos diferentes de ações ou ter diferentes estados iniciais. Desta forma, é possível que um jogador já esteja destinado a lucrar mais que outro. Por isto que o objetivo de cada jogador geralmente é maximizar o seu lucro, não importando se ele está sendo maior ou menor que o dos oponentes.

2.3.1 JOGOS DE REPETIÇÃO

O foco deste trabalho está nos jogos de repetição. Um jogo de repetição é definido como sendo um jogo em que após terminar, volta para o estado inicial e é recomeçado (MERTENS, 1986). Este processo é repetido por um determinado número de vezes, em que cada uma é definida como um *round*. Neste caso, o objetivo dos jogadores é relacionado ao seu lucro médio por jogo, obtido após todos os *rounds* serem realizados. Em jogos de repetição é comum se utilizar jogos que terminam após poucas ações, ou mesmo após somente uma ação por jogador. Em termos da representação, pode-se utilizar tando a forma extensiva quanto a forma normal para representar um jogo de repetição. A única diferença é em relação à necessidade de se armazenar também o lucro obtido em cada *round*. Realizar a repetição de um jogo supostamente simples pode levar a um aumento na quantidade de estratégias envolvidas. Além disso, o lucro máximo que pode ser obtido em um jogo de repetição depende não somente do jogo em si, mas também do comportamento do oponente. Desta forma a melhor estratégia para o jogo passa a depender também da estratégia do oponente. Isso será evidenciado em mais detalhes no Capítulo *3*.

2.4 JOGOS ESTUDADOS

Nesse trabalho, dois jogos foram escolhidos para o estudo dos algoritmos: *pedra papel tesoura* e *dilema do prisioneiro*. Estes jogos são frequentemente utilizados na literatura para o estudo de estratégias adaptativas para jogos de repetição.

2.4.1 PEDRA PAPEL TESOURA

No jogo a *pedra papel tesoura*, cada jogador possui três ações. Para cada ação existe uma outra que lhe derrota e outra que lhe concede vitória. Quando a mesma ação é escolhida pelos jogadores, o resultado é um empate. A matriz de recompensa do jogo *pedra papel tesoura* pode ser vista na Figura *2.4.1*.

$$G = \begin{array}{c|ccc} & R & P & S \\ \hline R & 0 & -1 & 1 \\ P & 1 & 0 & -1 \\ S & -1 & 1 & 0 \end{array}$$

Figura 2.4.1: Matriz de recompensas do *pedra papel tesoura*. R indica o movimento pedra, P o movimento papel e S o movimento tesoura.

2.4.2 DILEMA DO PRISIONEIRO

No jogo *dilema do prisioneiro*, cada jogador possui duas ações: cooperar ou trair. Apesar de existir uma estratégia dominante para este jogo, a estratégia pura trair, esta não é necessariamente a melhor escolha quando utilizada em jogos de repetição contra certos oponentes. Este jogo já fora estudado extensivamente para compreender como uma estratégia de cooperação pode surgir e se manter (AXELROD, 2006). O nome do jogo faz referência a uma situação em que dois criminosos são presos porém não existem provas contra todos os crimes que eles cometeram. Desta forma, a polícia coloca cada um deles em uma sala separada e é dado a eles as opções de delatar (Trair) ou se manter em silêncio (Cooperar) com seu comparsa. Caso um deles delate e o outro não, a pena do delator é reduzida e a do comparsa é aumentada. Caso ambos delatem, a pena de ambos será

maior do que a atual. Caso nenhum deles delate então a pena de ambos não é alterada. A matriz de recompensa do jogo *Dilema do Prisioneiro* pode ser vista na Figura *2.4.2*.

$$G = \begin{array}{|c|c|c|} \hline & T & C \\ \hline T & -1/-1 & 1/-2 \\ \hline C & -2/1 & 0/0 \\ \hline \end{array}$$

Figura 2.4.2: Matriz de recompenças do *dilema do prisioneiro*.

2.5 ESTRATÉGIAS

Uma estratégia pode ser definida como um conjunto de tuplas contendo um estado de jogo junto a um ação, ou seja, ela indica qual será a ação que um jogador realizará em cada estado do jogo (MERTENS, 1986).

Existem dois tipos de estratégias: puras e mistas. Uma estratégia é dita pura quando, para um estado do jogo, o jogador sempre utilizará a mesma ação. Uma estratégia é chamada de mista quando, para cada estado do jogo, o jogador tem uma distribuição de probabilidade sobre as ações que podem ser utilizadas (NEUMANN; MORGENSTERN, 1944). Em particular, no jogo *dilema do prisioneiro* um exemplo de estratégia pura seria sempre utilizar a ação T, enquanto uma estratégia mista seria sempre ter 30% de probabilidade de utilizar a ação T e 70% de probabilidade de utilizar a ação C.

2.5.1 ESTRATEGIA DOMINANTE

Uma estratégia é dita dominante quando a sua escolha sempre resultará em um lucro maior que todas as outras (NEUMANN; MORGENSTERN, 1944). Se uma estratégia dominante existir em um jogo, então utilizar a esta estratégia é o modo ótimo de se jogar. Caso uma estratégia A sempre resulte em resultados superiores a de uma estratégia B, então diz-se que a estratégia A domina a estratégia B e que a estratégia B é dominada pela estratégia A. Considere, por exemplo, o jogo *dilema do prisioneiro*, escolher a ação T é uma estratégia dominante. Independentemente se o oponente jogar com T ou com C, tem-se que se o jogador escolher T isto sempre resultará em um lucro maior do que C, logo pode-se considerar que a estratégia pura T domina a estratégia pura C.

2.5.2 ESTRATÉGIA ÓTIMA

Quando uma estratégia dominante existe, ela pode ser encontrada através da matriz do jogo. Assim, nestes casos, encontrar uma estratégia dominante e utilizar a mesma é um modo de jogar de forma ótima. Porém nem todos os jogos têm uma estratégia dominante. Um exemplo seria o jogo *pedra papel tesoura*, em que toda ação ganha de algum e perde de outro. Para casos como este, existem outros métodos de escolha de uma estratégia ótima. Neste caso é necessário definir uma estratégia que consiga maximizar o lucro do pior caso (TUROCY; STENGEL, 2002), ou seja, maximizar o lucro quando o oponente também está jogando do melhor modo possível. Alguns métodos para determinar a estratégia ótima nestes casos serão descritos detalhadamente no próximo capítulo.

Para o caso de jogos de repetição, dependendo do modo como os oponentes jogam, é possível obter um lucro maior do que o resultante da estratégia ótima, isto pode ser obtido caso o oponente seja previsível. Neste trabalho será apresentado um algoritmo que é capaz de obter resultados melhores que o estado da arte atual para jogos de repetição e também possuir um custo de memória mais baixo.

3 ESTADO DA ARTE E TRABALHOS RELACIONADOS

A teoria dos jogos tem suas origens com a publicação do livro *Theory of Games and Economic Behaviour* escrito por John von Neumman e Oskar Morgenstern (NEUMANN; MORGENSTERN, 1944), tendo como principal aplicação o estudo de relações econômicas. Inicialmente foram estudados jogos de dois jogadores de soma zero, nos quais se um jogador recebe lucro então o outro jogador recebe um prejuízo de mesmo valor. Em cada jogo o objetivo de cada jogador é maximizar a sua recompensa final. Normalmente se tem um jogador querendo maximizar o resultado final do jogo enquanto o outro querendo minimizar.

Os jogos podem ser representados na forma extensiva no formato de uma árvore na qual cada nó representa um estado do jogo em que se deve realizar um ação. As jogadas são realizadas de forma alternada. A cada ação realizada o valor da recompensa de cada jogador é atualizado, sendo que quando se chega a um nó terminal da árvore o jogo está terminado e cada jogador tem como resultado final a sua recompensa acumulada após a última ação ser realizada.

3.1 ESTRATÉGIA MINIMAX E EQUILÍBRIO NASH

Um jogo é considerado resolvido quando se sabe quais são as ações que um jogador deve realizar para se obter a sua recompensa ótima considerando que seu oponente também está jogando de maneira ótima. A estratégia ótima de um jogo de soma zero de dois jogadores pode ser dada pelo *Teorema Minimax* (NEUMANN; MORGENSTERN, 1944). Nele é considerado que o jogador tem como objetivo maximizar o resultado do pior caso possível, ou seja, jogando contra o oponente mais forte que possa existir.

Considerando a representação em forma de árvore, para calcular quais são as ações que devem ser tomadas segundo a estratégia minimax considera-se a alternância entre os dois jogadores. Um com objetivo de maximizar o seu resultado final e o outro com objetivo de minimizar. Após a construção da árvore para um determinado nível de profundidade, ao se atingir o nível dos estado terminais, realiza-se uma avaliação estática dos mesmos estados

considerando uma função de avaliação que maximiza a pontuação de um jogador (max) e minimiza a pontuação do outro (min). Se o estado representa uma vitória do jogador max seu valor é definido como $+\infty$, de forma contrária o valor é definido como $-\infty$. Após a avaliação dos estados terminais, deve-se subir um nível na árvore e ver qual será a escolha realizada pelo jogador acima considerando a possibilidade do mesmo ser do tipo max ou min. Caso seja do tipo max seu valor será igual o valor máximo dos filhos e do tipo min igual ao valor mínimo dos filhos. Este processo é realizado recursivamente até atingir-se o nível do estado raiz. Caso o jogo seja computável, ou seja, os estado terminais representem fim de jogo, obtém-se uma estratégia vencedora. Caso o jogo não seja computável, o valor do estado raiz proporcionará a escolha da ação associada à estratégia minimax.

Caso o jogo seja com ações simultâneas, normalmente representado na forma normal com uma matriz, sendo a escolha de um jogador associado às colunas e do outro associado às linhas, a estratégia minimax do jogador das linhas pode ser encontrada utilizando programação linear (ZIONTS, 1974) para se resolver o problema de maximização formulado a seguir:

$$\begin{aligned} &Maximizar\ u \\ &Sujeito\,a:\ G'x - 1u \geq 0 \\ &\qquad\qquad 1'x = 1 \\ &\qquad\qquad x \geq 0 \\ &u\ irrestrito\ em\ sinal, \end{aligned}$$

Sendo G' a matriz de recompensas em sua forma transposta $n \times n'$ e x um vetor de n' variáveis associadas à probabilidade de cada ação ser utilizada, representando uma política mista.

Do ponto de vista do jogador min a estratégia $maxmin$ deste jogador associado as colunas pode se encontrada utilizando o equivalente dual da formulação 3.1:

$$\begin{aligned} &Minimizar\ v \\ &Sujeito\,a:\ Gx - 1v \leq 0 \\ &\qquad\qquad 1'x = 1 \\ &\qquad\qquad x \geq 0 \\ &v\ irrestrito\ em\ sinal, \end{aligned}$$

Sendo G a matriz de recompensas $n' \times n$ e x um vetor de n variaveis associadas a probabilidade de escolha de cada ação.

Este conceito da estratégia minimax foi estendido para jogos de soma não zero por John Forbes Nash Jr. com seu Ponto de Equilíbrio (JR., 1951). O ponto de equilíbrio pode ser definido como um estado do jogo em que se apenas um jogador mudar a sua estratégia unilateralmente então ele obterá uma recompensa menor ou igual a que ele estava recebendo com sua estratégia antiga. O ponto de equilíbrio é considerado uma extensão do teorema minimax pois, para jogos de soma zero, o ponto de equilíbrio sempre coincide com a estratégia minimax. Porém para jogos de soma não zero eles podem se diferenciar. Caso apenas estratégias puras sejam permitidas então não se pode afirmar que existe um ponto de equilíbrio em um jogo, porém caso possa ser utilizado estratégias mistas então é provado que existe ao menos um ponto de equilíbrio. Porém cada ponto de equilíbrio em um mesmo jogo não precisa ter o mesmo valor que os outros, de modo que alguns pontos podem ser mais vantajosos.

Enquanto encontrar a estratégia minimax é um problema bem equacionado, encontrar um ponto de equilíbrio é mais difícil. Até hoje não foi formulado um algoritmo polinomial que consegue encontrar ao menos um ponto de equilíbrio em qualquer jogo, sendo este problema classificando como pertencente a uma subclasse de *NP* chamada de *PPAD (Polynomial Parity Argument for Directed Graphs)* (DASKALAKIS PAUL W. GOLDBERG, 2009) sendo este problema classificado como *PPAD-completo*.

Apesar da estratégia minimax e o ponto de equilíbrio Nash serem soluções para os jogos abordados, eles nem sempre podem retornar uma estratégia considerada ótima contra qualquer oponente.

No contexto de jogos de repetição, o mesmo jogo é jogado repetidas vezes pelos mesmos jogadores sendo que o objetivo final é maximizar o resultado médio de todos os *rounds* (MERTENS, 1986). Neste caso, podem existir estratégias que não são ótimas mas que gerem resultados finais melhores. Um exemplo disto pode ser visto no jogo de repetição *pedra papel tesoura*, representado na Figura *3.1.1*.

Neste jogo, ao calcular-se a estratégia minimax tem-se, para qualquer ação inicial escolhida, o mesmo resultado para o estado raiz. Deste modo é possível ver que, segundo a estratégia minimax não há diferença entre a escolha dos ações. Caso este jogo seja analisado segundo o equilíbrio Nash, a estratégia ótima seria jogar com a mesma probabilidade para cada ação, resultando em um valor esperado de *0*. Como neste caso o jogo é simétrico e de soma zero, este valor esperado também é independente da estratégia escolhida pelo

$$G = \begin{array}{c|ccc} & R & P & S \\ \hline R & 0 & -1 & 1 \\ P & 1 & 0 & -1 \\ S & -1 & 1 & 0 \end{array}$$

Figura 3.1.1: Matriz de recompensas do *pedra papel tesoura*.

oponente.

Entretanto em jogos de repetição, podem existir situações nas quais estas estratégias ótimas fracassam. Considere como exemplo um oponente que decida jogar com uma estratégia bem simples, de sempre repetir a mesma ação *pedra*. No caso da estratégia minimax todas as três ações tem a mesmo prejuízo no pior caso, assim ela poderia sempre escolher a ação tesoura, já que não importa qual ação deve ser tomada pelo jogador, pois o valor de cada ação é igual independentemente da escolha do oponente. Sendo assim, escolher usar a ação tesoura todos os *rounds* é uma possibilidade, mesmo que resulte em perder todos os *rounds*. Assim, é possível ver que a estratégia minimax não é suficiente para encontrar a estratégia capaz de obter o maior número de vitórias contra um oponente simples como este.

O mesmo pode ser observado no caso do equilíbrio de Nash, que escolhe a estratégia mista em que todas as ações possuem a mesma probabilidade. Esta estratégia será fixa, independentemente se o oponente é altamente previsível jogando sempre a mesma ação ou se ele segue uma estratégia mais complexa. Assim, é possível perceber que tanto o equilíbrio Nash quanto a estratégia minimax, apesar de serem ótimas no caso de um jogo de apenas um *round*, não necessariamente são a melhor opção se forem consideradas em um jogo de repetição e que oponentes podem ter comportamento dinâmico. Para se encontrar soluções mais eficientes para este tipo de situação são utilizadas estratégias adaptativas, ou seja, que procuram analisar o histórico de ações do jogo no sentido de encontrar algum padrão de jogo relacionado às ações do oponente.

3.2 ESTRATÉGIAS ADAPTATIVAS

Tanto o equilíbrio Nash quanto a estratégia minimax são consideradas estratégias ótimas por maximizarem o resultado do pior caso. Porém, caso o oponente não esteja jogando de modo ótimo, podem existir outras estratégias que sejam mais eficientes. Em jogos de repetição é possível descobrir qual é a estratégia do oponente dado uma quantidade suficiente de *rounds*. Por isto foram criadas estratégias que procuram se adaptar ao modo de jogar do oponente e desta forma se aproximarem do resultado ótimo.

3.2.1 FICTITIOUS PLAY

Uma das estratégias adaptativas mais antigas e simples que existem é chamada de *Fictitious Play* (BROWN, 1951), que foi originalmente criada para ser um método iterativo de determinar o ponto de equilíbrio de um jogo. A estratégia fora inicialmente proposta para jogos em que seus jogadores realizam suas alterações alternadamente, porém foi estendida posteriormente para jogos em que ambos os jogadores se atualizam simultaneamente (BERGER, 2007). A estratégia *Fictitious Play* funciona armazenando para cada ação uma probabilidade que indica o quão provável é do oponente utilizar cada ação no próximo *round*. Estas probabilidades são calculadas utilizando o histórico de jogadas realizadas pelo oponente até o *round* atual. Após o cálculo das probabilidades, o *Fictitious Play* joga segundo a estratégia mista que obtém o melhor resultado contra a estratégia formada pelas probabilidades que ele calculou, que representa o oponente. Sendo assim, caso um oponente esteja utilizando uma estratégia mista qualquer, as probabilidades atribuídas pelo *Fictitious Play* convergem para as mesmas probabilidades da estratégia mista utilizada pelo oponente com o passar dos rounds. No caso do exemplo em que o oponente sempre repete a mesma ação pedra o *Fictitious Play* seria capaz de aprender isto logo no segundo *round* e conseguir ganhar do oponente em todos os *rounds* futuros. Sendo assim, independentemente de qual ação o oponente escolha repetir para sempre, o *Fictitious Play* consegue perceber esta repetição e utilizá-la para ganhar os *rounds* futuros, algo que a estratégia minimax ou o ponto de equilíbrio não seriam capazes de fazer.

Outro jogo que é comumente usado em jogos de repetição é o *dilema do prisioneiro*, em que jogadores têm a opção de trair ou cooperar com seu oponente. Se ambos cooperarem, tem-se um desfecho melhor para ambos do que se eles se traírem mutualmente. Porém se

um jogador escolher trair enquanto o outro escolher cooperar então este é o melhor caso possível para quem traiu e o pior possível para quem decidiu cooperar.

Independentemente de qual ação o oponente escolher a estratégia minimax sempre escolherá trair pois esta é uma estratégia dominante neste jogo. O único ponto de equilíbrio existente no *dilema do prisioneiro* é o caso em que ambos os jogadores se traem. Se o algoritmo *Fictitious Play* for utilizado para jogar este jogo, ele também sempre resultará em trair o outro jogador já que esta é uma estratégia dominante. Esta é a estratégia ótima que pode ser utilizada neste jogo com apenas um *round*, porém em um jogo de repetição ela pode não ser necessariamente a melhor estratégia disponível. Um exemplo disto seria contra um oponente que inicia o jogo cooperando, porém, após uma traição, decide trair seu oponente nos demais *round*. Neste sentido todas as estratégias apresentadas até agora resultariam em traição durante todos os *rounds* do jogo, muito embora um resultado melhor poderia ser obtido utilizando a estratégia cooperar durante todos os *rounds*.

3.2.2 TIT FOR TAT

O jogo do *dilema do prisioneiro* já foi alvo de estudo na teoria de jogos, sendo um dos trabalhos mais influentes o livro (AXELROD, 2006), no qual é apresentado uma estratégia bem simples chamada de *tit for tat*. Esta estratégia fora desenvolvida para um torneio de estratégias para o jogo de repetição *dilema do prisioneiro*, sendo esta a estratégia vencedora. Ela funciona cooperando com o oponente no primeiro *round* e após isto, para todos os outros *rounds* seguintes, ela segue o padrão de cooperar se o oponente cooperou no round anterior e de trair se o oponente trair no round anterior.

Tit for tat é um exemplo de estratégia que pode obter resultados melhores do que o ponto de equilíbrio e a estratégia minimax. Por exemplo, ao jogar contra um oponente que começa cooperando, mas que caso seja traído irá trair nos demias *rounds*, o *tit for tat* conseguirá um resultado maior do que o alcançado pela estratégia minimax e o ponto de equilíbrio, pois estas duas sempre trairão o oponente. Por outro lado o *tit for tat* sempre cooperará com ele resultando em uma cooperação mútua. O bom desempenho da estratégia *tit for tat* também foi reforçado por estudos teóricos (LITTMAN; STONE, 2003). Devido a sua natureza retalhadora a estratégia *tit for tat* limita o adversário a nunca conseguir lucrar mais que ela no *dilema do prisioneiro*, com exceção do último

round. Assim o oponente fica limitado a obter no melhor caso o lucro de sempre cooperar reciprocamente e no pior caso o lucro de sempre trair reciprocamente.

Apesar do *tit for tat* ser considerado uma estratégia excelente para jogo do *dilema do prisioneiro*, ela não é considerada uma estratégia boa para outros tipos de jogos. Por exemplo, *pedra papel tesoura*, repetir o ação usada pelo oponente no *round* passado não necessariamente resulta no melhor resultado. O mesmo pode ser dito em relação a estratégia *fictitious play* que apesar de funcionar bem contra certos oponentes mais simples no jogo de *pedra papel tesoura*, ela acaba sempre escolhendo trair no jogo *dilema do prisioneiro*. Como se pode ver, apesar de gerar bons resultados nos jogos em que foram criadas, estas estratégias não geram resultados satisfatórios em outros jogos.

Para que um algoritmo gere bons resultados em diferentes jogos é importante que ele seja capaz de se adaptar à estratégia do seu oponente. Desta forma, estes algoritmos também devem ter como objetivo derrotar outros oponentes que estejam tentando se adaptar durante o jogo.

3.3 ESTRATÉGIA BASEADA EM SUBIDA DO GRADIENTE

Um algoritmo criado para jogar contra oponentes inteligentes é o *WoLF* (BOWLING; VELOSO, 2002) sendo essa uma sigla para *Win or Learn Fast* que é a filosofia central deste algoritmo. Caso não esteja conseguindo vencer, ele deve tentar aprender rapidamente como o oponente joga para voltar a ganhar. Ele começa a jogar utilizando a estratégia que está no ponto de equilíbrio Nash e segue com um método de subida do gradiente para computar a estratégia do oponente e fazer modificações do seu modelo ao longo dos *rounds*. Para tanto ele utiliza uma extensão da técnica de aprendizado por reforço *Q-learning*, que é garantida de convergir para a política ótima caso o oponente esteja jogando segundo uma estratégia estacionária. Concomitantemente ele utiliza o princípio *WoLF* para variar a taxa de aprendizado de modo que caso ele esteja perdendo o valor dela deva ser aumentado porém caso ele esteja ganhando o valor dela deva ser rebaixado. Apesar deste princípio garantir a convergência para a melhor estratégia possível contra certos grupos de oponentes ela se mostrou bem lenta, sendo em alguns casos necessário esperar centenas de milhares de *rounds*, podendo chegar até mesmo a um milhão de *rounds*. Sendo assim, ele não é uma boa escolha para ambientes nos quais o oponente altera sua estratégia após poucos rounds.

3.4 ESTRATÉGIA BASEADA EM ANÁLISE DE ENTROPIA

Uma estratégia criada para ser usada neste contexto de jogos de repetição em que o oponente está mudando a sua estratégia frequentemente e possivelmente se adaptando ao jogador é o algoritmo *ELPH* (JENSEN et al., 2005b,a) que significa *Entrophy Learning Prunning Hypothesys*. Este algoritmo utiliza o conceito de memória curta, sendo esta uma sequência de s ações consecutivas. Com base nesta sequência são utilizadas todas as subsequências de ações consecutivas de tamanho menor ou igual a s pertencentes à memória curta. Neste caso a quantidade total de subsequências existentes é dado por 2^s. Todas as subsequências que apareceram até o presente momento são armazenadas pelo algoritmo, ou seja, após um *round* ser realizado ele pega a memória curta atual e separa todas as suas subsequências possíveis procurando armazená-las em sua memória longa, que contém todas as subsequências diferentes que apareceram até então no jogo. A memória longa, além de armazenar as subsequências, também armazena quais ações apareceram junto com elas, ou seja, se o oponente utilizou a ação R no jogo *pedra papel tesoura* então todas as subsequências que estavam na memória curta nesta ação são atualizadas na memória longa para indicar a quantidade de vezes que o oponente usou a ação R até agora. Sendo assim, a memória longa armazena não somente quais subsequências apareceram mas também quais ações apareceram sucedendo elas e a quantidade de vezes em que eles apareceram.

O Algoritmo *ELPH* usa essa informação armazenada em sua memória longa para poder prever a próxima ação do oponente. Isto é feito calculando a entropia de cada subsequência de memória diferente armazenada na memória longa. A entropia é uma medida de previsibilidade utilizada na Teoria da Informação (SHANNON; WEAVER, 1963), no sentido de que quanto mais baixo for a entropia então mais fácil será a previsão de um evento, enquanto que quanto maior for a entropia mais imprevisível será a ocorrência do mesmo. A entropia é calculada pela seguinte fórmula:

$$-\sum_{a \in |A|} P_a \times log_2(P_a), \tag{3.1}$$

Sendo $|A|$ a quantidade de eventos e P_a a probabilidade do evento a acontecer.

A entropia é calculada para cada subsequência da memória longa levando em consideração as ações que as sucederam no sentido de se descobrir qual é a subsequência

que possui o menor grau de entropia e consequentemente qual a ação associada que deve ser escolhida pelo oponente. Este cálculo de entropia é realizado para todas as 2^s subsequências atuais. Devido a este crescimento exponencial de subsequências a cada *round* o *ELPH* realiza uma poda, ou seja, as subsequências em que a entropia fica muito alta comparado a um valor pré-determinado são removidas da memória longa devido ao fato de serem pouco relevantes. Das subsequências que sobrevivem a esta poda, é escolhida a subsequência com menor entropia como sendo a que melhor representa a próxima ação do oponente.

O algoritmo *ELPH* apresentou bons resultados contra oponentes que modificam suas estratégias rapidamente após poucos *rounds*, ele foi testado contra oponentes determinísticos e estocásticos em que a cada vinte *rounds* mudavam sua estratégia de jogo, mesmo assim o *ELPH* conseguiu derrotá-los no jogo de *pedra papel tesoura* sem precisar de muitos *rounds*. Com menos de 300 *rounds* ele já se mostrava superior. O *ELPH* também fora testado contra oponentes humanos no jogo de *pedra papel tesoura* demonstrando ser capaz de derrotar oponentes humanos, apesar de este resultado não ser muito forte pois faltam informações mais completas sobre o comportamento humano ao jogar *pedra papel tesoura* e também sobre as pessoas que competiram contra o *ELPH*. Mesmo assim, isto se mostra um resultado promissor para o *ELPH*.

Apesar de todas as vantagens apresentadas pelo *ELPH* em relação aos algoritmos anteriores ele não é isento de pontos negativos, principalmente em relação a utilização de memória. A quantidade de subsequências criadas pelo *ELPH* a cada *round* cresce de modo exponencial em relação ao tamanho da memória curta, de modo que o seu crescimento assintótico em relação ao tempo de execução deve ser proporcional ao crescimento das subsequências. Além disto, a quantidade de subsequências que são adicionadas na memória longa pode se tornar muito numerosa caso o processo de poda não seja eficiente, podendo no pior caso atingir o valor máximo de $(|A|+1)^s$, sendo A o conjunto de possíveis ações. Como o *ELPH* consegue prever somente qual é a ação com maior probabilidade de ser utilizada pelo oponente o mesmo não considera em sua decisão a probabilidade das outras ações. Consequentemente ele pode acabar obtendo um resultado menor que o ótimo quando for jogar contra um oponente que seja, por exemplo, estocástico. Por fim, outro problema apresentado pelo algoritmo é que quando ele joga *dilema do prisioneiro* ele sempre escolherá trair o oponente, mesmo que ele consiga aprender corretamente a

estratégia que o oponente utiliza, devido ao fato de trair ser uma estratégia dominante.

3.5 ESTRATÉGIA BASEADA EM *KERNEL* DE SIMILARIDADE

Na próxima seção será apresentado um novo algoritmo que é capaz de obter bons resultados em jogos de repetição contra oponentes inteligentes que conseguem se adaptar rapidamente ao longo do jogo. Este algoritmo consegue resolver alguns dos problemas do *ELPH* e das outras estratégias apresentadas, como, por exemplo, conseguir cooperar no dilema do prisioneiro e se mostrar competitivo no jogo de *pedra papel tesoura*. Também apresenta um custo de memória inferior ao requerido pelo algoritmo *ELPH*. Por fim, o algoritmo é capaz de calcular quais são as probabilidades de todas as ações imediatas e também de calcular uma estimativa das probabilidades das ações em *rounds* futuros possibilitando a utilização de uma estratégia de *lookahead*.

4 ESTIMATIVA DE DENSIDADE BASEADO EM UM *KERNEL* DE SIMILARIDADE (EDKS)

Como mostrado anteriormente, apesar de existirem algoritmos para jogos de repetição eles apresentam certos problemas para competir contra oponentes inteligentes ou quando foram confrontados em certos jogos onde não foram preparados para jogar bem. Um problema também observado de forma frequente é referente à memória e a quantidade de *rounds* necessários para se adaptar ao oponente. Nesta seção será apresentado o algoritmo *EDKS*, criado para jogar jogos de repetição contra oponentes independentemente se eles são estacionários ou inteligentes.

O algoritmo tem como ideia base utilizar um *kernel* para varrer o histórico de ações utilizadas pelo oponente à procura de padrões que possam ajudar a descobrir quais são as probabilidades de cada ação que poderá ser utilizada pelo oponente no próximo *round*. Estas probabilidades são posteriormente utilizadas para descobrir qual é a ação que tem o melhor valor esperado contra esta distribuição de probabilidades que modelam o oponente, sendo esta a ação que deverá ser utilizada pelo jogador no próximo *round*.

4.1 PREDIÇÃO DE AÇÕES

A memória do algoritmo é dividida em duas, uma chamada de memória curta que indica o estado atual do oponente e outra chamada de memória longa que armazena todos os estados anteriores observados juntamente com a ação do oponente que os sucedeu. A memória longa, denotada como M, armazena sequencialmente todas as sequências de tamanho s formadas pelas ações consecutivas realizadas pelo oponente até o *round* atual enquanto que a memória curta, denotada como m, armazena apenas a sequência de s ações mais recentes. A memória longa além de armazenar as ações consecutivas também armazena quais foram as ações a que a sucederam juntamente com a frequência com que elas apareceram $f(a)$. Após um *round* ser realizado, a memória curta atual junto com a ação que o oponente utilizou são adicionadas na memória longa. Caso já exista uma sequência nela que seja igual à da memória curta, então a frequência da ação utilizada pelo oponente é incrementada. Após isto ser realizado, a memória curta é atualizada

substituindo-se a ação mais antiga pela ação mais recente que o oponente utilizou.

Essas tuplas que contém as informações do oponente são utilizadas pelo *kernel* no sentido de poder prever qual será a ação utilizada pelo oponente no próximo *round*. Na Seção *4.2* será definido a forma do *kernel* que define uma medida de similaridade entre duas sequências m e m' de tamanho s como $\langle m, m' \rangle$. Consequentemente, o cômputo da distância entre duas sequencias m e m' pode ser definido como:

$$\|m - m'\|^2 := \|m\|^2 - 2\langle m, m' \rangle + \|m'\|^2 \tag{4.1}$$

O resultado do *kernel* é normalizado de modo que $||m||^2$ seja igual a 1 para cada m de tamanho s. Os valores de distância computados pelo *kernel* são utilizados junto das frequências $f(a)$ para encontrar a probabilidade de cada ação ser utilizada. Optou-se também pela utilização de um decaimento exponencial definindo para o cômputo das similaridades na forma:

$$e^{-\frac{\gamma}{2}\|m-m'\|^2} = e^{-\gamma(1-\langle m,m' \rangle)}, \tag{4.2}$$

onde γ é um parâmetro que regula o decaimento. Assim, a probabilidade de uma certa ação a ser utilizada em função de uma dada memória curta m é dada pelo somatório do produto da frequência de cada sequência $m' \in M(a)$ pela sua similaridade com m. Este valor é normalizado dividindo-se o mesmo pelo somatório das similaridades.

$$P_a(m) = \frac{\sum_{m' \in M(a)} f_a(m') e^{-\frac{\gamma}{2}\|m-m'\|^2}}{\sum_{m' \in M} e^{-\frac{\gamma}{2}\|m-m'\|^2}}. \tag{4.3}$$

Esta probabilidade é calculada para cada ação $a \in A$ sendo utilizada para escolher a ação a^* que será praticada pelo jogador no próximo *round*. Inicialmente, poderia ser escolhida a ação que obteve a maior probabilidade individual. Entretanto, para obter um melhor resultado optou-se pela escolha da ação associada ao maior valor esperado de recompensa. O cálculo do valor esperado leva em consideração a matriz de recompensa do jogo G. Sendo assim, a escolha da melhor ação é definida segundo a equação (*4.4*).

$$a^* = \arg\max_{a \in A} \sum_{b \in A} P_b(m) G(a, b) \tag{4.4}$$

Para se justificar este tipo de critério será utilizado como exemplo uma situação do

jogo *pedra papel tesouras*. Suponha que as seguintes probabilidades foram calculadas para modelar como o oponente agirá no próximo *round*: $P(r) = 0.4$, $P(p) = 0.25$, $P(s) = 0.35$, onde $P(r)$, $P(p)$ e $P(s)$ correspondem às probabilidades de *pedra*, *papel* e *tesoura*, respectivamente. Inicialmente pode-se pensar que utilizar *papel* pode ser a melhor escolha já que é a ação que derrota *pedra* sendo esta a de maior probabilidade. Porém o valor esperado de utilizar *papel* é dado por:

$$0.4 \times 1 + 0.25 \times 0 + 0.35 \times -1 = 0.05. \tag{4.5}$$

Entretanto, se for escolhido utilizar a ação *pedra* será obtido um valor esperado maior:

$$0.4 \times 0 + 0.25 \times -1 + 0.35 \times 1 = 0.1. \tag{4.6}$$

Deste modo, a ação a ser escolhida não deve ser a que derrota a ação mais provável do oponente mas sim a que possui o maior valor esperado.

4.2 KERNEL DE SIMILARIDADE

O *kernel* tem como objetivo computar a similaridade da sequência que se tem na memória curta com as sequências já armazenadas na memória longa. Como a quantidade de sequências diferentes pode ser muito grande é pouco provável que no *round* atual a sequência que tem-se na memória curta já tenha aparecido na memória longa, sendo assim o *kernel* funciona utilizando todas as sequências armazenadas na memória longa, ponderando cada uma delas de acordo com sua similaridade com a sequência da memória curta. Deste modo, o *kernel* compara a memória curta atual com cada sequência da memória longa para encontrar as suas similaridades e utilizar as frequências de ações que sucederam cada sequência para encontrar a probabilidade de cada ação ser usada pelo oponente. O modo como a similaridade entre duas sequências é calculada será explicado a seguir. Dado duas sequências m e m' de tamanho s, o *kernel* encontra a sua similaridade $\langle m, m' \rangle$ efetuando um conjunto de comparações. Cada comparação é realizada utilizando-se uma subsequência de tamanho $n \leq s$ formada por uma sequência de ações consecutivas que começam e terminam nas mesmas posições relativas. Por exemplo, a subsequência das três primeiras ações de m só é comparada com a subsequência das três primeiras ações de m'. A subsequência de m que começa na segunda posição e termina na quarta posição será

comparada com a subsequência de m' que também começa na segunda posição e termina na quarta posição. Ao comparar duas subsequências para saber o quão parecidas elas são procura-se pelas subsequências que sejam idênticas. Na Figura *4.2.1* pode-se ver um exemplo de comparações sendo realizadas entre duas sequências de tamanho $s = 3$ sendo elas $m = \{RPS\}$ e $m' = \{RPR\}$. Neste exemplo pode-se observar que a quantidade de pares de subsequências idênticas é igual a três, sendo elas os pares $\{R\}\{R\}$, $\{P\}\{P\}$ e $\{RP\}\{RP\}$, enquanto que a quantidade de subsequências diferentes é igual a 3 sendo elas os pares $\{S\}\{R\}$, $\{PS\}\{PR\}$ e $\{RPS\}\{RPR\}$.

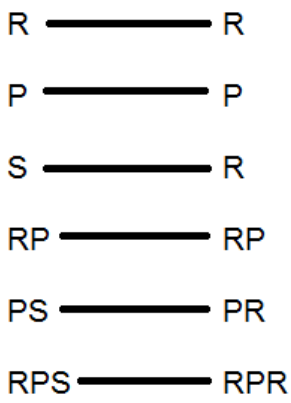

Figura 4.2.1: Pares de subsequências.

A quantidade total de subsequências é dada pela equação $s(s+1)/2$. Pode-se parecer então que o tempo necessário para comparar duas sequências seja quadrático, porém isto pode ser realizado em tempo linear. Tal procedimento é executado mantendo qual foi a última posição da sequência m que estava diferente do elemento da mesma posição relativa na sequência m'. As comparações devem começar da última posição das sequências caminhando em direção à primeira posição das sequências, sempre comparando elementos que estejam na mesma posição relativa, por exemplo, o último elemento de m deve ser comparado com o último elemento de m', o terceiro elemento de m deve ser comparado apenas com o terceiro elemento de m'. O algoritmo proposto segue as seguintes regras: caso os elementos das duas listas sejam diferentes então o valor da última posição diferente é atualizado para o valor da posição atual; caso os elementos forem iguais então o contador de similaridades é incrementado pela diferença entre o valor posição atual e o valor da última posição diferente. Em seguida, a posição atual é decrementada em uma unidade a fim de refletir a posição subsequente nas duas sequências.

Ao aplicar este método na comparação das sequências RPS e RPR observa-se que a comparação inicia pela última posição, com a última posição diferente tendo valor inicial igual a 4. Em seguida é comparado o par de elementos da terceira posição sendo eles S e R. Como elas são ações diferentes, pode-se inferir que todos os pares de subsequências que comecem na terceira posição serão formados por subsequências diferentes. Então, o contador da última posição diferente é atualizado para o valor 3. Após isto, prossegue-se para a comparação da segunda posição das sequências onde será comparado P com P. Como eles são iguais então isto quer dizer que todas as subsequências que comecem na segunda posição e terminem antes da terceira, onde existe um par de elementos diferentes, são iguais. Neste caso, existe apenas uma subsequência que satisfaz esta condição, sendo a subsequência que começa e termina na segunda posição, assim o contador de similaridades é incrementado em 1. A seguir será comparado os pares de elementos da primeira posição, sendo eles R e R, como eles são iguais pode-se dizer que todas as subsequências que começam na primeira posição e tem seu fim antes da terceira posição são similares, existem assim duas subsequências que satisfazem esta restrição sendo elas a subsequência que começa e termina na primeira posição e a subsequência que começa na primeira posição e termina na segunda, então o contador de similaridades é incrementado em 2, tendo o valor final 3. Após encontrar a quantidade total de subsequências similares, normaliza-se este valor dividindo o mesmo pela quantidade total de subsequências existentes, que é dado pela equação $s(s+1)/2$. No exemplo anterior o valor final da normalização será de $1/2$. Este método para o cálculo linear da similaridade pelo *kernel* é dado pelo algoritmo *1*.

Este método de cálculo de similaridades em tempo linear também oferece uma outra vantagem em termos de tempo ao ser utilizada uma estrutura de dados correta para armazenar as sequências de memória. Caso nós tenhamos duas sequências armazenadas na memória longa que somente na primeira posição sejam diferentes entre si, por exemplo RPS e SPS pelo método linear é possível ver que ao comparar elas com uma sequência qualquer que esteja na memória curta elas terão o mesmo resultado ao comparar a terceira e a segunda posições, somente na primeira posição que poderá haver uma divergência. Assim, é possível economizar tempo se compararmos as partes destas subsequências que forem iguais apenas uma vez. Isso pode ser feito facilmente se as sequências da memória longa forem armazenadas utilizando uma estrutura de dados *R-Way Trie* (KNUTH,

Algoritmo 1: Algoritmo do *kernel* de Similaridade.

Input: Par de sequências m e m'.
Output: Número de subsequências similares entre m and m'.
similaridades $\leftarrow 0$;
ultimaPosicaoDiferente $\leftarrow s + 1$;
posicaoAtual $\leftarrow s$;
while *posicaoAtual* $\neq 0$ **do**
  if *m[posicaoAtual]=m'[posicaoAtual]* **then**
    similaridades $\leftarrow$ similaridades+ultimaPosicaoDiferente-posicaoAtual;
  else
    ultimaPosicaoDiferente $\leftarrow$ posicaoAtual;
  end if
  posicaoAtual $\leftarrow$ posicaoAtual -1;
end while
return $\frac{\text{similaridades}}{\frac{s(s+1)}{2}}$;

1998) em que cada sequência da memória longa é armazenada com as últimas posições da sequência perto da raiz e a primeira posição de cada sequência sendo um nó terminar da árvore. A cada *round* quando for necessário comparar a sequência da memória curta com todas as sequências da memória longa a *R-Way Trie* será percorrida realizando um caminhamento em profundidade em que a cada nó é realizado uma comparação segundo o método linear armazenando a quantidade de subsequências iguais nele, quando se chega a um nó raiz então temos uma sequência comparada completamente. Deste modo ao passar o *kernel* em várias sequências diferentes a quantidade de comparações é minimizada, já que parte das comparações utilizadas em uma sequência podem ser aproveitadas para outra. Juntamente com isto, o espaço necessário para armazenar toda a memória longa é menor do que se fosse armazenar todas as sequências individualmente.

A figura 4.2.2 mostra uma árvore em que as sequências *RPR* e *RPS* foram adicionadas. Neste caso, apenas *4* nós são necessários para armazenar estas duas sequências. Para encontrar as probabilidades de cada movimento, são realizados apenas *4* cálculos pelo *kernel* ao invés de *6*.

4.2.1 COMPLEXIDADE TEÓRICA DE TEMPO E ESPAÇO

A seguir será discutido a complexidade tanto em termos de espaço quanto de tempo do algoritmo utilizando a estrutura de dados *R-Way Trie*. Primeiramente, será analisado a inserção de uma nova sequência. Como cada nó da árvore armazena um elemento

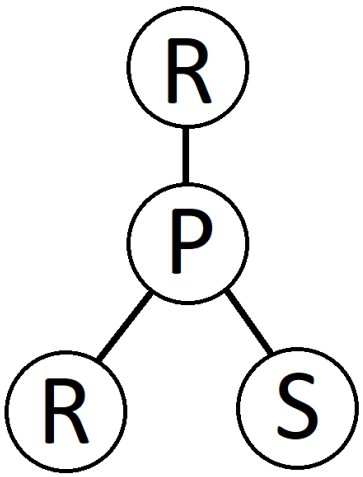

Figura 4.2.2: Memória longa representada por uma *R-Way Trie* contendo as sequências *RPR* e *RPR*.

da sequência, a inserção pode ser realizada em tempo linear em relação ao tamanho da sequência apenas caminhando pela árvore inserindo um elemento de cada vez. O custo de tempo para calcular o *kernel* em uma sequência também é linear em relação ao tamanho da sequência como demonstrado anteriormente. Como a cada *round* uma nova sequência diferente pode acabar sendo adicionada na memória longa o custo de calcular o *kernel* em todas as sequências para conseguir encontrar as probabilidades corretamente é dado no pior caso por $s \times r$, onde r é a quantidade de *rounds* que se passaram desde o início do jogo até agora. Para a análise da complexidade total é possível calcular um limite superior tanto para tempo quanto para espaço pois a quantidade máxima de sequências diferentes que podem ser adicionadas na memória longa é igual a $|A|^s$, dado que o tamanho de cada sequência é s e que a quantidade de ações diferentes que podem ser utilizadas pelo oponente é dado por $|A|$. Assim, pode-se inferir que o pior caso de tempo para calcular as probabilidades é igual a $s \times |A|^s$. Analisando o melhor caso possível em que esta jogando contra um oponente que sempre utiliza a mesma ação, temos que apenas uma sequência diferente irá aparecer. Assim, será necessário armazenar apenas uma sequência e consequentemente o tempo necessário para encontrar as probabilidades será igual a s.

4.3 MEMÓRIA DAS AÇÕES DO JOGADOR

Armazenar as ações realizadas pelo adversário para poder prever o futuro dele é uma boa técnica contra oponentes que funcionam seguindo um conjunto de regras pré-definidas como um autômato por exemplo. Porém ao jogar contra um oponente que seja adapta-

tivo, também tomando suas decisões baseadas no histórico de ações de seu adversário, observar somente as ações do oponente pode não ser a melhor solução. Neste tipo de caso é mais sensato também utilizar o histórico do jogador para prever a ação do oponente, entretanto, como não é possível saber se o oponente é inteligente ou não antes do jogo começar, corre-se o risco se armazenar apenas as ações do jogador e com isso ficar em desvantagem caso o oponente não seja adaptativo. Para evitar este tipo de problema são armazenadas duas memórias, uma primeira para as ações do oponente e uma segunda para armazenar as ações do jogador. O funcionamento do algoritmo continua o mesmo porém com a seguinte mudança no cálculo das probabilidades: primeiro elas são calculadas utilizando a primeira memória curta como se não tivéssemos a segunda; depois são calculadas novamente utilizando apenas a segunda memória curta como se a primeira não existisse. Assim, no final haverá duas distribuições de probabilidade modelando o oponente, elas então são somadas e, por fim, normalizadas para formar uma distribuição final que será usada para encontrar qual será a ação que deverá ser realizada no próximo *round*.

4.4 LOOKAHEAD

Quando se está jogando um jogo de repetição o objetivo real é maximizar o lucro médio obtido após o último *round* ser realizado, deste modo, jogar de modo guloso ou mesmo utilizar o melhor valor esperado considerando somente o próximo *round* pode não ser o melhor modo de jogar. Um exemplo de quando considerar apenas o lucro do próximo *round* pode acarretar em uma estratégia não ótima está no jogo *dilema do prisioneiro* tendo o algoritmo *tit for tat* como oponente. Considerando que antes do primeiro *round* já se tenha conhecimento de que o oponente está jogando segundo a estratégia *tit for tat* e que o algoritmo procura maximizar apenas o lucro do próximo *round* independentemente se o oponente irá trair ou cooperar a escolha ótima sempre será trair, pois é uma ação dominante. Portanto o algoritmo sempre trairá o oponente acarretando em um lucro médio de -1 enquanto que se fosse escolhido cooperar terminaria com lucro médio de 0. Considerando agora o caso em que o algoritmo que esta sendo utilizado procura maximizar o lucro médio dos próximos dois *rounds* e não apenas do primeiro. Como o oponente é a estratégia *tit for tat*, e considerando que se tem o conhecimento prévio disto, é sabido que no primeiro *round* ele irá cooperar, logo existem quatro possíveis escolhas de ações para os próximos *rounds* sendo elas cooperar e depois cooperar, cooperar e depois trair,

trair e depois cooperar, trair e depois trair. Se for escolhido cooperar no primeiro *round* será obtido um lucro imediato de 0 e no próximo *round* o oponente cooperará novamente, caso seja escolhido cooperar novamente, resultando outra vez em um lucro imediato de 0 e totalizando um lucro médio final de 0. Se for escolhido cooperar e depois trair será obtido 0 no primeiro *round* e depois 2 no segundo *round*, tendo assim um lucro médio final de 1. Se for escolhido trair no primeiro *round* será obtido um lucro de 2 nele porém saberemos que o oponente trairá no *round* seguinte. Caso seja escolhido trair o oponente novamente nós obteremos um lucro de -1 no segundo *round* totalizando uma média de 0.5. Caso seja escolhido trair no primeiro *round* será obtido lucro 2 e cooperar no segundo *round* será obtido lucro -2 nele totalizando um lucro médio de 0. Assim é possível ver claramente que a melhor escolha para maximizar o lucro nos próximos dois *rounds* é cooperar no próximo *round* para poder trair logo em seguida.

O algoritmo apresentado neste capítulo consegue descobrir as probabilidades de cada ação ser usada no próximo *round* e isto pode ser modificado facilmente para realizar *lookahead*. O valor esperado do próximo *round* deve ser calculado para todas as possibilidades de ações que tanto o jogador quanto o oponente podem utilizar. Para cada uma é criada uma nova memória curta em que se considera que ambos os jogadores utilizaram as ações e então ela é utilizada para calcular as probabilidades utilizando as memórias armazenadas na memória longa. Este processo é realizado de modo a encontrar qual é a ação que maximiza a soma do lucro deste e do próximo *round*. A equação (*4.7*) mostra o cálculo do *lookahead* de profundidade 1.

$$a^* = \arg\max_{a \in A} \sum_{b \in A} (P_b(m)G(a,b) + P_b(m) \max_{c \in A} \sum_{d \in A} P_d(m_{(a,b)})G(c,d)), \tag{4.7}$$

onde: $P_b(m)$ denota a probabilidade da ação b ser utilizada pelo oponente dado a memória curta m; $G(a,b)$ denota o lucro obtido pelo jogador ao utilizar a ação a e o oponente utilizar a ação b; c denota um ação que o jogador pode utilizar no próximo *round*; d denota uma ação que o oponente pode utilizar no próximo *round*; $G(a,b)$denota o lucro obtido pelo jogador ao utilizar a ação c e o oponente utilizar a ação d; $P_d(m_{(a,b)})$ denota a probabilidade do oponente utilizar a ação d dado que a memória curta atual equivale a memória curta do *round* anterior em que o jogador utilizou a ação a e o oponente a ação b.

Este mesmo processo pode ser realizado recursivamente para poder estender o *loo-*

kahead para um número maior de *rounds*.

O custo computacional do *lookahead* pode ser calculado tendo como base a quantidade de vezes em que o algoritmo é executado para cada possível futuro que é dado por $|A|^L$ sendo L o tamanho do *lookahead*, deste modo o tempo de execução para o pior caso é dado por $s \times r \times |A|^L$.

4.5 PONDERAÇÃO

Como o algoritmo leva em consideração todas as subsequências contínuas para encontrar as probabilidades, tem-se que o aumento de subsequências é quadrático e isto pode acarretar um problema caso o oponente possa ser modelado corretamente com apenas algumas poucas subsequências as outras subsequências podem ser consideradas como um tipo de ruído que acaba modificando os valores das probabilidades de cada ação a ser utilizada pelo oponente. Este ruído pode ocasionar um erro na escolha correta da ação que leva ao melhor resultado possível. Utilizando o mesmo exemplo do *tit for tat* utilizado na seção anterior para explicar a importância do *lookahead*. Supondo que exista um ruído de 30%, ou seja, caso o oponente cooperou no *round* passado calcula-se que existe 70% de chance dele cooperar e 30% dele trair. O resultado para a escolha de primeiro cooperar é de -0.6 e depois quando for trair -0.1 totalizando uma média de -0.35. Caso optar-se por trair na primeira ação, será obtido 1.3 e depois quando for trair novamente teremos -0.1, resultando em uma média de 0.6. Com isso é possível ver que um ruído de 30% leva a crer erroneamente que escolher trair duas vezes leva a uma escolha mais vantajosa do que escolher cooperar e depois trair. Para amenizar este problema do ruído, as subsequências são ponderadas de modo que as subsequências que causem mais ruído tenham um peso menor em relação as que modelam corretamente o oponente. Para encontrar uma boa ponderação é realizado uma descida do gradiente visando minimizar a entropia global de modo que as subsequências que tenham uma maior entropia pesem menos do que as mais puras. O peso de cada subsequência é modificado de acordo com o valor do gradiente como indicado na equação abaixo.

$$\alpha_j = \alpha_j - \mu \frac{\partial H(m, \alpha)}{\partial \alpha_j}, \tag{4.8}$$

em que α_j indica o valor poderado de uma subsequência de indice j, com j variando de

1 até $\frac{s(s+1)}{2}$, μ é o valor do passo da atualização do alpha e $H(m,\alpha)$ indica a entropia calculada utilizando a memória curta m e o conjunto de ponderações α. O cálculo de $\frac{\partial H(m,\alpha)}{\partial \alpha_j}$ é realizado as seguintes equações:

$$H(m,\alpha) = -\sum_{a\in A} P_a^\alpha(m) log(P_a^\alpha(m)) \tag{4.9}$$

$$\frac{\partial H(m,\alpha)}{\partial \alpha_j} = -\sum_{a\in A} log(1+P_a^\alpha(m)) \frac{\partial P_a^\alpha(m))}{\partial \alpha_j} \tag{4.10}$$

$$\frac{\partial P_a^\alpha(m))}{\partial \alpha_j} = \widetilde{P}_a^\alpha(m) - P_a^\alpha(m) \sum_{a\in A} \widetilde{P}_a^\alpha(m) \tag{4.11}$$

$$\widetilde{P}_a^\alpha(m) = \frac{\sum_{m'\in M} f_a(m') e^{\frac{-\|m-m'\|_\alpha^2 \gamma}{\|m\|_\alpha^2}} - \gamma \frac{\partial \|m-m'\|_\alpha^2}{\partial \|m\|_\alpha^2}}{\sum_{m'\in M} e^{\frac{-\|m-m'\|_\alpha^2 \gamma}{\|m\|_\alpha^2}}} \tag{4.12}$$

$$\frac{\partial \|m-m'\|_\alpha^2}{\partial \|m\|_\alpha^2} = \frac{2\|m\|_\alpha^2 (1-F(m,m')) - \|m-m'\|_\alpha^2}{\|m\|_\alpha^4} \tag{4.13}$$

Inicialmente todas as subsequências têm a mesma ponderação de $\frac{s(s+1)}{2}$ e a partir disto o gradiente as modifica até que as mudanças estejam menores que um certo valor σ. A escolha de γ não precisa ser feita de forma arbitrária. Na próxima seção será mostrado como ele pode ser calculado de modo a minimizar o número de iterações do gradiente.

4.5.1 REGRA DE ARMIJO

A Regra de Armijo é um algoritmo usado para determinar um tamanho de passo adequado para uma busca de minimização em linha, evitando que ele não seja muito grande de modo a evitar divergência e também que não seja pequeno demais de modo a não demorar para convergir a um ponto de mínimo que seja aceitável. Informalmente ela pode ser descrita como caso após o passo ser realizado e o ponto que se chegou seja melhor que o ponto de antes deve-se aumentar o tamanho do passo, caso o ponto seja pior do que o anterior então o tamanho do passo deverá ser diminuído. Este processo deve ser realizado até chegar no valor de mínimo. Esta regra pode ser interpretada graficamente como comparando o ponto em que se chegou com a linha tangente do ponto de onde se partiu: se estiver abaixo dela então deveria ter andado mais; se estiver acima dela então deveria ter andado

menos. A Regra de Armijo pode ser explicada por duas equações como será a mostrado a seguir.

$$F(x + \phi \times d) \leq F(x) + \varepsilon \times \phi \times \nabla F(x) \times d, \tag{4.14}$$

onde $f(x)$ é a função objetivo, ϕ é o tamanho do passo, d é a direção do passo e ε um parâmetro que deve ser maior ou igual a 0 e menor ou igual a 1. Se a Equação (*4.14*) for satisfeita então isto quer dizer que o tamanho de passo ϕ não é grande demais. Caso a Equação (*4.14*) esteja sendo infringida então deve-se aumentar o tamanho do passo pois ele está pequeno.

$$F(x + \eta \times \phi \times d) > F(x) + \varepsilon \times \eta \times \phi \times \nabla F(x) \times d, \tag{4.15}$$

onde η é um parâmetro que deve ser maior que 1. Se a Equação (*4.15*) for satisfeita então significa que o tamanho de passo ϕ não é pequeno demais. Caso a Equação (*4.15*) esteja sendo infringida então o tamanho do passo está grande e deve ser diminuído.

A implementação desta Regra de Armijo diminuiu em muito a quantidade de iterações requeridas pelo algoritmo para encontrar uma ponderação boa para as subsequências. No próximo capítulo serão mostrados os testes realizados com o algoritmo mostrando seus resultados quando confrontando com 20 autômatos clássicos no jogo do *dilema do prisioneiro* e contra o algoritmo *ELPH* no jogo *pedra papel tesoura*. Além disto, também é mostrado o quanto os incrementos do *lookahead*, ponderação e Armijo modificam seu tempo de execução e espaço requerido.

5 TESTES E RESULTADOS

Para verificar a eficácia do algoritmo *EDKS*, foram realizados um conjunto de testes. Para o jogo *dilema do prisioneiro* foram utilizadas 20 estratégias diferentes (PICCOLO; SQUILLERO, 2011) com o objetivo de verificar como seria o comportamento do algoritmo em um jogo de cooperação. Para verificar como seria o comportamento contra uma estratégia adaptativa foi utilizado o jogo *pedra papel tesoura* tendo como oponente o algoritmo *ELPH*.

Para que o *EDKS* consiga realizar a previsão das ações é necessário ter ao menos uma sequência em sua memória longa. Foi decidido que ele jogará aleatoriamente nos primeiros *rounds* até possuir uma sequência para ser utilizada na predição. O motivo da não utilização de uma estratégia de equilíbrio é para impedir que seja realizado a mesma ação, com isso possibilitando explorar como o adversário reage a diferentes ações. O mesmo foi escolhido para ser utilizado pelo *ELPH* enquanto ele não possuir uma sequência em sua memória longa. Definindo como s o tamanho da memória curta e A como o conjunto de ações, tem-se que a quantidade de memórias iniciais diferentes que podem ser geradas aleatoriamente é igual a $|A|^s$. Como todos algoritmos apresentam um comportamento determinístico tem-se que a diferença entre o resultado final de duas execuções é altamente influenciada pela execução dos primeiros *rounds*. Deste modo, a quantidade de vezes que um jogo deveria ser executado para saber como é o comportamento médio dos algoritmos em todos os casos cresce de modo exponencial em relação ao valor de s. Nesse sentido, optou-se neste trabalho por uma analise estatística dos dados considerando para cada experimento o cálculo das respectivas médias, desvio padrões e intervalos de confiança dos resultados obtidos.

5.1 DILEMA DO PRISIONEIRO

Os resultados dos gráficos referentes aos testes do *dilema do prisioneiro* devem ser interpretados da seguinte forma: se o resultado estiver entre -2 e -1 significa que o *EDKS* está cooperando enquanto o oponente o trai, este é o pior caso possível. Se estiver próximo a -1 então é equivalente a estratégia de sempre trair para ambos. Se estiver entre -1 e 0 então houveram interações além da traição mútua sendo elas não favoráveis ao

EDKS. Se estiver próximo a 0 então é equivalente a em todos os rounds ser realizado uma cooperação mútua. Entre 0 e 2 significa que o resultado foi mais vantajoso para o *EDKS* do que uma cooperação mútua em todos os rounds. Se for próximo a 2 significa que o oponente escolheu cooperar e o *EDKS* o traiu em todos os rounds, sendo esta a melhor situação possível.

Para o jogo do *dilema do prisioneiro* foram realizados testes contra 20 autômatos clássicos (PICCOLO; SQUILLERO, 2011). O tamanho da memória foi definido como 3 sendo que o automato com maior memória tem tamanho 5. Foram realizados 15 jogos de 1000 *rounds* contra os autômatos. A Figura *5.1.1* mostra a média obtida no último *round* acrescentada e subtraída do desvio padrão.

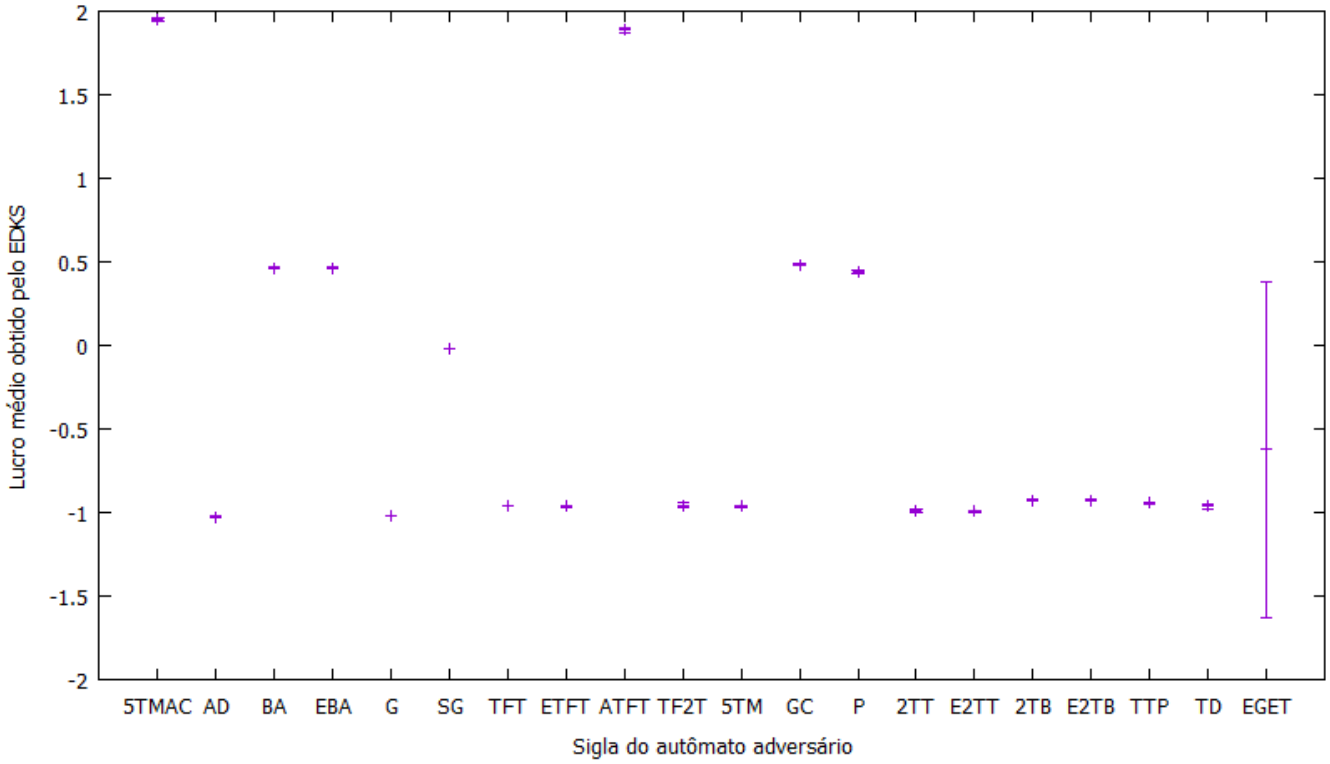

Figura 5.1.1: Lucro médio obtido pelo *EDKS* ao jogar contra diferentes autômatos no jogo *Dilema do prisioneiro*.

Pode-se perceber que contra nenhuma estratégia houve uma perda maior que -1. Contra seis estratégias o lucro ficou maior que 1. Contra outras estratégias como o *tit for tat* (TFT), por exemplo, o resultado foi o mesmo de sempre trair, estando bem distante do resultado ótimo contra ele, que seria terminar com resultado 0.

Para conseguir cooperar com o *tit for tat* e com outras estratégias é necessário a utilização de *lookahead* como será mostrado a seguir.

5.1.1 LOOKAHEAD

Para que seja possivel realizar cooperação é necessário a utilização de *lookahead*. A Figura *5.1.2* mostra o algoritmo *EDKS* sendo executado com *lookahead* de profundidade 1, em azul, sobreposto aos resultados da Figura *5.1.1*, em vermelho.

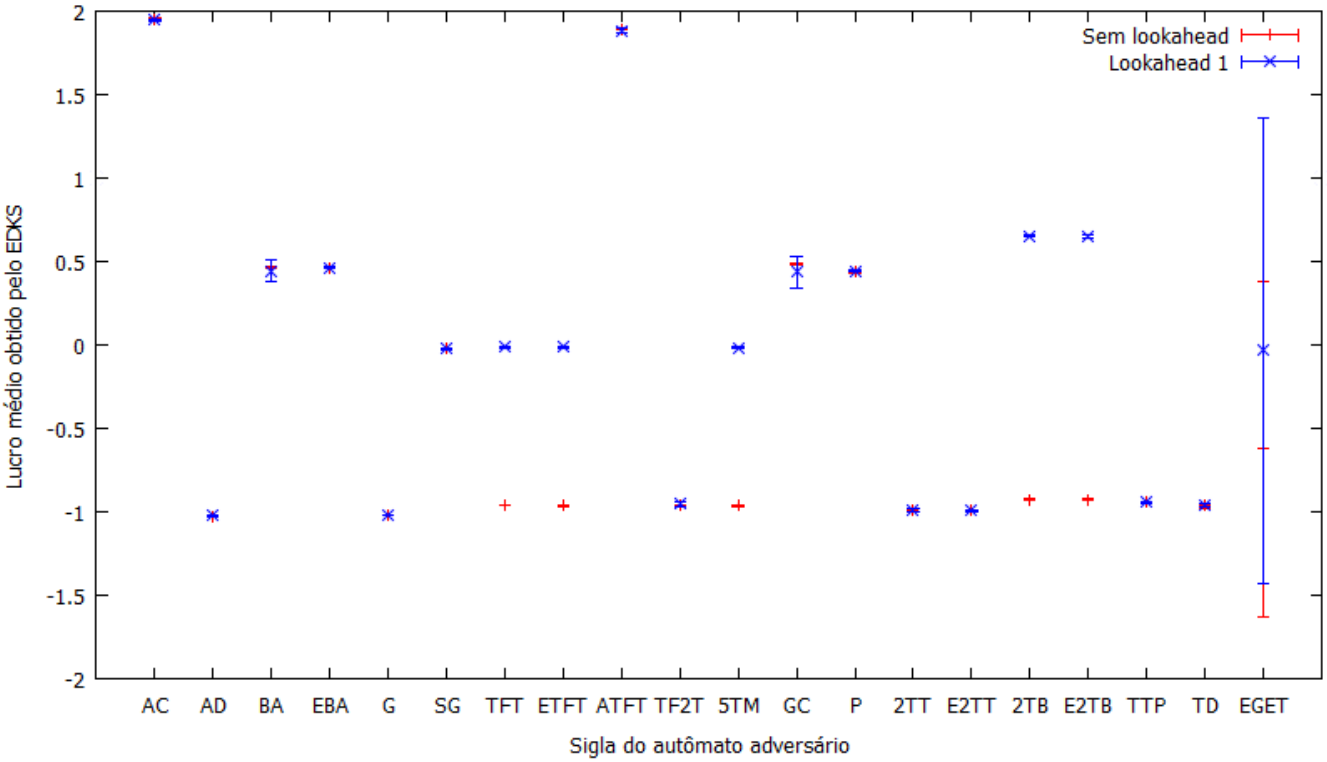

Figura 5.1.2: Lucro médio obtido pelo *EDKS* com lookahead 1, em azul, e sem lookahead, em vermelho, ao jogar contra diferentes autômatos no jogo *Dilema do prisioneiro*.

É possível perceber que com *lookahead* de tamanho 1 foi possível melhorar o resultado contra cinco estratégias. Contra três delas o resultado foi o mesmo que sempre cooperar. Para as outras duas obteve-se um resultado de média maior que 0. Isso mostra que a utilização do *lookahead* permite a cooperação com certos oponentes e até mesmo conseguir ganhar mais do que outros.

5.1.2 PONDERAÇÃO

A ponderação tem como objetivo diminuir o ruído causado por uma quantidade muito grande de subsequências que podem estar atrapalhando a predição correta do oponente. A ponderação deve ser usada caso o *lookahead* seja de tamanho maior que 1. Pode-se ver na Figura *5.1.3*, em azul, o algoritmo *EDKS* sendo executado com *lookahead* de profundidade

6, em conjunto com a ponderação, sobreposto aos resultados da Figura *5.1.2*, em vermelho. O valor 6 fora escolhido devido a ser uma unidade maior que o maior tamanho de memória dos autômatos.

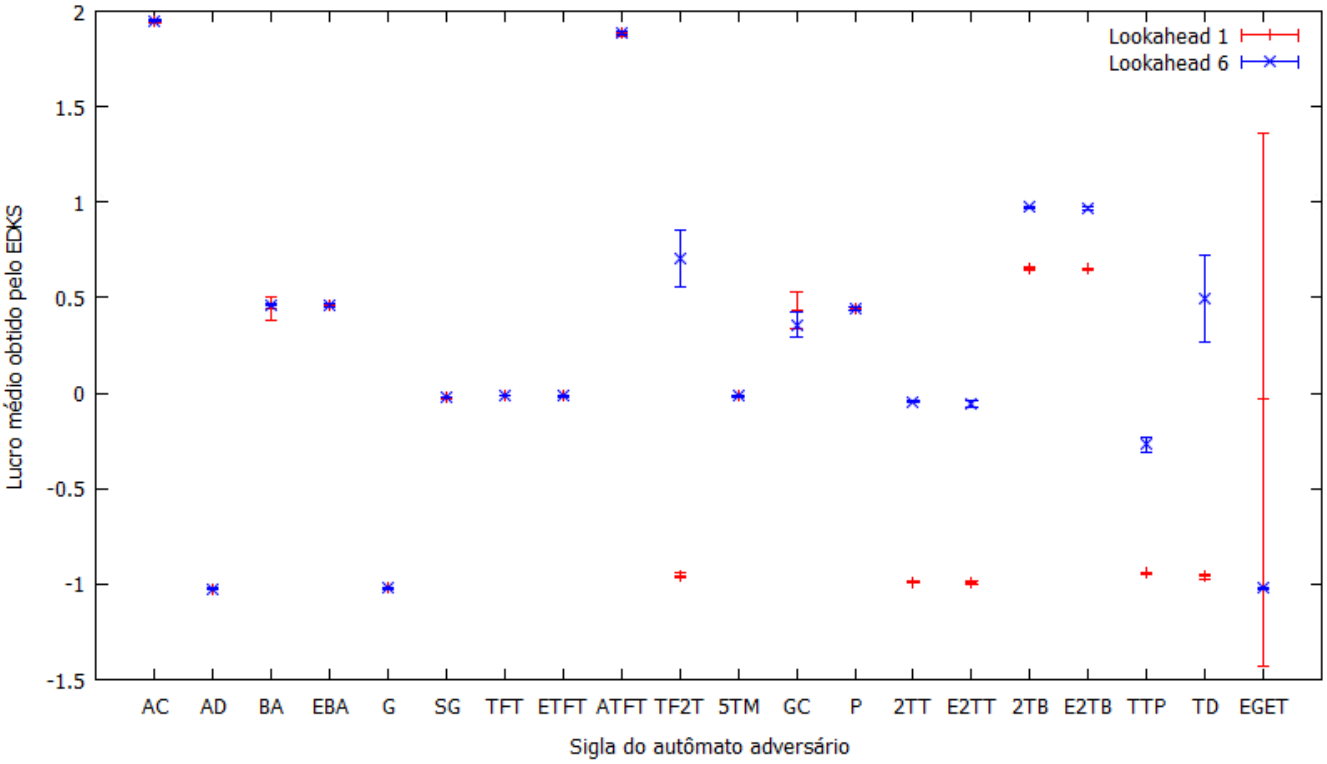

Figura 5.1.3: Lucro médio obtido pelo *EDKS* com lookahead 6, em azul, e com lookahead 1, em vermelho, ao jogar contra diferentes autômatos no jogo *Dilema do prisioneiro*.

Ao analisar a Figura *5.1.3* é possível ver uma melhoria considerável dos resultados contra algumas estratégias, em sete casos houveram melhorias. Em cinco deles foi possível sair de um resultado de sempre trair para um de sempre cooperar. Também pode-se observar em especial dentre eles dois casos onde o resultado final com a ponderação ficou maior que 0. Outros três resultaram em um lucro próximo de 0. Nos outros dois casos pode-se observar que com o *lookahead* 1 já se obtinha um lucro maior que 1, porém ao utilizar *lookahead* de profundidade 6 foi possível aumentar ainda mais o lucro.

5.2 PEDRA PAPEL TESOURA

O algoritmo *ELPH* foi escolhido para os testes devido a ser o estado da arte de algoritmos adaptativos para jogos de repetição. Como também é capaz de se adaptar em poucos *rounds*, ele foi escolhido para testar o *EDKS* contra um oponente adaptativo complexo.

Foi avaliado como o *EDKS* se comporta jogando contra o *ELPH* realizando 15 jogos de 1000 *rounds* e ambos possuindo uma memória curta de mesmo tamanho. O valor 7 foi escolhido devido a ser o valor utilizado na literatura (JENSEN et al., 2005a,b). A Figura *5.2.1* mostra esta comparação, a linha do meio mostra a média enquanto as linhas segmentadas mostram a média acrescentada e diminuída pelo desvio padrão. Pode-se observar que após 80 *rounds* o *EDKS* começa a ganhar do *ELPH* e este comportamento se estabiliza após 200 *rounds*. Pode-se dizer com intervalo de confiança de 99.8% que no *round* 1000 o algoritmo *EDKS* está ganhando do *ELPH*.

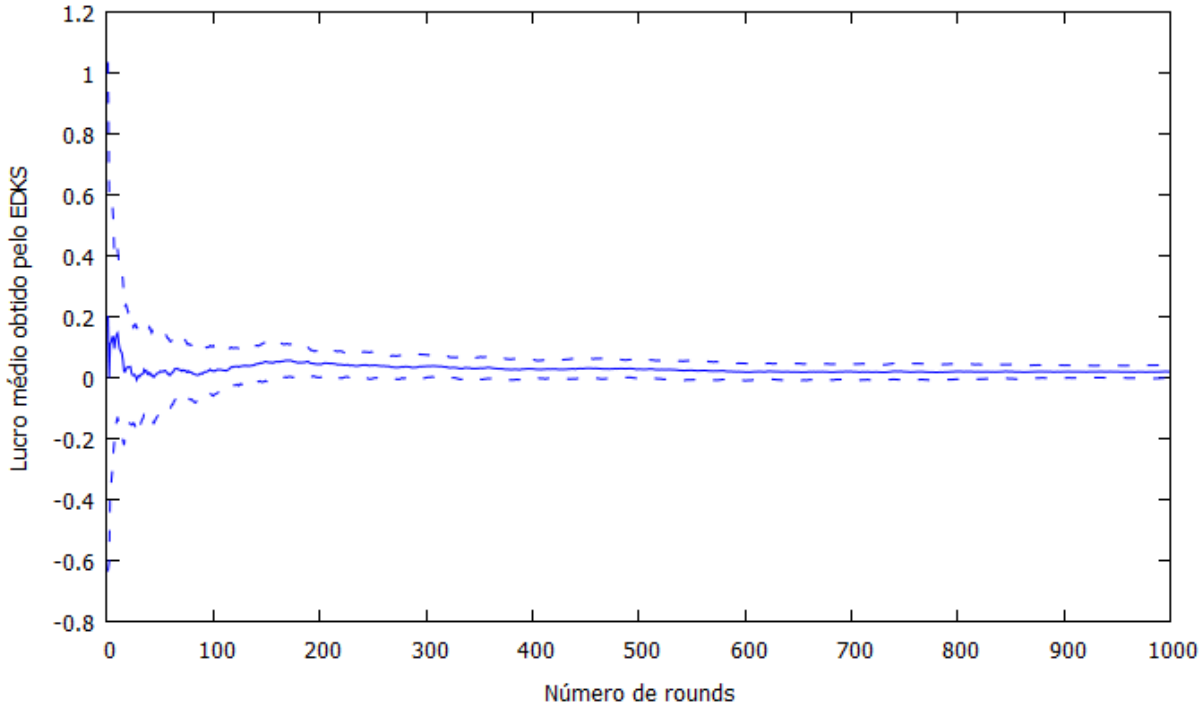

Figura 5.2.1: Lucro médio obtido pelo *EDKS* jogando contra o *ELPH* no jogo *pedra papel tesoura*.

Também foi verificado como a variação do tamanho da memória curta de ambos os algoritmos influencia o resultado final. A Figura *5.2.2* mostra a influência da variação do tamanho da memória do *EDKS* com o tamanho de memória do *ELPH* sendo fixada a 7. A linha contínua mostra o lucro médio recebido pelo algoritmo *EDKS* após 1000 *rounds* enquanto as linhas segmentadas mostram o desvio padrão adicionado e subtraído da média. Observando a figura 5.2.2 pode-se perceber que o *EDKS* obtem uma média superior a do *ELPH* quando o tamanho de sua memória curta se a próxima do tamanho da do *ELPH*. O melhor resultado fora obtido quando a memória de ambos estão com o mesmo tamanho. Caso seja aumentado mais o tamanho da memória do *EDKS*, não é

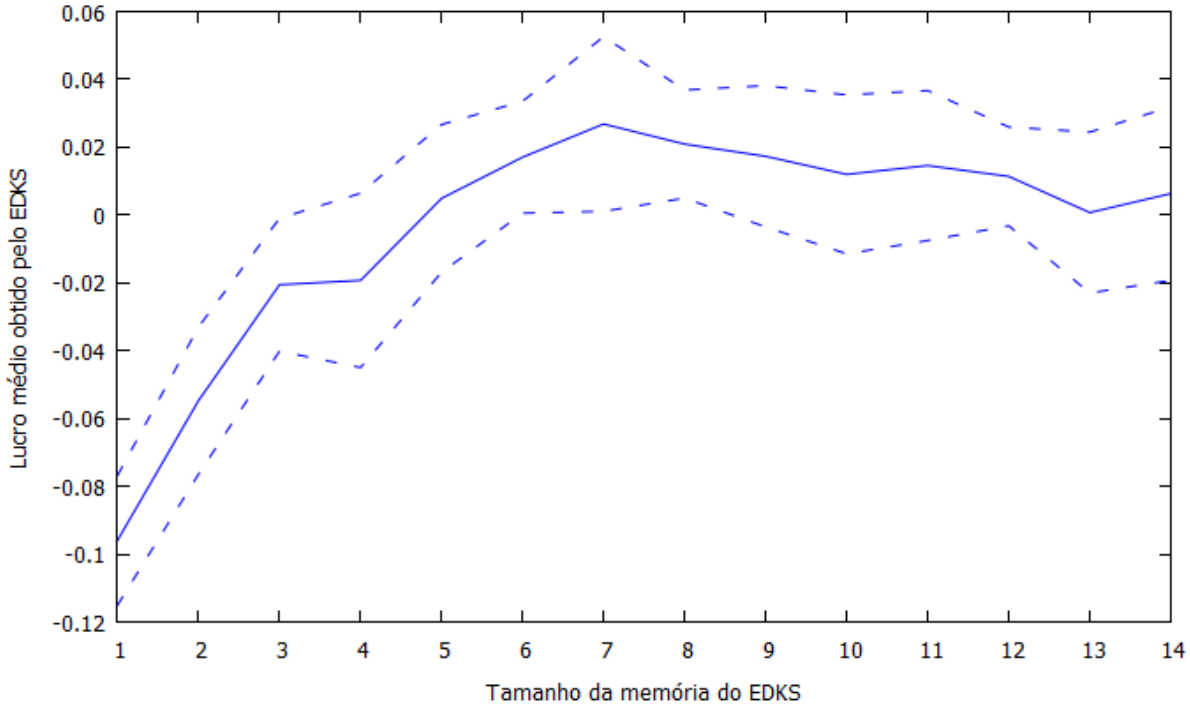

Figura 5.2.2: Lucro médio obtido pelo *EDKS* contra o algoritmo *ELPH* no jogo *pedra papel tesoura* à medida que a memória do *EDKS* é aumentada.

obtido uma melhoria no resultado, porém o *EDKS* continua superando o *ELPH*.

A Figura *5.2.3* analisa a situação inversa, com o tamanho de memória do *ELPH* variando enquanto o *EDKS* tem a memória fixada em 7. A linha contínua mostra o lucro médio recebido pelo algoritmo *EDKS* após 1000 *rounds* enquanto as linhas segmentadas mostram o desvio padrão adicionado e subtraído da média. Pode-se perceber que mesmo quando o tamanho da memória curta do *ELPH* chega a ser o dobro da memória do *EDKS*, com tamanho 14 enquanto o *EDKS* está com tamanho 7, o *EDKS* consegue ter uma média superior a 0.

Também foi analisado a utilização do *lookahead* no jogo *pedra papel tesoura* como pode ser visto na Figura *5.2.4*. Pode-se dizer com confiança de 99.999% de que o *SKDE* ganha do *ELPH*.

5.2.1 CUSTO DE TEMPO

O custo de tempo do *EDKS* também foi avaliado. Como as estratégias utilizadas no jogo do *dilema do prisioneiro* tem um funcionamento simples elas acabam gerando poucas sequências diferentes o que faz o *EDKS* ficar próximo de seu melhor caso. Portanto para a avaliação dos custo de tempo e espaço foi utilizado o jogo *pedra papel tesoura*

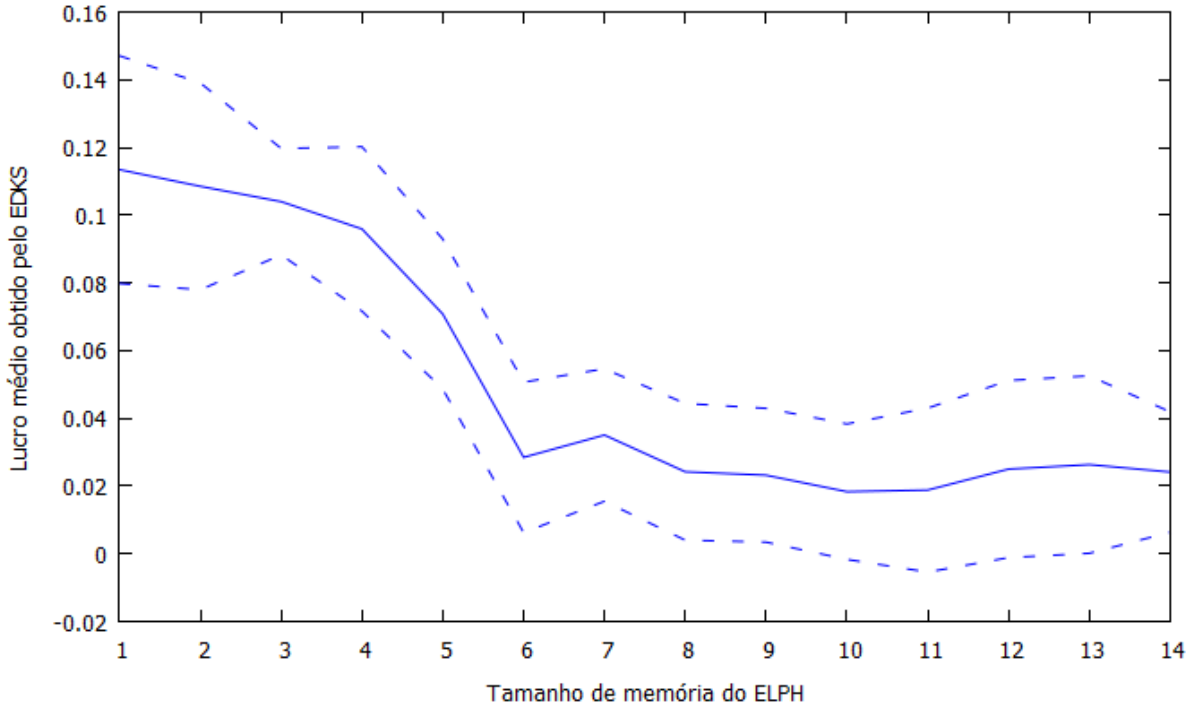

Figura 5.2.3: Lucro médio obtido pelo *EDKS* contra o algoritmo *ELPH* no jogo *pedra papel tesoura*, a medida que a memória do *ELPH* é aumentada.

comparando-se o desempenho do algoritmo *EDKS* contra o algoritmo *ELPH*.

A Figura *5.2.5* mostra o quanto o tempo de execução aumenta em relação ao aumento no tamanho de memória. Em azul está representado a memória do *EDKS* sem ponderação variando enquanto a do *ELPH* é fixada a 7 enquanto que na linha vermelha representa a memória do *ELPH* variando enquanto a do *EDKS* é fixada a 7. Pode-se perceber que o custo de tempo do *ELPH* tem um crescimento não polinomial enquanto o do *EDKS* é bem mais rápido. Também é possível perceber que a utilização do *lookahead* leva a um aumento no custo

A Figura *5.2.6* mostra o custo de tempo ao variar o tamanho do *lookahead* do *EDKS* contra o *ELPH*, ambos com memória fixada a tamanho 7. Pode-se perceber que o aumento da profundidade do *lookahead* leva a um crescimento considerável no tempo de execução.

5.2.2 CUSTO DE ESPAÇO

Para realizar o cálculo da memória requerida pelo algoritmo *EDKS* foi realizado a contagem de quantas inserções de sequências diferentes foram adicionadas. Os resultados podem ser observados nas Figuras *5.2.7* e *5.2.8*. O mesmo foi feito para o algoritmo *ELPH*. Pode-se observar que enquanto o custo de espaço do *ELPH* cresce de modo expo-

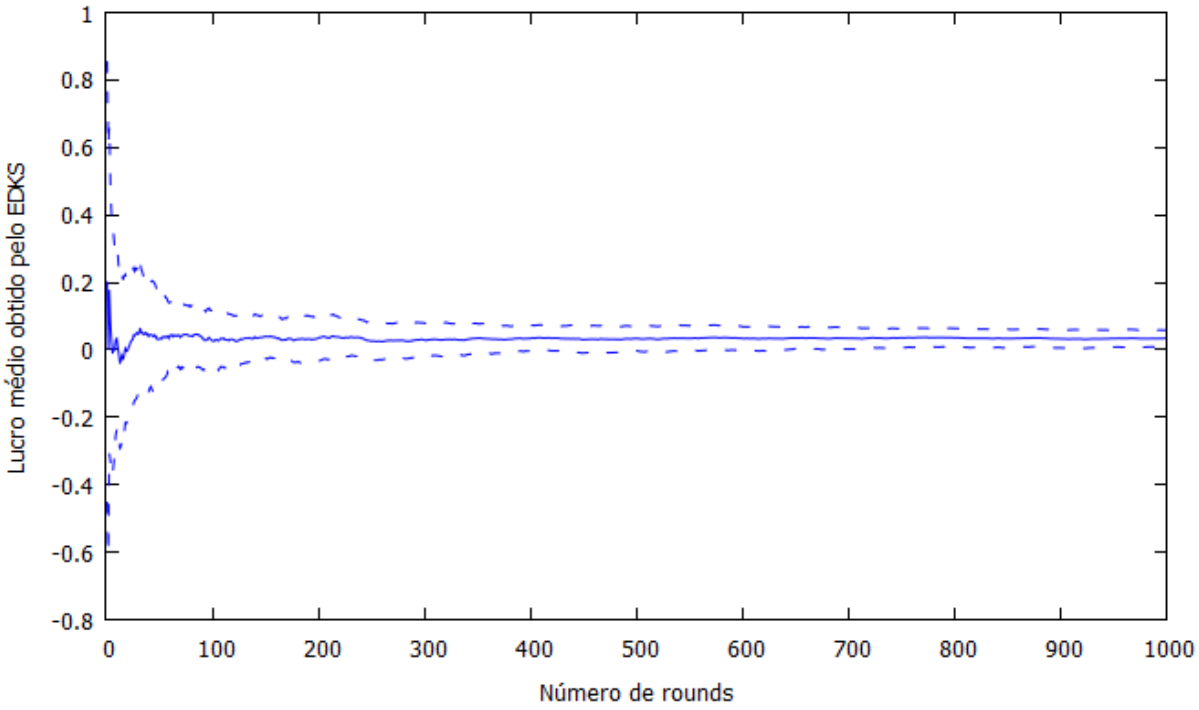

Figura 5.2.4: Lucro médio obtido pelo *EDKS* com *lookahead* 1 jogando contra o *ELPH* no jogo *pedra papel tesoura*.

nencial o *EDKS* apresenta um crescimento linear. Além disto pode-se notar que o custo do *ELPH* está várias ordens de grandeza acima do custo do *EDKS*.

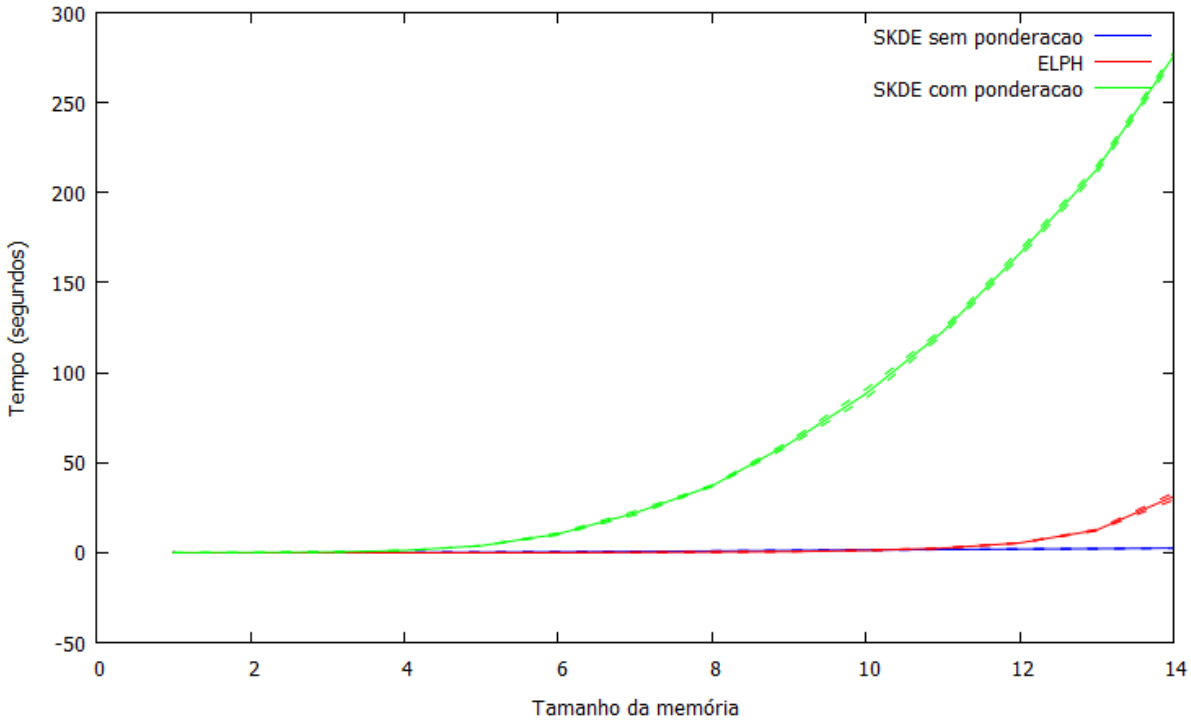

Figura 5.2.5: Custo de tempo do *EDKS* comparado com o custo de tempo do *ELPH*.

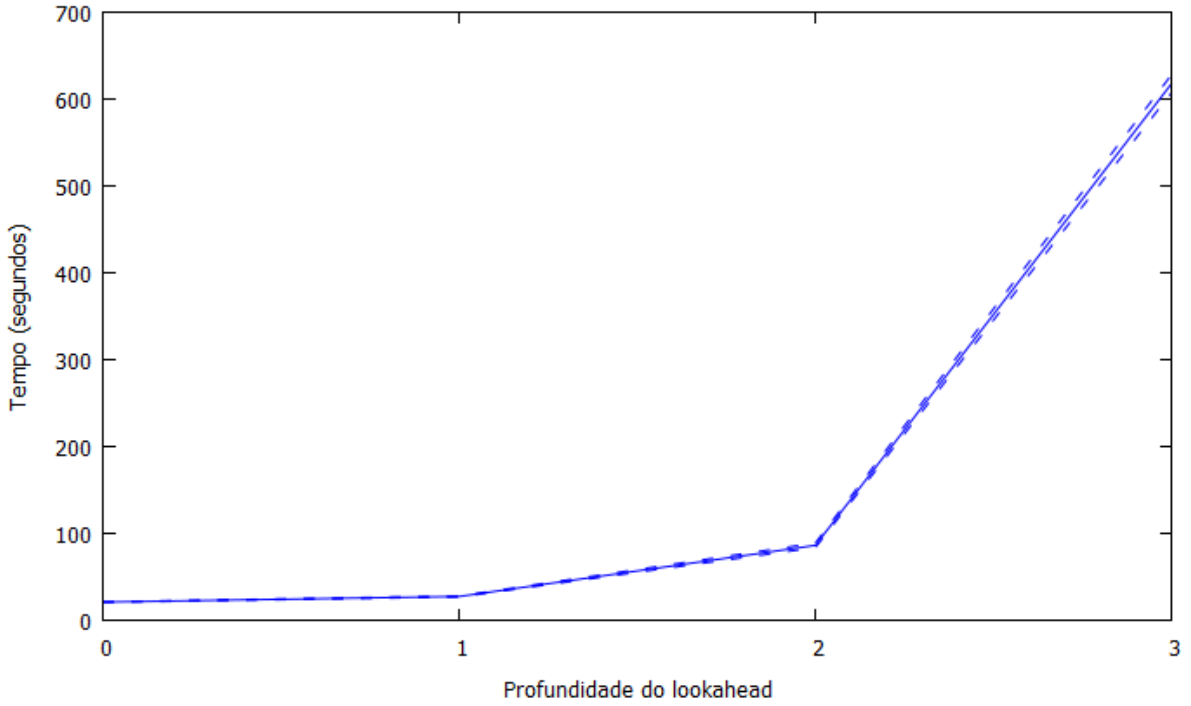

Figura 5.2.6: Custo de tempo do *lookahead*.

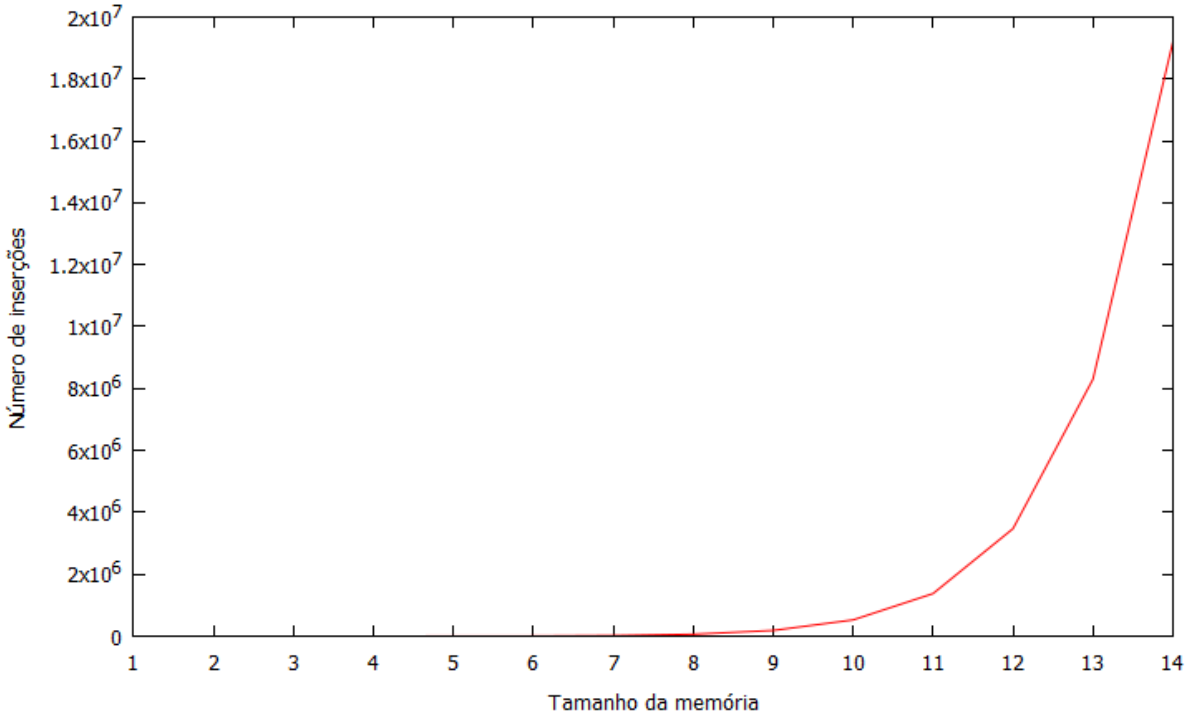

Figura 5.2.7: Custo de espaço do *ELPH*.

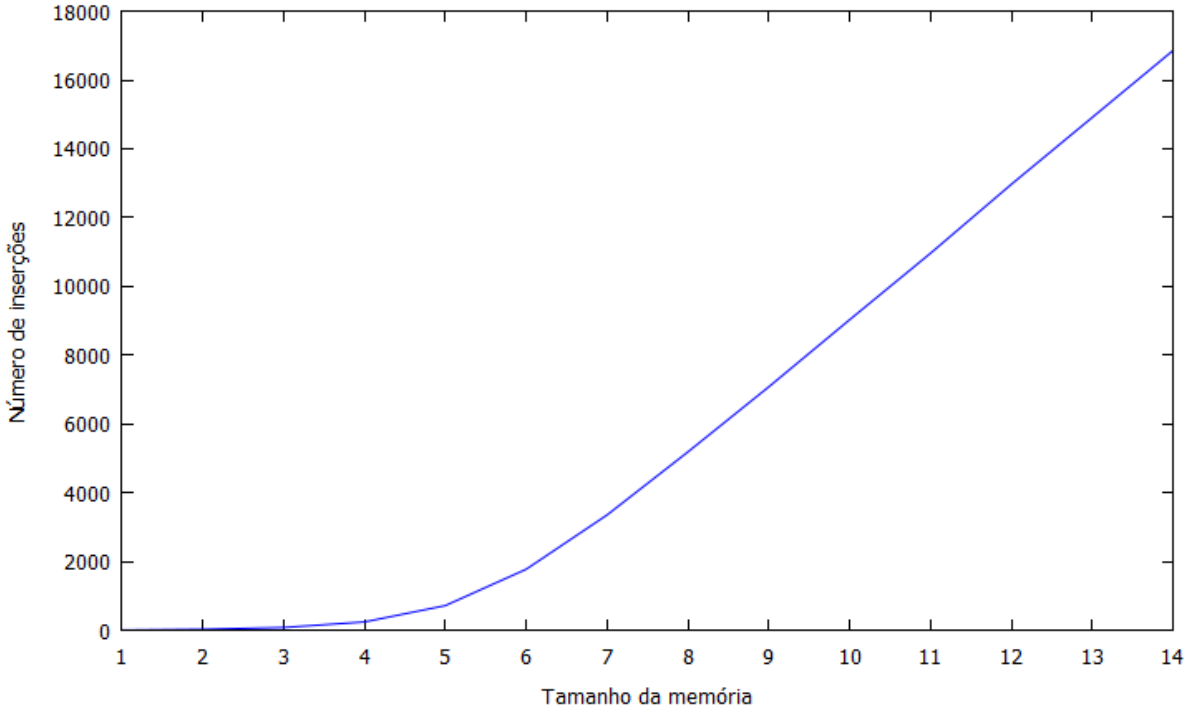

Figura 5.2.8: Custo de espaço do *EDKS*.

6 CONCLUSÕES E TRABALHOS FUTUROS

Nesta dissertação foi proposto um algoritmo para jogos de repetição capaz de adaptar sua estratégia para obter melhores resultados. A velocidade de adaptação se mostrou ser rápida, necessitando de poucos *rounds* para modelar o oponente. O algoritmo apresentado também consegue realizar *lookahead*, o que é necessário em uma estratégia adaptativa para obter o resultado ótimo contra certas estratégias como pode ser visto no jogo *dilema do prisioneiro*.

O algoritmo desenvolvido realiza o aprendizado da estratégia do oponente armazenando sequências de ações consecutivas. São armazenadas as ações utilizadas tanto pelo jogado quanto pelo o oponente. Junto destas sequências também são armazenadas as frequências das ações que as sucederam. Estas sequências são utilizadas por um *kernel* para o cálculo das probabilidade de cada ação ser utilizada pelo oponente. Esta distribuição de probabilidades é utilizada para escolher o próximo movimento a ser utilizado pelo jogador.

Para obter melhores resultados em jogos como o *dilema do prisioneiro*, é necessário a utilização de *lookahead*. O algoritmo *EDKS* pode realizar o cálculo de *lookahead* simulando como a memória curta estará no proximo *round*. Este processo pode ser realizado recursivamente para aumentar a profundidade do *lookahead*.

Foram realizados testes contra o algoritmo *ELPH* no jogo *pedra papel tesoura*. O *EDKS* obteve resultados melhores, sendo possível dizer com um alto nível de confiança que ele derrota o *ELPH*.

Também foi mostrado vantagens na quantidade de espaço requerido pelor *EDKS*, tendo um crescimento linear, enquanto o *ELPH* apresenta um crescimento exponencial.

Foram também realizados testes no jogo *dilema do prisioneiro* onde o algoritmo se mostrou ter bons resultados quando utilizado em conjunto com *lookahead* e ponderação. Foi possível ganhar de alguns oponentes e ficar bem próximo do valor ótimo de outros. Contra nenhum oponente foi obtido um lucro menor do que -1, ou seja, cooperar enquanto o oponente trai.

Por fim, é possível concluir que o algoritmo desenvolvido consegue superar outros métodos que são o estado da arte atual para jogos de repetição. Como trabalho futuro

tem-se a possibilidade de utilização de programação dinâmica com objetivo de realizar o cálculo de um *lookahead* infinito. Além disso, cogita-se a aplicação do algoritmo em jogos mais complexos como, por exemplo, jogos de luta e jogos de azar.

REFERÊNCIAS

ARMSTRONG, M.; PORTER, R. (Ed.). **Handbook of Industrial Organization**, 1989. 329–414 p. ISBN 978-0-444-82435-6.

AXELROD, R. M. **The Evolution of Cooperation**, 2006. ISBN 0-465-00564-0.

BERGER, U. Brown's original fictitious play. **Journal of Economic Theory**, v. 135, n. 1, p. 572–578, jul. 2007.

BICCHIERI, C. **The Grammar of Society: The Nature and Dynamics of Social Norms**, 2006. ISBN 0-521-57490-0.

BOWLING, M.; VELOSO, M. Multiagent learning using a variable learning rate. **Artificial Intelligence**, v. 136, p. 215–250, mai. 2002.

BROWN, G. W. Iterative solutions of games by fictitious play. **Activity Analysis of Production and Allocation**, p. 374–376, 1951.

DASKALAKIS PAUL W. GOLDBERG, C. H. P. C. The complexity of computing a nash equilibrium. **SIAM Journal on Computing**, v. 39, n. 1, p. 195–259, Mai. 2009. ISSN 0097-5397.

JENSEN, S.; BOLEY, D.; GINI, M.; SCHRATER, P. Non-stationary policy learning in 2-player zero sum games. In: AAAI, 2005.

JENSEN, S.; BOLEY, D.; GINI, M.; SCHRATER, P. Rapid on-line temporal sequence prediction by an adaptive agent. **In Fourth International Joint Conference on Autonomous Agents and Multiagent Systems**, jul. 2005.

JR., J. F. N. Non-cooperative games. **The Annals of Mathematics**, v. 54, n. 2, p. 286 –295, set. 1951.

KNUTH, D. E. **The Art of Computer Programming: Volume 3: Sorting and Searching**, 1998. ISBN 0201896850.

LITTMAN, M. L.; STONE, P. A polynomial-time nash equilibrium algorithm for repeated games. **ACM Conference on Electronic Commerce**, jun. 2003.

MERTENS, J.-F. Repeated games. **Proceedings of the International Congress of Mathematicians**, 1986.

NEUMANN, J. von; MORGENSTERN, O. **Theory of Games and Economic Behavior**, 1944. ISBN 978-0691130613.

PICCOLO, E.; SQUILLERO, G. Adaptive opponent modelling for the iterated prisoner's dilemma. **Proceedings of the IEEE Congress on Evolutionary Computation**, jun. 2011.

SELTEN, R. J. R. A simple model of imperfect competition, where 4 are few and 6 are many. **International Journal of Game Theory**, v. 2, n. 1, p. 141–201, dez. 1973. ISSN 1432-1270.

SHANNON, C. E.; WEAVER, W. **The Mathematical Theory of Communication**, 1963. ISBN 0-252-72548-4.

SMITH, J. M. **Evolution and the Theory of Games**, 1982. ISBN 0-521-28884-3.

TUROCY, T. L.; STENGEL, B. von. Game theory. **Encyclopedia of Information Systems**, p. 403–420, 2002.

ZIONTS, S. **Linear and Integer Programming**, 1974. ISBN 0135367638.