UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO MARCOS BRITO MATIAS

ESTRATÉGIAS VENCEDORAS PARA JOGOS DE CONVEXIDADE EM GRAFOS

FORTALEZA

2023

JOÃO MARCOS BRITO MATIAS

ESTRATÉGIAS VENCEDORAS PARA JOGOS DE CONVEXIDADE EM GRAFOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Rudini Sampaio.
Coorientador: Prof. Me. Samuel Nascimento de Araújo.

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M38e Matias, João Marcos Brito.
Estratégias vencedoras para jogos de convexidade em grafos / João Marcos Brito Matias. – 2023.
54 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Ciências, Curso de Computação, Fortaleza, 2023.
Orientação: Prof. Dr. Rudini Menezes Sampaio.
Coorientação: Prof. Me. Samuel Nascimento de Araujo.

1. Jogos de convexidade. 2. Teoria de Sprague-Grundy. 3. Jogos de combinatória. I. Título.
CDD 005

JOÃO MARCOS BRITO MATIAS

ESTRATÉGIAS VENCEDORAS PARA JOGOS DE CONVEXIDADE EM GRAFOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 30/11/2023.

BANCA EXAMINADORA

Prof. Dr. Rudini Sampaio (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Samuel Nascimento de Araújo (Coorientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Prof. Dra. Rosiane de Freitas
Universidade Federal do Amazonas (UFAM)

Aos meus pais, por sua capacidade de acreditar e investir em mim. Aos meus irmãos pelas manifestações de carinho e amor que trouxeram alegria e motivação para continuar. Por fim, às minhas tias que me inspiram manifestando cuidado, abnegação e resiliência.

AGRADECIMENTOS

À Instituição CNPq, pelo apoio financeiro com a manutenção da bolsa de auxílio.

Aos professores Rudini Sampaio e Samuel Araújo pela excelente orientação.

Aos professores participantes da banca examinadora Rosiane de Freitas e Ana Karolina Maia de Oliviera pelo tempo, pelas valiosas colaborações e sugestões.

Aos meus amigos André Braz, Hugo Carlos, Juliany Ferreia e Weberty Rocha pelas reflexões, críticas e bons momentos.

Aos membros do laboratório de pesquisa ParGO do Departamento de Computação da Universidade Federal do Ceará pela colaboração e sugestões recebidas.

RESUMO

A teoria dos jogos combinatórios é uma linha da matemática e da teoria da computação na qual estuda jogos sequenciais de informação perfeita. Um jogo combinatório é dito imparcial se a única diferença entre o primeiro e o segundo jogador é que o primeiro começa o jogo, mas ambos possuem as mesmas possibilidades de jogada. A teoria de Sprague-Grundy diz que todo jogo imparcial finito pode ser associado a um número, chamado nimber, referente ao tamanho de uma pilha do jogo Nim, um clássico jogo imparcial resolvido matematicamente por C.Bouton em 1901. Com isso, determinar o nimber de um jogo imparcial é importante para obter uma estratégia vencedora, de acordo com as estratégias conhecidas do jogo Nim. Além disso, um jogo combinatório é dito partizan quando alguns movimentos são permitidos para um jogador e não para outro. Uma das contribuições de John H. Conway na teoria dos jogos partizan foi associar um número diádico a esses jogos com o intuito de obter estratégias vencedoras neles. De outro modo, de acordo com Duchet (1987), o primeiro artigo sobre convexidade em grafos, em inglês, foi o artigo *"Convexity in graphs"* publicado em 1981. Um dos seus autores, Frank Harary, introduziu em 1984 os primeiros jogos de convexidade em grafos, focado na convexidade geodésica, que foram investigados em uma sequência de cinco artigos que terminou em 2003. Diante disso, nesse trabalho, continuaremos essa linha de pesquisa. As definições dos jogos de convexidade foram estendidas para outras convexidades e obtivemos resultados para as suas estratégias vencedoras. Dentre esses resultados, solucionamos os jogos em árvores através da teoria de Sprague-Grundy em jogos imparciais e através da teoria de Conway para jogos combinatórios partizan.

Palavras-chave: jogos de convexidade; jogos imparciais; teoria de Sprague-Grundy; jogos partizan; Conway.

ABSTRACT

The combinatorial games theory is a branch of mathematics and theoretical computer science that studies sequential games with perfect information. A combinatorial game is called impartial if the only difference between the first players and second player is that the first started the game, but both players have the same possibility of play. The Sprague-Grundy's theory says that every finite impartial game can be associated with a number, called nimber, regarding size of a Nim game heap, a classical impartial game solved for C. Bouton in 1901. . Therefore, determine the nimber of a impartial game is important to obtain a winning strategy, acoording to know winning strategies of the Nim game. Moreover, a combinatorial game is called partizan if the some moves are allowed for a player and not to another. One of John H. Conway's contributions in partizan game theory was associated a dyadic number with this games in order to obtain winning strategies in them. Futhermore, accordingly to Duchet (1987), the first paper of convexity on graphs, in english, it the paper "Convexity in graphs"published in 1981. One of its authors, Frank Harary, introduced in 1984 the first graph convexity games, focused on the geodesic convexity, which were investigated in a sequence of five papers tha endend in 2003. In this work, we continue this research line, extend these games to other graph convexities, and obtain winning strategies results. Among them, we obtain winning strategies for trees from the Sprague-Grundy theory on impartial games and Conway's combinatorial game theory on partizan games.

Keywords: convexity games; impartial games; Sprague-Grundy's theory; partizan games; Conway.

SUMÁRIO

1 INTRODUÇÃO

Segundo Roger B. Myerson (MYERSON, 1997), a teoria dos jogos pode ser definida como o estudo de modelos matemáticos de conflito e cooperação entre inteligências racionais capazes de tomar decisão. Em outras palavras, a teoria dos jogos provê técnicas matemáticas para analisar situações em que dois ou mais indivíduos tomam decisões que irão influenciar o estado uns dos outros. Myerson afirma que na linguagem da teoria dos jogos, um *jogo* se refere a qualquer situação social envolvendo dois ou mais indivíduos, chamados de *jogadores*. Logo, a teoria dos jogos buscar analisar conflitos e teorizar a respeito de tomadas de decisão.

A teoria dos jogos moderna teve seu início a partir de 1910 com o estudo de estratégias para jogos de dois jogadores e de soma-zero, ou seja, jogos em que o ganho de um jogador representa necessariamente perda para o outro jogador. Damas, Xadrez e jogo da velha são exemplos de jogos de dois jogadores e de soma-zero.

Nesse sentido, a teoria dos jogos moderna produziu diversas aplicações em problemas reais. Como exemplo, em 1913, Ernst Zermelo demonstrou que o jogo de xadrez é estritamente dominado, isto é, um jogo em que a cada etapa da partida pelo menos um dos jogadores possui uma estratégia que lhe trará a vitoria ou conduzirá o jogo ao empate (ZERMELO, 1913).

Não obstante, um trabalho de grande destaque foi o do matemático húngaro John Von Neumann , quando em 1928 publicou o artigo *On the Theory of Games of Strategy* (NEUMANN, 1928) em que provava a existência de solução em estratégias mistas (quando se leva em consideração distribuições probabilísticas) para jogos finitos de soma-zero com dois jogadores. O trabalho de Von Neumann na teoria dos jogos culminou, em 1944, na publicação do livro *Theory of Games and Economic Behavior* (NEUMANN; MORGENSTERN, 1947), com co-autoria do economista Oskar Morgenstern. Nele, Von Neunmann e Morgenstern conceberam uma teoria matemática inovadora de organização econômica e social baseada na teoria dos jogos de estratégia.

Segundo, Roger B. Myerson (MYERSON, 1997), Robert W. Dimand (DIMAND, 1996) e John M. Smith (SMITH, 1982), a partir de 1950 a teoria dos jogos teve grandes avanços e foi aplicada em várias áreas do conhecimento como economia, sociologia e filosofia, assim como foi amplamente utilizada como ferramenta didática em ambientes escolares.

Em 1970, John M. Smith aplicou os conceitos da teoria dos jogos na biologia com a esperança de entender a evolução do comportamento de animais em ambientes sociais (SMITH, 1979). Além disso, em 1994, John Nash recebeu o Prêmio Nobel de Ciências Econômicas

por sua contribuição na teoria dos jogos aplicada na economia. Nash introduziu o conceito de equilíbrio de Nash que soluciona jogos não cooperativos, isto é, jogos em que os jogadores não podem formar alianças.

De maneira geral, a teoria dos jogos procura entender conflitos e cooperações entre indivíduos através de modelos quantitativos e exemplos hipotéticos. Normalmente esses exemplos são mais simples do quê os problemas reais, pois ignoram menores detalhes. Contudo, mesmo em contextos onde as posições dos indivíduos não são bem definidas, ainda é possível compreender melhor essas situações reais estudando os contextos hipotéticos da teoria dos jogos.

Tendo isso em vista, a teoria dos jogos é uma área da matemática que pode ser aplicada em diversos contextos e auxiliar no desenvolvimento e na análise de comportamentos nas relações entre indivíduos. Então, sabendo da importância do estudo desse ramo da matemática, esse trabalho visa estudar e apresentar resultados na teoria dos jogos combinatórios, um ramo da teoria dos jogos que estuda jogos sequenciais e de informação perfeita. Para isso se faz necessário a apresentação de algumas terminologias.

Um jogo é dito sequencial se os jogadores realizam movimentos alternadamente. Além disso, chamamos de "estado" ou "posição" de um jogo a disposição dos elementos que compõem o jogo em um dado momento. Assim como, dado um estado de um jogo combinatório, o conjunto dos movimentos possíveis contém todas as jogadas válidas que um jogador pode fazer naquele estado. Logo, um jogo é dito de informação perfeita se cada estado do jogo e cada conjunto de movimentos possíveis é conhecido por todos os jogadores.

Existem vários jogos sequenciais e de informação perfeita entre dois jogadores, a sabe: xadrez, damas, jogo da velha, Nim, Red-Blue Hackenbush, dentre outros. Muitos desses jogos já foram solucionados, isto é, dada qualquer posição desses jogos é possível dizer se o primeiro jogador possui um conjunto de movimentos que o levará a vitória.

Não obstante, Frank Harary no ano de 1984 introduziu os jogos de convexidade em grafos (HARARY, 1984), que serão apresentados doravante. Harary resolveu os jogos de convexidade para algumas classes de grafos, como ciclos, rodas e grafos bipartidos, mas para algumas outras classes de grafos, como árvores, o jogo ainda não havia sido solucionado.

Tendo isso em vista, o objetivo dessa pesquisa é, através dos conceitos definidos por Harary, introduzir novos jogos de convexidade em grafos e utilizar os métodos de resolução dos jogos matemáticos clássicos, já solucionados, para resolver os problemas de decisão nos jogos de convexidade em grafos.

Inicialmente serão apresentadas definições mais precisas a respeito dos jogos de combinatória, depois serão apresentados os conceitos de jogos de convexidade e, por fim, os resultados obtidos através dessa pesquisa solucionando os jogos de convexidade imparciais em árvores e apresentando a relação dos jogos de convexidade partizan com os jogos clássicos já trabalhados na literatura.

2 TEORIA DOS JOGOS DE COMBINATÓRIA

A teoria dos jogos combinatórios é um ramo da matemática e da teoria da computação que tipicamente estuda jogos sequenciais com informação perfeita. As regras dos jogos apresentados neste trabalho garantem que eles acabam após uma sequência finita de movimentos, e o vencedor é definido por quem fez o último movimento. Na versão normal, o último jogador a fazer um movimento ganha. Na versão misère, o último jogador a fazer um movimento perde.

Em 1913, Zermelo demonstrou (ZERMELO, 1913) que em jogos finitos, de informação perfeita e sem empates, um jogador sempre possui uma estratégia vencedora. Logo, para os jogos combinatórios desse estudo, o problema de decisão é se o primeiro jogador possui uma estratégia vencedora.

2.1 Jogos imparciais

Um jogo combinatório é dito imparcial se a única diferença entre o primeiro jogador e o segundo jogador é que o primeiro começa o jogo. Em outras palavras, todos os movimentos possíveis dependem apenas da posição do jogo e não de qual jogador irá fazer o movimento. Os dois jogadores nos jogos imparciais são tradicionalmente chamados de Alice, que é a primeira a jogar, e Bob.

2.1.1 Nim

Nim é um dos jogos mais importantes para a teoria dos jogos combinatórios pois é a base fundamental da teoria de Sprague-Grundy, utilizada para análise de estratégias vencedoras de jogos imparciais. O jogo consiste em a cada turno ambos os jogadores removerem objetos de montes disjuntos até não haver mais objetos. Em cada turno, um jogador escolhe um monte não vazio e remove uma quantidade positiva de objetos desse monte.

Uma instância de Nim é dada por uma sequência $\Phi = (h_1, h_2, \ldots, h_k)$, onde k é o número de montes e $h_i \geq 1$ é o tamanho do i-ésimo monte. Nim foi matematicamente resolvido por Bouton em 1901 para qualquer instância Φ (BOUTON, 1901). Em outras palavras, o problema para decidir qual dos jogadores tem uma estratégia vencedora é resolvido em tempo linear em ambas variantes (normal e misère) do Nim.

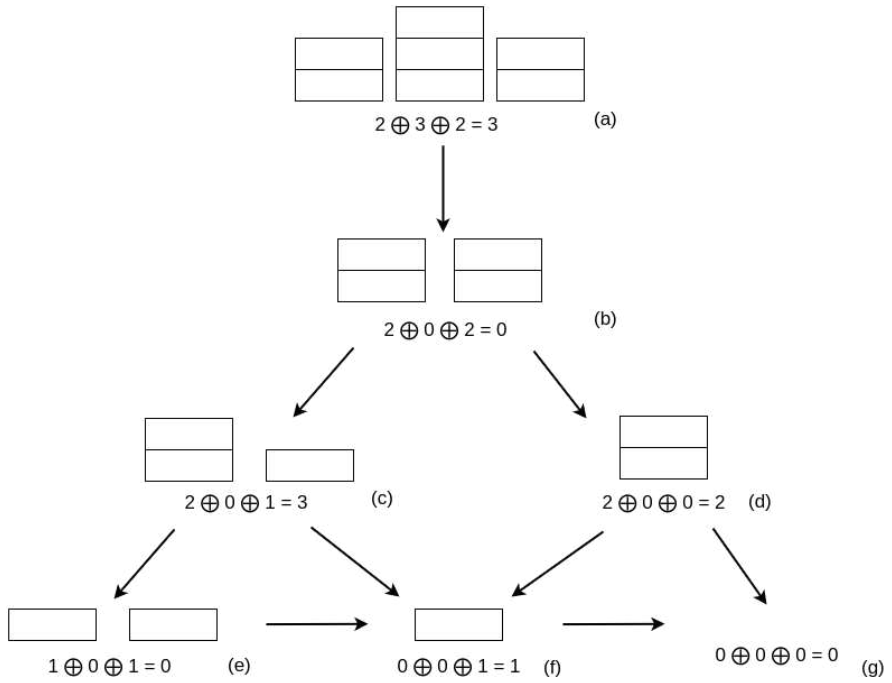
Dada uma sequência Φ com k elementos de uma instância no Nim, a função **nim-sum** : $\mathbb{N}^k \to \mathbb{N}$ é tal que **nim-sum**$(\Phi) = h_1 \oplus h_2 \oplus \cdots \oplus h_k$, onde $\oplus$ é a operação bitwise xor.

Então, qualquer instância em um jogo de Nim pode ser associada a um valor inteiro positivo.

A estratégia vencedora para o Nim está relacionada com a função **nim-sum**. Alice tem uma estratégia vencedora na variante normal de Nim se e somente se o **nim-sum**$(\Phi) > 0$. Assim como, Alice tem uma estratégia vencedora na variante misère de Nim se e somente se **nim-sum**$(\Phi) > 0$ e há pelo menos um monte com mais de um objeto ou **nim-sum**$(\Phi) = 0$ e todos os montes possuem exatamente um objeto.

A Figura 1 é um exemplo de uma representação de um jogo de Nim que inicia com instância representada por (a) e apresenta uma estratégia vencedora para Alice.

Figura 1 – Desenho de uma representação de um jogo de Nim



Fonte: elaborado pelo autor

Em (a), o **nim-sum**$(2,3,2) = 2 \oplus 3 \oplus 2 = 3$. Logo, em ambas as variantes Alice possui uma estratégia vencedora. Ela pode retirar todos os objetos da segunda pilha e isso resultará em (b) com **nim-sum**$(2,0,2) = 0$. Após isso, Bob ou removeria um objeto de uma das pilhas e isso resultaria em (c) com **nim-sum**$(2,0,1) = 3$, ou removeria todos os objetos de uma das pilhas e isso resultaria em (d) com **nim-sum**$(2,0,0) = 2$.

No caso de (c), Alice ainda teria estratégia vencedora para ambas as variantes. Na variante normal, Alice retiraria um objeto do primeiro monte e isso resultaria em (e) com **nim-sum**$(1,0,1) = 0$ e Bob retiraria o ultimo objeto de um dos dois montes levando a (f) e Alice, por fim, retiraria o último objeto do último monte resultando em (g) e venceria. Na variante

misère, Alice retiraria todos os objetos do primeiro monte e isso resultaria em (f), Bob teria que retirar o ultimo objeto do último monte e perderia.

No caso de (d), Alice na variante normal retiraria todos os objetos do monte resultando em (g) e venceria o jogo. Na variante misère, Alice removeria apenas um objeto do monte resultando em (f) e Bob perderia.

2.1.2 Teoria de Sprague-Grundy

A teoria de Sprague-Grundy, desenvolvida independentemente por R.P.Sprague (SPRAGUE, 1936) e P. M. Grundy (GRUNDY, 1939) em 1930, diz que é possível associar um número (chamado de *nimber*) a toda posição em um jogo imparcial, associando a um jogo de Nim de apenas um monte.

A conclusão com essa associação é que após um movimento em uma posição com nimber $h > 0$ de um jogo imparcial, é possível obter uma posição com qualquer nimber em $\{0, 1, \ldots, h-1\}$. Dado um estado H em um jogo imparcial, seja $\{h_1, \ldots, h_k\}$ o conjunto que contem todos os nimbers de uma posição obtida através de um movimento em H, o nimber de H pode ser calculado pelo valor $\text{mex}\{h_1, \ldots, h_k\}$, onde mex é o mínimo inteiro não negativo que não pertence ao conjunto. Além disso, uma posição obtida por uma união de k posições disjuntas com nimbers $h_1, \ldots, h_k$ tem nimber $h_1 \oplus \cdots \oplus h_k$.

Por fim, fazendo uma associação direta ao jogo Nim, para qualquer jogo imparcial de uma instância com nimber h temos que Alice possui uma estratégia vencedora na variante normal se e somente se $h > 0$. Além disso, Alice possui uma estretégia vencedora na variante misère se e somente se $h > 0$ e há pelo menos uma posição disjunta com nimber $h' > 1$, ou $h = 0$ e todas as posições disjuntas possuem nimber $h' = 1$.

2.2 Jogos partizan

Um jogo de combinatória é chamado partizan se ele não for imparcial, ou seja, se alguns movimentos são permitidos para um jogador e não para outro. Os dois jogadores nos jogos partizan são tradicionalmente chamados de Left (L) e Right (R). Algumas regras convencionais ajudam a organizar quem está jogando ou quais jogadas são permitidas: normalmente a cor azul (b**L**ue) está relacionada com Left e a cor vermelho (**R**ed) está relacionada com Right e as cores verde e cinza estão relacionadas com movimentos neutros.

2.2.1 Red-Blue Hackenbush

Hackenbush é um jogo de combinatória clássico inventado por John H. Conway. A construção e desenvolvimento da teoria de jogos de combinatória em jogos partizan de Conway é formalmente apresentada em dois livros clássicos: *"On numbers and games"* (CONWAY, 1976) e *"Winning ways for your mathematical plays"* (BERLEKAMP *et al.*, 1982). Hackenbush consiste em pontos conectados por segmentos de reta. Alguns desses pontos estão conectados a um "chão" representado, convencionalmente, como uma linha horizontal desenhada na parte inferior da área do jogo. Dois jogadores, Left e Right, alternadamente cortam algum segmento de reta e o segmento é deletado junto com qualquer outro segmento que não está mais conectado ao chão. Os argumentos nessa subseção são para a variante normal do Hackenbush, ganha o jogador que cortar o último conjunto de segmentos que estava ligado ao chão.

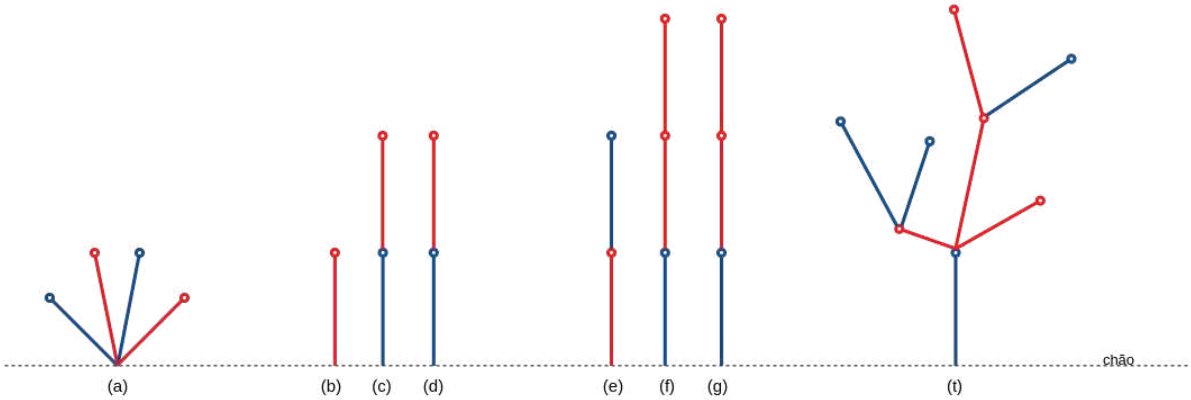
Red-Blue Hackenbush é uma versão de Hackenbush cujo os segmentos de reta são coloridos de vermelho ou azul. Left pode cortar apenas segmentos da cor azul e Right pode cortar apenas segmentos da cor vermelha.

Uma posição de um jogo Red-Blue Hackenbush é composta por partes disjuntas e Conway propõe que é possível dar um número diádico (número racional cujo denominador é uma potência de dois) a essas partes. Esse valor é mensurado em termos de movimentos livres para Left. Uma posição é chamada *"fuzzy"* se o próximo jogador sempre irá vencer. Uma posição em que o próximo jogador sempre irá perder terá valor zero. Valores positivos indicam uma estratégia vencedora para Left e valores negativos indicam uma estratégia vencedora para Right, não importa quem começar. Por fim, assim como nos jogos imparciais, que usam a operação bitwise xor na teoria de Sprague-Grundy, o valor de uma posição de um Red-Blue Hackenbush formada por n posições disjuntas é a soma de seus valores.

Por exemplo, a posição (a) na Figura 2 tem valor 0, logo o primeiro jogador irá perder, não importa se o primeiro jogador é Left ou Right. A posição (b) tem valor -1, então Blue perderá invariavelmente. Ambas as posições (c) e (d) têm valor 1/2, logo o jogo formado por (b), (c) e (d) terá seu valor igual a soma de seus valores que resultará em 0: o primeiro a jogar perde. Por fim, as posições (f) e (g) tem valor 1/4 cada e a posição (e) tem valor -1/2. Logo a soma das posições (f), (g) e (e) resulta em uma posição com valor 0.

O número surreal de uma posição no Red-Blue Hackenbush tem notação $\{L|R\}$ em que L < R e L é o maior valor possível dentre os valores das posições obtidas após um movimento de Left e R é menor valor possível dentre os valores das posições obtidas após um movimento de

Figura 2 – Desenho de instâncias no Red-Blue Hackenbush



Fonte: elaborado pelo autor

Right.

Outras notações são $\{L|\cdot\}$ e $\{\cdot|R\}$ com $L \geq 0$ e $R \leq 0$, onde $\cdot$ significa vazio (o jogador não tem movimentos). Por exemplo, a notação $\{\cdot|\cdot\} = 0$ (nenhum jogador possui movimentos e o primeiro a jogar perde), $\{n|\cdot\} = n+1$ ($n+1$ segmentos azuis e nenhum segmento vermelho), $\{0|\frac{1}{2^{k-1}}\} = \frac{1}{2^k}$ (uma linha com 1 segmento azul ligada ao chão seguida por k segmentos vermelhos, como a posição (f) da Figura 2) e $\{n|n+1\} = n + \{0|1\} = n + \frac{1}{2}$ (n segmentos azuis mais uma posição (c) da Figura 2).

Para computar o valor de $\{L|R\}$ em que satisfaça $L < \{L|R\} < R$ é utilizada a chamada "regra da simplicidade". O valor de $\{L|R\}$ é o único valor racional x que satisfaz $L < x < R$ cujo denominador é a menor potência de dois possível. Por exemplo, na posição (f) da Figura 2 após um movimento de Left temos o seguinte conjunto de valores $\{0\}$, além disso, após um movimento de Right podemos ter valor 1 ou podemos ter o mesmo valor da posição (c) que é $\frac{1}{2}$. Portanto, o jogo tem valor $\{0|\frac{1}{2}, 1\} = \{0|\frac{1}{2}\} = \frac{1}{4}$.

Outra maneira para computar o valor de caminhos, como a posição (f) da Figura 2, segue estes passos:

a) Contar a quantidade de segmentos azuis (ou vermelhos) que estão conectados em um caminho contínuo ligado ao chão;

b) Para cada outro segmento do caminho, é atribuído o valor dele como metade do valor do segmento anterior. Se o segmento que está sendo computado for azul esse valor será positivo, senão o valor será negativo.

c) Depois de atribuído todos os valores dos segmentos, basta somar todos eles.

Por exemplo, a posição (f) possui apenas uma aresta azul em um caminho contínuo ligado ao chão e terá valor 1. A próxima aresta é vermelha e tem valor $-\frac{1}{2}$ e a próxima aresta

vermelha tem valor $-\frac{1}{4}$. Logo, o valor da posição (f) é $1-\frac{1}{2}-\frac{1}{4}=\frac{1}{4}$.

2.2.1.1 Representação Red-Blue Hackenbush em strings

Uma posição Red-Blue Hackenbush pode ser representada por uma string. O conjunto $C=\{B,R,(,)\}$ contém todos os caracteres que podem pertencer a uma string nessa representação. Uma posição no jogo é uma sequência de $'B's$ e $'R's$. Caso haja alguma ramificação, ela é representada dentro de parênteses.

Por exemplo, a Tabela 1 mostra as strings para todas as posições da Figura 2.

Tabela 1 – Representação das posições da Figura 2 em strings

Posição	String
(a)	(B)(R)(B)(R)
(b)	R
(c)	BR
(d)	BR
(e)	RB
(f)	BRR
(g)	BRR
(t)	B (R(B)(B)) (R(R)(B)) (R)

Fonte: elaborada pelo autor.

2.2.1.2 Árvores Red-Blue Hackenbush

Dado uma posição P de um jogo Red-Blue Hackenbush, uma subposição de P é um jogo Red-Blue Hackenbush obtido após qualquer movimento válido em P.

Diante do exposto, utilizando a regra de simplicidade é possível computar recursivamente o valor de uma posição Red-Blue Hackenbush com valores das suas subposições. Em árvores, como a posição (t) na Figura 2, é possível fazer esse cálculo em tempo polinomial.

Uma das contribuições desse trabalho foi a implementação do algoritmo que recebe uma string de uma posição que é uma árvore no Red-Blue Hackenbush e retorna seu valor. A linguagem utilizada na implementação foi *Python 3.0*. Foram desenvolvidas funções para o cálculo do número diádico simples, para a análise e a manipulação da string da posição e para o cálculo recursivo do valor da posição usando técnicas de programação dinâmica para armazenar os valores de subposições já calculados na recursão. Os códigos-fontes da implementação estão disponíveis no Apêndice A e os algoritmos para computar esses valores estão descritos na Seção 4.2.

3 JOGOS DE CONVEXIDADE EM GRAFOS

Convexidade é um tópico clássico estudado em vários ramos da matemática. O estudo de convexidade aplicado em grafos foi iniciado recentemente. Em 1972, Erdős (ERDőS *et al.*, 1972) publicou um artigo que foi um dos primeiros sobre o tópico, focado em torneios (grafos orientados em que para qualquer par de vértices existe exatamente um arco entre eles). De acordo com Duchet (DUCHET, 1987), o primeiro artigo sobre convexidade em grafos gerais foi publicado em 1981 por Frank Harary e Juhani Nieminem (HARARY; NIEMINEM, 1981) chamado "Convexity in graphs". Em 1984, Harary introduziu os primeiros jogos de convexidade no resumo "Convexity in graphs: achievement and avoidance games"(HARARY, 1984). Frank Harary publicou uma sequência de cinco artigos (HARARY, 1984; BUCKLEY; HARARY, 1985b; BUCKLEY; HARARY, 1985a; NEcáSKOVá, 1988; HAYNES *et al.*, 2003), todos focados na convexidade geodésica.

Em conformidade com as definições apresentadas nos trabalhos supracitados, dado um grafo G e um conjunto $S \subseteq V(G)$, a função de intervalo geodésico $I_g(S)$ resulta em S e em todos os vértices de algum caminho mínimo entre dois vértices de S (EVERETT; SEIDMAN, 1985; FARBER; JAMISON, 1987) e a função de intervalo monofônico $I_m(S)$ resulta em S e em todos os vértices de algum caminho induzido entre dois vértices de S (DOURADO *et al.*, 2010; DUCHET, 1988; FARBER; JAMISON, 1986). Dado uma convexidade C, S é C-convexo se $I_C(S) = S$. O fecho C-convexo de S é o menor conjunto convexo $hull_C(S)$ que contém S. Sabe-se que $hull_C(S)$ pode ser obtido aplicando a função de intervalo $I_C(\cdot)$ exaustivamente em S até obter um conjunto convexo.

Dentre os jogos introduzidos por Harary (HARARY, 1984), há o jogo de intervalo geodésico e o jogo de intervalo geodésico fechado. Diante disso, podemos dar uma definição geral para os jogos de convexidade em grafos.

3.1 Jogos de convexidade imparciais

Definição 3.1. *Seja G um grafo. Nos jogos definidos abaixo, seja L o conjunto dos vértices rotulados inicialmente vazio e as definições dos conjuntos $f_1(L)$ e $f_2(L)$ dependem do jogo. Dois jogadores (Alice e Bob, começando por Alice) rotulam alternadamente um vértice não rotulado que não está em $f_1(L)$. O jogo acaba quando $f_2(L) = V(G)$.*

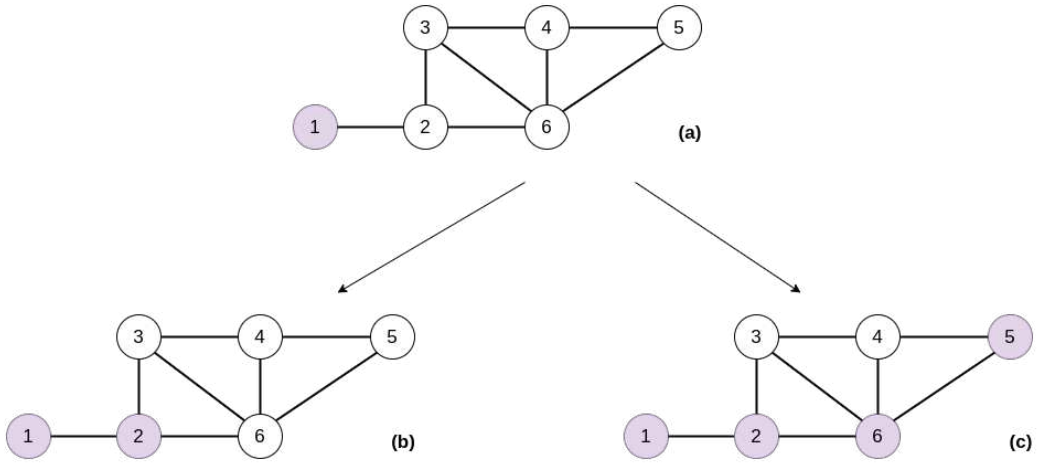
- *No jogo de envoltória HG_C: $f_1(L) = L$ e $f_2(L) = hull_g(L)$.*

– *No jogo de intervalo IG_C:* $f_1(L) = L$ *e* $f_2(L) = I_g(L)$.
– *No jogo de envoltória fechado CHG_C:* $f_1(L) = f_2(L) = hull_g(L)$.
– *No jogo de intervalo fechado CIG_C:* $f_1(L) = f_2(L) = I_g(L)$.

Os jogos da Definição 3.1 são imparciais visto que são jogos sequenciais, de informação perfeita e a única diferença entre Alice e Bob é que Alice começa o jogo. Nos jogos definidos acima há as variantes normal e misère (cuja definição foi apresentada no capítulo 2) e o problema de decisão é se Alice possui uma estratégia vencedora, isto é, se Alice ganhará caso jogue de maneira ótima.

A Figura 3 é um exemplo de jogo de convexidade geodésica de intervalo fechado em um grafo G supondo que Alice comece rotulando o vértice 1, resultando na instância (a). Considere que L é o conjunto dos vértice já rotulados e ele é inicialmente vazio. Perceba que $V(G) = \{1,2,3,4,5,6\}$ e, na instância (a), $f_1(L) = f_2(L) = I_g(L) = \{1\}$.

Figura 3 – Representação gráfica do jogo de convexidade geodésica de intervalo fechado em um grafo



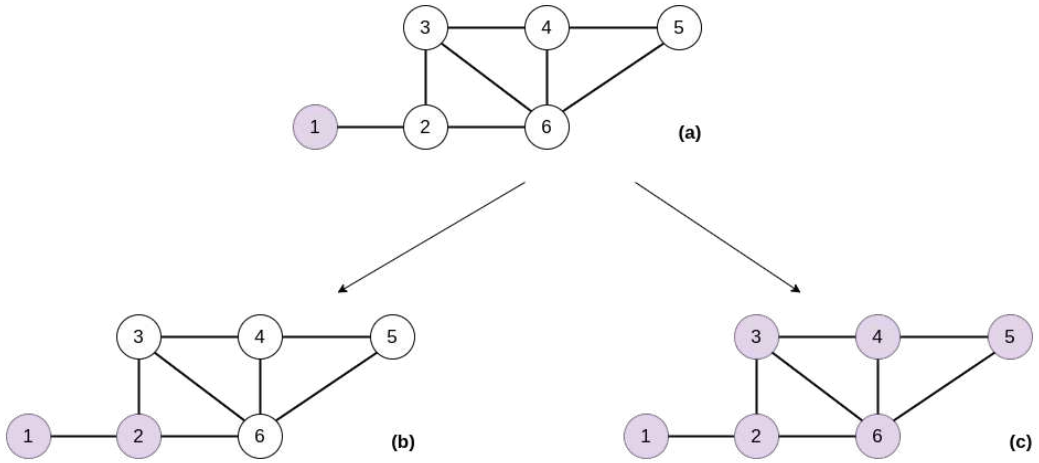
Fonte: elaborado pelo autor

Após o movimento de Alice, Bob poderá rotular o vértice 2, aplicando a função de intervalo geodésico apenas os vértices 1 e 2 estarão rotulados já que $f_1(L) = f_2(L) = I_g(L) = \{1,2\}$, o que resultaria na instância (b). De outro modo, Bob poderia rotular o vértice 5 e aplicando a função de intervalo, os vértices 1, 2, 6 e 5 seriam rotulados já que $L = \{1,5\}$ e os vértices 2 e 6 fazem parte do menor caminho entre os vértices de L. Logo, $f_1(L) = f_2(L) = I_g(L) = \{1,2,5,6\}$ resultando na instância (c). Desta forma, o jogo continua até que todos os vértices estejam rotulados, isto é, $f_2(L) = V(G)$.

Por outro lado, a Figura 4 mostra um exemplo de jogo de convexidade monofônica

de intervalo fechado no mesmo Grafo G supondo novamente que Alice comece rotulando o vértice 1, resultando na instância (a). De maneira análoga à Figura 3, L é o conjunto dos vértice já rotulados que inicialmente é vazio, $V(G) = \{1,2,3,4,5,6\}$ e, na instância (a), $f_1(L) = f_2(L) = I_g(L) = \{1\}$.

Figura 4 – Representação gráfica do jogo de convexidade monofônica de intervalo fechado em um grafo



Fonte: elaborado pelo autor

A partir da instância (a), Bob poderá rotular o vértice 2, nesse caso aplicando a função de intervalo monofônico apenas os vértices 1 e 2 estarão rotulados já que $f_1(L) = f_2(L) = I_m(L) = \{1,2\}$ que resulta na instância (b). Não obstante, outra possível jogada de Bob é rotular o vértice 5, nesse caso, aplicando a função de intervalo monofônico todos os vértices serão rotulados já que há o caminho monofônico que contém os vértices 1, 2, 3, 4 e 5 e o caminho monofônico que contém os vértices 1, 2, 6 e 5. Então, $f_1(L) = f_2(L) = I_g(L) = \{1,2,3,4,5,6\}$, resultando na instância (c) e o jogo acabará já que $f_2(L) = V(G)$. Bob vencerá na variante normal e perderá na variante misère nesse caso.

Em 1985, os jogos de intervalo geodésico foram solucionados (foi possível determinar, baseado na estrutura do grafo, qual será o jogador vitorioso) para ciclos, rodas e grafos bipartidos completos (BUCKLEY; HARARY, 1985b). Em 1988, em (NEcáSKOVá, 1988) foi demonstrado que o resultado de (BUCKLEY; HARARY, 1985b) para grafos roda estava incorreto e foram determinadas condições necessárias e suficientes para que o primeiro jogador fosse vitorioso em grafos roda . Em 2003, Haynes, Henning e Tiller (HAYNES *et al.*, 2003) obtiveram resultados para árvores e grafos multipartidos completos nas variantes normal e misère.

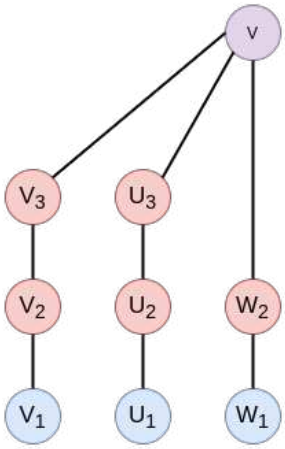
3.2 Jogos de convexidade partizan

A teoria dos jogos partizan apresentada na Seção 2.2 conduz a uma definição de uma versão partizan para os jogos de convexidade.

Definição 3.2. *Seja Ψ um jogo da Definição 3.1. Seja PARTIZAN Ψ o jogo obtido das seguintes modificações em Ψ: os vértices do grafo G são coloridos com A ou com B, Alice só pode rotular vértices da cor A e, de maneira análoga, Bob só pode rotular vértices da cor B.*

O exemplo na figura 5 mostra um jogo PARTIZAN CIG_G em um grafo G que começa com o vértice $v \in V(G)$ já rotulado. Além disso, a cor de Bob é vermelho e de Alice é azul.

Figura 5 – Representação gráfica do jogo partizan de convexidade geodésica de intervalo fechado em um grafo G cujo o vértice v já está rotulado



Fonte: elaborado pelo autor

Neste exemplo, Alice só poderá rotular os vértices v_1, u_1 e w_1. As jogadas possíveis de Bob só rotularão os vértices que estão no meio do caminho entre os vértices v_1, u_1 e w_1 e o vértice v já rotulado.

Tendo isso em vista, nesse trabalho, continuando a linha de pesquisa apresentada, foram obtidos resultados para os jogos partizan de convexidade geodésica e monofônica em grafos. Foi desenvolvido um algoritmo para solucionar o problema de decisão de jogos de convexidade geodésica fechada para árvores. Além disso, será apresentada a relação dos jogos de convexidade partizan em grafos com o jogo Red-Blue Hackenbush que, como já demonstrado anteriormente, pode ser resolvido em tempo polinomial. Estes resultados estão descritos na próxima seção.

4 RESULTADOS

Diante do exposto, a contribuição desse trabalho está focada em obter resultados com técnicas e teorias consolidadas dos jogos imparciais e partizan nos jogos de convexidade em grafos.

4.1 Aplicação da teoria de Sprague-Grundy nos jogos de intervalo fechado e envoltória fechado em árvores

Inicialmente, é necessário perceber que os jogos de convexidade geodésica de intervalo fechado (CIG_G) e os jogos de convexidade monofônica de intervalo fechado (CIG_M) em árvores são equivalentes, isto é, em qualquer instância de um jogo de convexidade de intervalo fechado (CIG_C) em uma árvore, o resultado da aplicação da função de intervalo monofônico nessa instância será equivalente ao resultado da aplicação da função de intervalo geodésico. Isso ocorre pois só há um caminho mínimo entre dois vértices de uma árvore e esse caminho também é o único caminho induzido entre os dois vértices já que, por definição, árvores são acíclicas e conexas. Tendo esse fato em vista, como os resultados são equivalentes para ambas as convexidades abordadas, nesta subseção trataremos dos jogos de convexidade de intervalo fechado de maneira geral.

Em árvores os movimentos no jogo de convexidade de intervalo fechado são equivalentes aos jogos de convexidade de envoltória fechado já que dado um conjunto S de vértices de uma árvore T, $I_C(S)$ é C-convexo e é o menor conjunto convexo que contém S. Portanto, os resultados apresentados do CIG_C são equivalentes aos resultados do CHG_C.

Além disso, outra observação necessária é que o CIG_C é um jogo combinatório imparcial já que é um jogo sequencial (Alice e Bob jogam alternadamente), de informação perfeita (ambos os jogadores possuem total conhecimento dos elementos do jogo e de todas as jogadas possíveis em qualquer instância) e ambos os jogadores possuem as mesmas possibilidades de jogada em qualquer momento do jogo. Portanto, é possível aplicarmos a teoria de Sprague-Grundy nas instâncias desses jogos e calcularmos um *nimber* e relacionarmos o CIG_C com o Nim para determinarmos se Alice possui uma estratégia vencedora. Primeiramente, apliquemos a teoria de Sprague-Grundy no CIG_C em caminhos e depois em árvores.

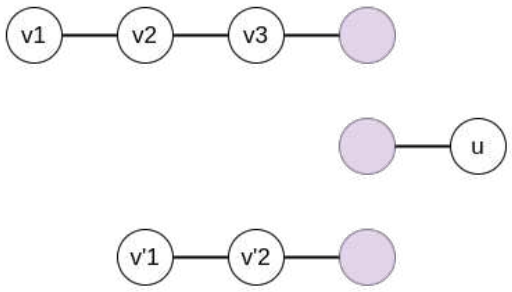
4.1.1 Jogos de convexidade de intervalo fechado em caminhos

Seja P um grafo que é um caminho e $V(P) = \{v_1, \ldots, v_n\}$. Em um jogo de convexidade fechada em P, considere que apenas v_1 foi rotulado. Todo os movimentos possíveis a partir desta instância podem rotular $1, \ldots, n-1$ vértices. Perceba que essa instância se assemelha a um jogo de Nim com apenas um monte de tamanho $n-1$, já que os únicos movimentos possíveis seriam retirar $1, \ldots, n-1$ objetos do monte.

Portanto, um caminho P de n vértices em um jogo de convexidade fechada, cujo o primeiro vértice do caminho já está rotulado terá *nimber* igual a $n-1$ visto que os movimentos possíveis a partir dessas instância terão valor $\{0, ..., n-2\}$ e o mínimo valor excludente desse conjunto é $mex\{0, ..., n-2\} = n-1$.

Na Figura 6, a floresta T terá $nimber = 3 \oplus 1 \oplus 2 = 0$. Isso significa, segundo a teoria de Sprague-Grundy, que o próximo jogador perderá nas variantes normal e misère.

Figura 6 – Representação gráfica do jogo de convexidade de intervalo fechado na floresta T



Fonte: elaborado pelo autor

4.1.2 Jogos de convexidade de intervalo fechado em árvores

Seja G uma árvore. Em um jogo de convexidade de intervalo fechado em G todas as jogadas possíveis rotularão um conjunto de vértices, exceto a primeira jogada, e produzirão conjuntos de vértices não rotulados disjuntos que ou serão caminhos ou serão árvores. Perceba também que em cada conjunto de vértices não rotulados apenas um vértice do conjunto é vizinho de um vértice já rotulado.

Considere que em G foi feito um movimento que rotulou um conjunto C de vértices. Para calcular o *nimber* da instância resultante é necessário, segundo a teoria de Sprague-Grundy, fazer a operação bitwise xor entre todos os $nimber's$ dos jogos independentes entre si que foram gerados após o conjunto C ser rotulado. Nos jogos de convexidade em árvores, esses jogos

independentes entre si são os conjuntos de vértices não rotulados.

Caso o conjunto de vértices não rotulado seja um caminho é possível calcular seu *nimber* pelo método descrito na subseção 4.1.1 já que exatamente um vértice desse caminho é vizinho de um vértice já rotulado. Contudo, se o conjunto de vértices não rotulados for uma árvore, segundo a teoria de Sprague-Grundy, é possível calcular seu *nimber* através do mínimo valor excludente (mex) do conjunto dos valores obtidos após um movimento nessa árvore.

Logo, é possível de maneira recursiva calcular o nimber de um CIG_C em uma árvore. Para demonstrar isso, é necessário primeiro definirmos e solucionarmos o jogo em uma versão simplificada.

Definição 4.1. *Dado um grafo G e um vértice $v \in V(G)$, seja o jogo simplificado de convexidade de intervalo fechado $SCIG_C$ de uma instância (G,v) um CIG_C no grafo G com exceção de que v já está rotulado no início do jogo. Isso significa que ao invés de estar vazio no começo do jogo, o conjunto L de vértices rotulados é tal que $L = \{v\}$. Analogamente, definimos as versões simplificadas para os outros jogos da Definição 3.1 e da Definição 3.2.*

Teorema 4.1.1. *O problema de decisão para as variantes normal e misère para o jogo de convexidade de intervalo fechado $SCIG_C$ e para o jogo de convexidade de envoltória fechado $SCHG_C$ é resolvido em tempo polinomial em árvores.*

Demonstração. Seja T uma árvore e o vértice $v \in V(T)$. Em um $SCIG_C$ na instância (T,v), um novo vértice w é rotulado, então os vértice no caminho entre v e w também serão rotulados no jogo.

Se v não tem vizinhos, então (T,v) tem $nimber = 0$ e Alice perde na variante normal e ganha na variante misère. Mas se v tiver vizinhos, seja u um vizinho de v. Dizemos que (T,v,u) é um jogo simplificado da instância $(T_{v,u},v)$ onde $T_{v,u}$ é uma subárvore de T contendo v obtida através da remoção de todas as arestas que incidem em v exceto vu. Note que v é uma folha de $T_{v,u}$.

Vejamos o cálculo do nimber de (T,v,u). Seja $u_1,...,u_k$ os vizinhos de u distintos de v. Assuma que o *nimber* de (T,u,u_i), denotado por h_i, em $SCIG_g$ é conhecido. Se Alice rotular u, então os jogos de $(T,u,u_1),...,(T,u,u_k)$ são independentes entre si, isto é, um movimento em uma instância não afeta outra. Pela teoria de Sprague-Grundy, a posição resultante tem nimber $h_1 \oplus \cdots \oplus h_k$, onde h_i é o nimber da posição resultante de (T,u,u_i).

Se Alice rotular um vértice w_i em (T,u,u_i), que é distindo de u, então u também será rotulado e novamente o jogo nas subárvores serão independetes entre si. Seja h'_i o nimber da posição resultante de (T,u,u_i). Então, pela teoria de Sprague-Grundy, a posição resultante de (T,v,u) após o movimento em w_1 tem nimber $h_1 \oplus ... \oplus h'_i \oplus ... \oplus h_k$. Além disso, h'_i pode ter qualquer valor em $\{0,1,...,h_i-1\}$.

Seja N o conjunto de todos os nimbers de $h'_1 \oplus ... \oplus h'_k$ onde $h'_i = h_i$ para todo $i = 1,...,k$ exceto que $h'_i \in \{0,...,h_i-1\}$. Portanto, o nimber de (T,v,u) é exatamente $\text{mex}(N)$, desde que N contenha todas as nimbers possíveis após um movimento em (T,v,u).

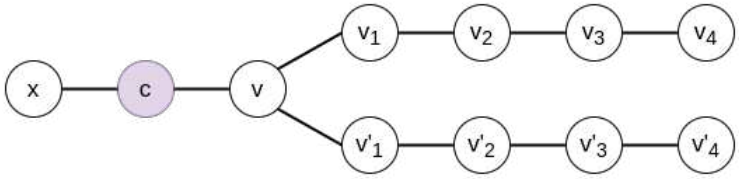
Esses argumentos conduzem ao Algoritmo 1 que é recursivo, de tempo polinomial, que calcula o *nimber* de (T,v,u) de uma árvore T enraizada em uma folha v com um vizinho u.

Agora, considere o jogo em toda árvore T e seja $v_1,...,v_k$ os vizinhos de v com $k \geq 2$. Pelos argumentos supracitados, é possível determinar o *nimber* l_i de (T,v,v_i) em $SCIG_C$. Então, como antes, temos que os jogos nas subárvores são independentes e o nimber de (T,v) é exatamente $h = l_1 \oplus \cdots \oplus l_k$. Logo, é possível calcular o nimber de (T,v) de uma árvore T enraizada em v. Isso leva ao Algoritmo 2 que recebe como entrada uma árvore T enraizada em v e calcula o *nimber* h da instância (T,v).

Por fim, fazendo uma associação direta ao jogo Nim, temos que Alice possui uma estratégia vencedora na variante normal se e somente se $h > 0$. Além disso, Alice possui uma estratégia vencedora na variante misère se e somente se $h > 0$ e há pelo menos uma subárvore com mais do quê um vértice não rotulado, ou $h = 0$ e todas as subárvores possuem exatamente um vértice não rotulado. □

No exemplo da árvore T da Figura 7, assumindo que Alice rotulou o vértice c no primeiro movimento no CIG_C em T, temos a posição (T,c) de um $SCIG_C$ onde Bob é o primeiro a jogar.

Figura 7 – Representação gráfica do jogo simplificado de convexidade de intervalo fechado na árvore T



Fonte: elaborado pelo autor

Seja T' obtido quando x é rotulado. Inicialmente vamos calcular o *nimber* da

Algoritmo 1: $SCIG_C$-leaf(T, v, u)

if *nimber[u] já foi calculado* **then**
  return nimber[u];
end
if *u tem grau 1 em T* **then**
  nimber[u] $\leftarrow 1$;
  return 1;
end
let $u_1, \ldots, u_k$ os vizinhos de u distintos de v em T
for $i = 1, \ldots, k$ **do**
  $SCIG_C$-leaf(T, u, u_i);
  let $h_i \leftarrow nimber[u_i]$;
end
if $k > 1$ **then**
  $N \leftarrow \{h_1 \oplus \cdots \oplus h_k\}$;
  for $i = 1, \ldots, k$ **do**
    for $h_i' = 0, \ldots, h_i - 1$ **do**
      N $\leftarrow N\{\mathrm{h}_1 \oplus \cdots \oplus h_i' \oplus \cdots \oplus h_k\}$;
    end
  end
  $nimber[u] \leftarrow mex(N)$
  return $nimber[u]$;
end
else
  $nimber[u] \leftarrow \mathrm{h}_1 + 1$;
  return $nimber[u]$;
end

instância (T', v). Se Bob rotular v, então o conjunto de vértices não rotulados disjuntos serão dois caminhos com quatro vértices não rotulados, logo o *nimber* dessa instância é $4 \oplus 4 = 0$.

Se Bob rotular outro vértice de T', então ainda haverá dois caminhos de vértices não rotulados disjuntos. Um deles terá $k \leq 3$ vértices não rotulados e o outro terá exatamente 4 vértices não rotulados. Associando a um jogo de Nim, haverá um monte de tamanho $k \leq 3$ e outro monte de tamanho 4. Então o *nimber* de (T', c) é $mex\{4 \oplus 4, 4 \oplus 3, 4 \oplus 2, 4 \oplus 1, 4 \oplus 0\} = 1$.

Por fim, considerando novamente o vértice x, o *nimber* de (T', c) é $1 \oplus 1 = 0$.

Algoritmo 2: $SCIG_C$-tree(T, v)

```
let h ← 0;
let v_1, ..., v_k os vizinhos de v em T;
Demonstração. Seja T uma árvore.
for i = 1, ..., k do
    h ← h ⊕ SCIG_C-leaf(T, v, v_i)
end
if normal-play then
    if h > 0 then
        Alice wins;
    end
    else
        Bob wins;
    end
end
if misère-play then
    if todas as subárvores de v possuem exatamente um vértice não rotulado then
        if h = 0 then
            Alice wins;
        end
        else
            Bob wins;
        end
    end
    else
        if h > 0 then
            Alice wins;
        end
        else
            Bob wins;
        end
    end
end
```

Portanto, o segundo jogador (que é Alice nesse caso) terá uma estratégia vencedora para as variantes normal e misère.

Corolário 4.1.1. *O problema de decisão das variante normal e misère para o jogo de convexidade de intervalo fechado CIG_C e para o jogo de convexidade de envoltória fechado $SCHG_C$ é resolvido em tempo polinomial em árvores.*

Demonstração. Seja T uma árvore. Alice possui uma estratégia vencedora no CIG_C em T se e somente se existe um vértice $v \in V(T)$ cujo o segundo jogador do $SCIG_C$ na instância (T,v) possui uma estratégia vencedora e isto foi demonstrado no Teorema 4.1.1. □

4.2 Cálculo dos valores de instâncias do jogo Red-Blue Hackenbush em árvores

Como dito anteriormente, foram desenvolvidos algoritmos para calcular valores de instância do jogo Red-Blue Hackenbush em árvores. Nesta seção serão descritos os algoritmos que foram implementados para esse cálculo. Além disso, os códigos-fontes da implementação deste cálculo estão no Apêndice A.

Inicialmente, apresentemos o Algoritmo 3 para calcular o valor do jogo Red-Blue Hackenbush em caminhos. Ele recebe como entrada uma string red-blue hackenbush S (assim como foi definida na Seção 2.2.1.1) que é um caminho e retorna seu valor. O método utilizado foi o descrito no final da Seção 2.2.1.

Primeiramente, serão contadas todas as arestas da mesma cor em um caminho contínuo até o chão, o valor atribuído a esse caminho continuo é a quantidade de arestas que ele contém. Caso as arestas sejam azuis esse valor é positivo, caso sejam vermelhas esse valor é negativo. O que se segue é que cada outra aresta do caminho terá valor igual a metade do valor computado anteriormente, exceto pela primeira aresta que terá valor $\frac{1}{2}$, e, novamente, esse valor será positivo caso a aresta seja azul ou será negativo caso a aresta seja vermelha. Por fim, esses valores atribuídos serão somados e o resultado será retornado como o valor do caminho.

Diante disso, para calcular as instâncias que são árvores será necessário calcular o número diádico simples entre valores das posições. Portanto, o Algoritmo 4 recebe como entrada dois números x e y e retorna o número diádico simples entre eles.

Nesse sentido, a primeira condição do Algoritmo 4 é checar se existe algum inteiro entre x e z, caso haja, então esse valor é retornado como o número diádico simples. Caso contrário, x e y devem ser manipulados para terem o mesmo denominador. Seja D o denominador de x e y, seja A o numerador de x e B o numerador de y. O número diádico pode ser obtido da seguinte maneira: Se $A = B - 1$, então o número diádico será $(A+B)/2 * D$, caso contrário

Algoritmo 3: $RBH - Path(S)$

```
let V ← 0;
let cor-anterior ← " ";
for Cada caractere s da string S do
    if cor-anterior = " " OU s = cor-anterior then
        cor-anterior ←s;
        if s = B then
            V ← V + 1;
        end
        else
            V ← V − 1;
        end
    end
    else
        cor-anterior ← N;
        if s = B then
            V ← V + V/2;
        end
        else
            V ← V − V/2;
        end
    end
end
return V;
```

considere $k = \log_2(B - A - 1)$, o número diádico será $[((A/2^k) + 1) * 2^k]/D$.

Por fim, o algortimo 5 recebe como entrada uma string S que é uma árvore Red-Blue Hackenbush e um hash H para guardar os valores já calculados das subposições da árvore da string S e retorna o valor do jogo na instância S.

Utilizando técnicas de programação dinâmica, o hash H no Algoritmo 5 é utilizado para guardar o valor de subposições de S para otimização no que tange ao tempo de execução do algoritmo. Então, primeiramente é checado se a instância S já foi calculada, caso tenha sido, então basta retornar o valor já computado. Contudo, se a instância ainda não foi calculada e é um caminho, basta chamar o Algoritmo 3 para calcular aquela instância. Esse valor será armazenado em H e retornado pela função. Finalmente, caso a instância seja uma árvore, os valores das subárvores serão calculadas recursivamente, serão armazenados nos conjuntos *Red* e *Blue* e, por

Algoritmo 4: $NDS(x,y)$

```
if Se existe um inteiro z entre x e y then
    return z;
end
Manipular x e y para ter o mesmo denominador;
let D ← denominador de x e y;
if x < y then
    let A ← numerador de x;
    let B ← numerador de y;
end
else
    let A ← numerador de y;
    let B ← numerador de x;
end
if A = 0 E B = 0 then
    return 0;
end
if A = B-1 then
    return (A+B) / (2*D);
end
else
    let N ← B-A-1;
    let K ← log2 N;
    let C ← A/2^k;
    return (C+1 * 2^k)/D;
end
```

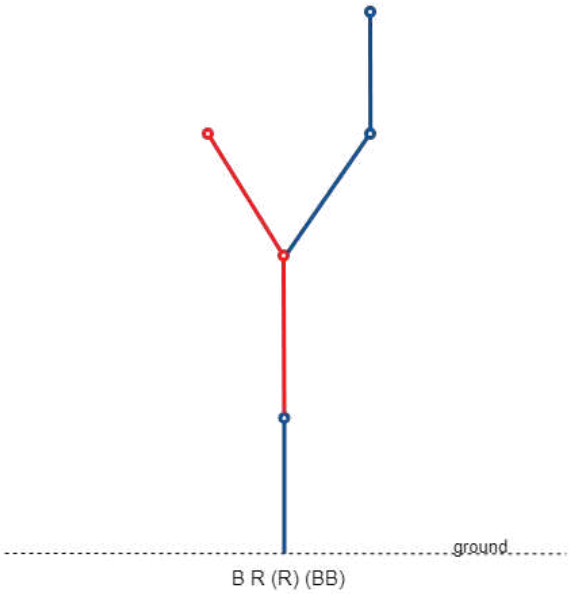
fim, será retornado o valor diádico simples entre o menor valor do conjunto *Red* e o maior valor do conjunto *Blue*.

Diante disso, tome como exemplo a instância do jogo Red-Blue Hackenbush representada na Figura 8 cuja string é "B R (R) (BB)". Logo, a chamada do algoritmo para essa instância será $RBH-Tree((BR(R)(BB)),H)$ e retornará o valor $\frac{3}{4}$. Isso significa que o Left terá vantagem nessa instância e será vitorioso caso faça movimentos de maneira ótima.

Algoritmo 5: $RBH - Tree(S,H)$

if *Existe um valor v em H que corresponde à árvore S* **then**
  return v;
end
if *S é um caminho* **then**
  let $v \leftarrow$RBH-Path(S);
  Armazenar v em H;
  return v;
end
let *Blue* o conjunto dos valores $RBH - Tree(S',H)$ sendo S' a substring após a remoção de alguma aresta azul em S;
let *Red* o conjunto dos valores $RBH - Tree(S',H)$ sendo S' a substring após a remoção de alguma aresta vemelha em S;
let $r \leftarrow NDS(min(Red), max(Blue))$;
Armazenar r em H;
return r;

Figura 8 – Desenho da instância B R (R) (BB) no jogo Red-Blue Hackenbush



Fonte: elaborado pelo autor

4.3 Relação entre o jogo Red-Blue Hackenbush e o jogo PARTIZAN $SCHG_G$ e $SCIC_G$

O lema 4.3.1 a seguir mostra uma forte relação entre o jogo Red-Blue Hackenbush e os jogos partizan de convexidade geodésica simplificados de intervalo fechado e de envoltória fechada em grafos. Uma estrela subdividida é uma árvore obtida após a subdivisão de cada aresta

da estrela $K_{1,p}$ quantas vezes forem necessárias.

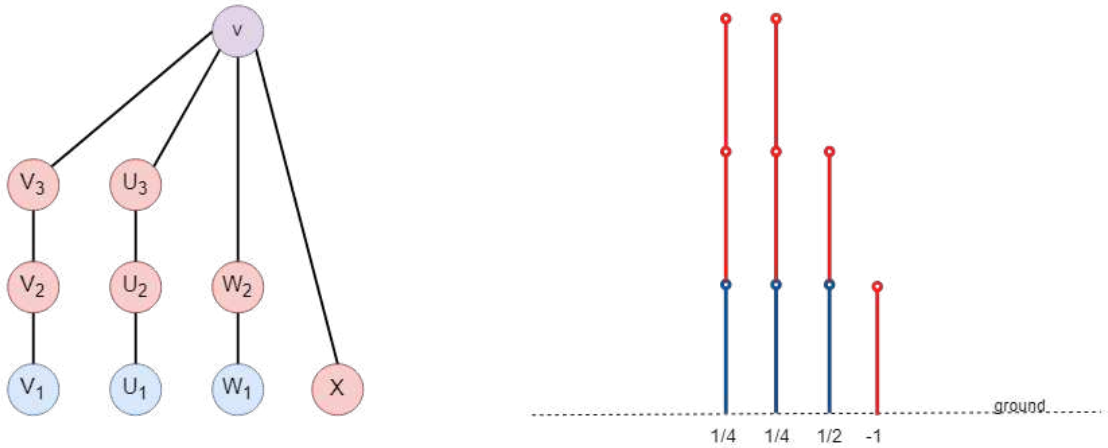
Lema 4.3.1. *Seja G uma estrela subdividida com o centro c cujo os vértices (exceto c) são coloridos por A ou B. Então, o valor do jogo PARTIZAN $SCHG_G$ e $SCIG_G$ em (G,c) é igual ao valor do jogo Red-Blue Hackenbush formado pela união de strings disjuntas, onde cada string representa o caminho maximal em G começando de c tal que o k-ésimo primeiro segmento é azul se e somente se o k-ésimo vértice do caminho é colorido por A.*

Demonstração. Em uma string Hackenbush, se o k-ésimo primeiro segmento é cortado, todos os segmentos do k-ésimo ao último também serão cortados. No jogo PARTIZAN $SCHG_G$ e $SCIG_G$ em (P_{n+1}, v_{n+1}) onde o caminho P_n possui os vértices $v_1, \ldots, v_{n+1}$, se v_k é rotulado, então os vértices no caminho $v_k, ..., v_{n+1}$ não poderão mais ser rotulados nas próximas jogadas.

Isso demonstra a equivalência entre strings em Red-Blue Hackenbush e caminhos no PARTIZAN $SCIG_G$ e $SCHG_G$. Perceba que na estrela subdividida G um movimento em algum ramo de G não afeta os demais ramos. Então, o jogo em G é uma união disjunta dos jogos dos ramos. □

A Figura 9 é um exemplo do PARTIZAN $SCIG_G$ na estrela subdividida G onde o vértice v já está rotulado no início do jogo. Este jogo é equivalente a instância Red-Blue Hackenbush com valor $\frac{1}{4}+\frac{1}{4}+\frac{1}{2}-1=0$. Logo, o primeiro jogador irá perder, independentemente se o primeiro a jogar é Alice ou Bob.

Figura 9 – Representação gráfica do PARTIZAN $SCIG_G$ em (G,v) e o desenho da sua instância equivalente no jogo Red-Blue Hackenbush

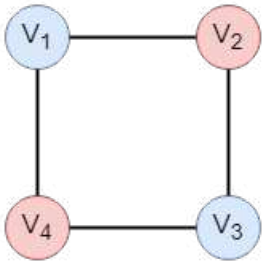


Fonte: elaborado pelo autor

A relação entre o PARTIZAN $SCIG_G$ e o Red-Blue Hackenbush é válida para estrelas

subdivididas, mas não é válida para qualquer árvore. Contudo, assim como as instâncias do Red-Blue Hackenbush, uma posição no PARTIZAN CIG_G pode ser *"fuzzy"*, isto é, o primeiro jogador sempre ganhará. Por exemplo, na Figura 10 em que um ciclo C_4 é tal que $V(C_4) = \{v_1, v_2, v_3, v_4\}$ cujo os vértices v_1 e v_3 tem cor A e os vértices v_2 e v_4 tem cor B, o primeiro a jogar ganhará em todos os jogos PARTIZAN IG_G, PARTIZAN HG_G, PARTIZAN CIG_G e PARTIZAN CHG_G.

Figura 10 – Representação gráfica do jogo de convexidade partizan em um C_4 cujo os vértices v_1 e v_3 tem cor A (azul) e os vértices v_2 e v_4 tem cor B (vermelho)



Fonte: elaborado pelo autor

Além disso, o Teorema 4.3.1 demonstra que as posições dos jogos PARTIZAN CIG_G e PARTIZAN CHG_G podem ser resolvidos em tempo polinomial em árvores.

Teorema 4.3.1. *Em árvores, os jogos PARTIZAN CIG_G e PARTIZAN CHG_G são solucionados em tempo polinomial.*

Demonstração. Seja T uma árvore. Primeiramente considere o jogo PARTIZAN $SCIG_G$ em (T, v) onde $v \in V(T)$. Se $V(T) = \{v\}$, então o valor da instância é zero, ou seja, o primeiro a jogar perde. Então, seja u um vizinho de v. Escrevemos (T, v, u) para a instância $(T_{v,u}, v)$ de um jogo PARTIZAN $SCIG_G$, onde $T_{v,u}$ é uma subárvore de T contendo v que é obtida após a remoção de todas as arestas incidentes em v exceto vu. Dessa forma, v é uma folha de $T_{v,u}$. Então, foquemos inicialmente em resolver a instância no jogo (T, v, u).

Se u não possui vizinhos, então o valor da instância será $\{0|\cdot\} = 1$ caso u tenha cor A ou será $\{\cdot|0\} = -1$ caso u tenha cor B. Então, considere que $u_1, \ldots, u_k$ são os vizinhos de u distindo de v. Assuma, por indução, que o os valores $\{A_i|B_i\}$ de (T, u, u_i) no PARTIZAN $SCIG_G$ são conhecidos e que são positivos caso sejam instâncias vencedoras para Alice ou são negativos caso sejam instâncias vencedoras para Bob. Note que $A_i < \{A_i|B_i\} < B_i$ para todo i quando A_i e B_i não são vazios.

Tendo isso em vista, vamos determinar o maior valor A_{vu} após um movimento de Alice em (T, v, u). Se nenhum vértices de (T, v, u) tem cor A,então Alice não tem nenhuma

jogada e A_{vu} será vazio. Se u possui cor A, então a melhor opção para Alice é rotular o vértice u, já que $A_i|B_i > A_i$ para todo i e, consequentemente, o valor $A_{vu} = \sum_{i=1}^{k}\{A_i|B_i\}$, tendo em vista que o jogo nas subárvores são independentes após esse movimento. De outro modo, considere que u tem cor B, então Alice deve rotular um vértice $w_j \neq u$ em uma subárvore (T, u, u_j) e nesse caso o vértice u também será rotulado, e consequentemente

$$A_{vu} = \max_{j=i}^{k}\left\{A_j - \{A_j|B_j\} + \sum_{i=1}^{k}\{A_i|B_i\} : A_j \textit{ não é vazio}\right\}. \tag{4.1}$$

De maneira análoga, vamos determinar o menor B_{vu} após um movimento de Bob em (T, v, u). Se nenhum vértice de (T, v, u) tem cor B, então Bob não tem nenhuma jogada e B_{vu} será vazio. Se u possui cor B, então Bob rotula u, já que $\{A_i|B_i\} < B_i$ para todo i e o valor $B_{vu} = \sum_{i=1}^{k}\{A_i|B_i\}$. De outro modo, assumindo que u tem cor A, Bob deve rotular algum vértice de uma subárvore (T, u, u_j), dessa forma u também será rotulado e consequentemente

$$B_{vu} = \min_{j=i}^{k}\left\{B_j - \{A_j|B_j\} + \sum_{i=1}^{k}\{A_i|B_i\} : B_j \textit{ não é vazio}\right\}. \tag{4.2}$$

A partir disso, é possível resolver o PARTIZAN $SCIG_G$ em (T, v, u) calculando $\{A_{vu}|B_{vu}\}$. Em relação ao PARTIZAN $SCIG_G$ em (T, v), note que os jogos nas subárvores de v são independentes, logo é possível calcular o valor de $\{A_i|B_i\} = \sum_{u\in N(v)}\{A_{vu}|B_{vu}\}$.

Finalmente, considere o PARTIZAN CIG_G em T. Vamos calcular o melhor valor A_T para Alice e o melhor valor B_T para Bob. Se nenhum vértice de T tem cor A, então A_T é vazio. De maneira análoga, se nenhum vértice de T tem cor B, então B_T é vazio. Caso contrário, $A_T = \max_{v\in V(T)}\{A_v|B_v\}$ e $B_T = \min_{v\in V(T)}\{A_v|B_v\}$. Isso leva a um algoritmo recursivo de tempo polinomial. □

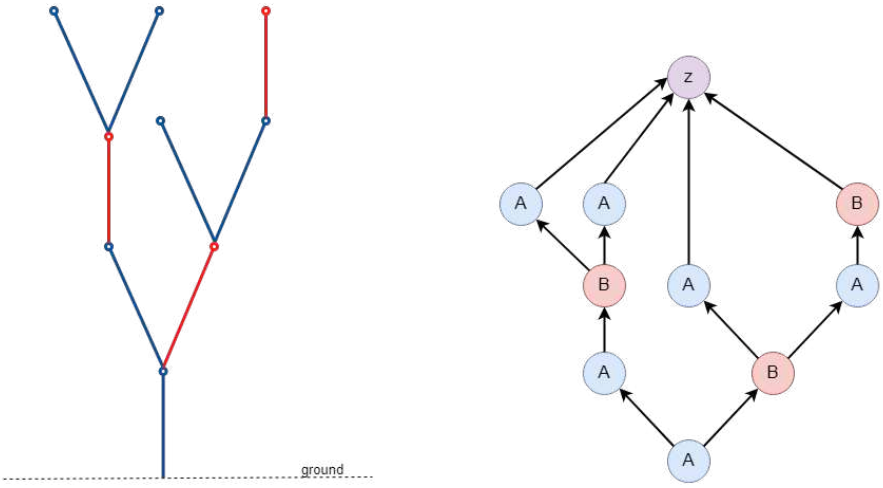
Além disso, o lema 4.3.2 demonstra que os jogos Red-Blue Hackenbush em árvores são equivalentes aos jogos PARTIZAN $SCIG_M$ em grafos direcionados e acíclicos.

Lema 4.3.2. *O jogo Red-Blue Hackenbush em árvores é equivalente ao jogo PARTIZAN $SCIG_M$ em DAGs (grafos acíclicos direcionados) que possui uma fonte, um sumidouro, já está rotulado no início do jogo, e todos os outros vértices têm grau 1.*

Demonstração. Dado um jogo Red-Blue Hackenbush em uma árvore T, é possível obter um grafo orientado $\vec{G}$ a partir do que se segue. Para toda aresta x de T, é criado um vértice

v_x em $\vec{G}$. Se x e y são arestas adjacentes em T e x está abaixo de y, então é criado um arco $v_x v_y$ em $\vec{G}$. Além disso, é criado um vértice z em $\vec{G}$ e é criado todos os arcos $v_x z$ quando x não possui uma aresta adjacente acima dela em T e considere que z já está rotulado desde o início do jogo. Finalmente, v_x terá cor A em $\vec{G}$ se e somente se a aresta x tem cor azul em T. A Figura 11 mostra um exemplo. Note que $\vec{G}$ é um *DAG* e há exatamente um sumidouro (o vértice z) e uma fonte (o vértice associado a aresta de T conectada ao chão). Além disso, todo vértice de $\vec{G}$ tem grau 1, exceto a fonte e o sumidouro. Também é possível perceber que quando o vértice v_x de $\vec{G}$ é rotulado no jogo PARTIZAN $SCIG_M$ é impossível nas próximas jogadas rotular os vértices associados as arestas de T que estão acima de x, oque é equivalente a cortar a aresta x no jogo Red-Blue Hackenbush.

Figura 11 – Desenho de uma instância de jogo Red-Blue Hackenbush em árvore e a Representação gráfica de sua equivalência no jogo de convexidade PARTIZAN $SCIG_M$ em um grafo $\vec{G}$



Fonte: elaborado pelo autor

Finalmente, seja $\vec{G}$ um *DAG* (grafo acíclico e direcionado) com uma fonte s, uma sumidouro z e todos os demais vértices com grau 1, onde o sumidouro já está rotulado no começo do jogo PARTIZAN $SCIG_M$. É possível obter uma um jogo Red-Blue Hackenbush em uma árvore T pelo oque se segue. Para todo vértice $v \neq z$ de $\vec{G}$, é criado um vértice v em T. Para cada arco uv de $\vec{G}$ com $v \neq z$, é criado uma aresta uv em T. Além disso, é criado um vértice g em T que é o chão em um jogo Red-Blue Hackenbush e são criadas as arestas gs, onde s é o vértice de T que corresponde a fonte s de $\vec{G}$. Por fim, a cor da aresta uv de T será azul se e somente se o vértice v tem cor A em $\vec{G}$.

Note que T é uma árvore Red-Blue Hackenbush. Assim como, também é possível

perceber que cortando uma aresta uv de T se torna impossível nas próximas jogadas cortar arestas acima de uv em T e isto é equivalente ao PARTIZAN $SCIG_M$ em $\vec{G}$, cujo sumidouro z já está rotulado. □

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho buscou investigar a resolução de jogos de convexidade aplicados à algumas classes de grafos, por meio da utilização dos métodos e de soluções aplicados a jogos clássicos da literatura. Diante disso, é possível concluir que o principal objetivo deste trabalho foi alcançado, tendo em vista que as principais contribuições foram:

a) Solução do problema de decisão em jogos de convexidade de intervalo fechado em caminhos, aplicando a teoria de Sprague-Grundy;
b) Construção de um algoritmo recursivo polinomial para solucionar o problema de decisão em jogos de convexidade de intervalo fechado em árvores tanto nas versões simplificadas, quanto nas demais versões;
c) Baseado na teoria de Conway para jogos partizan, foi desenvolvido um algoritmo polinomial para calcular o valor de caminhos no jogo Red-Blue Hackenbush;
d) Foi desenvolvido um algoritmo que calcula o número diádico simples entre dois números;
e) Fundamentado nas duas últimas contribuições, foi construído um algoritmo recursivo polinomial para calcular valores de árvores no jogo Red-Blue Hackenbush;
f) Demonstrado que jogos partizan simplificados de convexidade geodésica de intervalo fechado em grafos estrelas subdivididos são equivalentes aos jogos Red-Blue Hackenbush em uniões de strings disjuntas;
g) Provado que jogos partizan de convexidade geodésica não são fuzzy e que são solucionados em tempo polinomial;
h) Demonstrado que o jogo Red-Blue Hackenbush em árvores é equivalente ao jogo partizan simplificado de convexidade monofônica de intervalo fechado em grafos acíclicos direcionados;

Em suma, para os jogos de convexidade imparciais é possível utilizar a teoria de Sprague-Grundy para determinar os nimbers de suas instâncias e assim gerar estratégias vencedoras. Além disso, para os jogos de convexidade partizan é possível utilizar as teorias de Conway com números surreais para atribuir valor as posições desses jogos e assim, também, resolver seus problemas de decisão.

Contudo, os jogos de convexidade imparciais em algumas classes de grafos ainda não foram solucionadas, a saber: grades e cografos, por exemplo. Logo, um possível trabalho futuro é analisar os jogos de convexidade imparciais em grafos que contém ciclos.

REFERÊNCIAS

BERLEKAMP, E. R.; CONWAY, J. H.; GUY, R. K. **Winning Ways for Your Mathematical Plays**. [*S. l.*]: Academic Press, 1982. v. 1. ISBN 9780429945595.

BOUTON, C. L. Nim, a game with a complete mathematical theory. **Annals of Mathematics**, v. 3, n. 1/4, p. 35–39, 1901.

BUCKLEY, F.; HARARY, F. Closed geodetic games for graphs. **Congressus Numerantium**, v. 47, p. 131–138, 1985. Proceedings of the 16th Southeastern Conference on Combinatorics, Graph Theory and Computing.

BUCKLEY, F.; HARARY, F. Geodetic games for graphs. **Quaestiones Mathematicae**, v. 8, p. 321–334, 1985.

CONWAY, J. H. **On numbers and games**. London: Academic Press, 1976. (L.M.S. monographs ; no. 6). ISBN 0121863506.

DIMAND, R. W. D. M.-A. **The History Of Game Theory, Volume 1: From the Beginnings to 1945**. [*S. l.*]: Routledge, 1996. v. 1. (Routledge Studies in the History of Economics, v. 1). ISBN 9780415072571.

DOURADO, M. C.; PROTTI, F.; SZWARCFITER, J. L. Complexity results related to monophonic convexity. **Discrete Applied Mathematics**, Elsevier, v. 158, n. 12, p. 1268–1274, 2010.

DUCHET, P. Convexity in combinatorial structures. In: **Proc. 14th Winter School on Abstract Analysis**. [*S. l.*]: Circolo Matematico di Palermo, 1987. p. [261]–293.

DUCHET, P. Convex sets in graphs ii, minimal path convexity. **Journal of Combinatorial Theory, Series B**, v. 44, p. 307–316, 1988.

ERDőS, P.; FRIED, E.; HAJNAL, A.; MILNER, E. Some remarks on simple tournaments. **Algebra universalis**, v. 2, p. 238–245, 1972.

EVERETT, M. G.; SEIDMAN, S. B. The hull number of a graph. **Discrete Mathematics**, Elsevier, v. 57, n. 3, p. 217–223, 1985.

FARBER, M.; JAMISON, R. E. Convexity in graphs and hypergraphs. **SIAM Journal on Algebraic Discrete Methods**, SIAM, v. 7, n. 3, p. 433–444, 1986.

FARBER, M.; JAMISON, R. E. On local convexity in graphs. **Discrete Mathematics**, Elsevier, v. 66, n. 3, p. 231–247, 1987.

GRUNDY, P. M. Mathematics and games. **Eureka**, v. 2, p. 6–8, 1939.

HARARY, F. Convexity in graphs: Achievement and avoidance games. In: **Annals of Discrete Mathematics (20)**. [*S. l.*]: North-Holland, 1984. v. 87, p. 323.

HARARY, F.; NIEMINEM, J. Convexity in graphs. **Journal of Differential Geometry**, v. 16, n. 1, p. 185–190, 1981.

HAYNES, T.; HENNING, M.; TILLER, C. Geodetic achievement and avoidance games for graphs. **Quaestiones Mathematicae**, v. 26, p. 389–397, 2003.

MYERSON, R. B. **Game Theory: Analysis of Conflict**. [*S. l.*]: Harvard University Press, 1997. ISBN 0674341163.

NEUMANN, J. v. Zur theorie der gesellschaftsspiele. **Mathematische Annalen**, v. 100, p. 295–320, 1928. Disponível em: http://eudml.org/doc/159291.

NEUMANN, J. von; MORGENSTERN, O. **Theory of games and economic behavior**. [*S. l.*]: Princeton University Press, 1947. ISBN 9780691130613.

NEcáSKOVá, M. A note on the achievement geodetic games. **Quaestiones Mathematicae**, v. 12, p. 115–119, 1988.

SMITH, J. M. Game theory and the evolution of behaviour. **Proceedings of the Royal Society of London. Series B, Biological Sciences**, The Royal Society, v. 205, n. 1161, p. 475–488, 1979. ISSN 00804649. Disponível em: http://www.jstor.org/stable/77441.

SMITH, J. M. **Evolution and the Theory of Games**. [*S. l.*]: Cambridge University Press, 1982. ISBN 9780521288842.

SPRAGUE, R. Über mathematische Kampfspiele. **Tôhoku Mathematical Journal**, v. 41, p. 438–444, 1936. ISSN 0040-8735.

ZERMELO, E. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In: **Proc. 5th Int. Congress of Mathematicians**. [*S. l.*: *s. n.*], 1913. p. 501–504.

APÊNDICE A – CÓDIGOS-FONTES UTILIZADOS PARA O CÁLCULO DOS VALORES DE POSIÇÕES EM ÁRVORES NO RED-BLUE HACKENBUSH

Código-fonte 1 – Classe Fracao

```
1  from math import gcd,lcm
2
3  #Classe inspirada em fraction
4  #Operacoes mais simples e nao simplifica a fracao sempre
       que possivel
5  #Necessario para deixar os valores com o mesmo denomidador
       para o calculo do numero diadico simples
6  class Fracao:
7
8    #inicializacao da fracao
9    def __init__(self, x, y = None):
10
11     if y == None:
12
13       if type(x) is int:
14         self.numerador = x
15         self.denominador = 1
16
17       elif type(x) is float:
18         self.numerador, self.denominador = x.
               as_integer_ratio()
19
20       elif isinstance(x, Fracao) or type(x) == Fracao:
21         self.numerador = x.numerador
22         self.denominador = x.denominador
23
24     else:
25
```

```
26        if type(x) is int is type(y):
27          self.numerador = x
28          self.denominador = y
29
30        elif isinstance(x, float) or isinstance(y,float):
31          x_n, x_d = x.as_integer_ratio()
32          y_n, y_d = y.as_integer_ratio()
33
34          self.numerador = x_n * y_d
35          self.denominador = x_d * y_n
36
37
38    #Metodos da classe
39    #Operacoes usando sobrecarga de operadores
40
41    #Definido o operador *
42    def __mul__(self, x):
43      return(Fracao(self.numerador * x, self.denominador))
44
45    #Definindo o operador <
46    def __lt__(self, x):
47      if type(x) is int or type(x) is float:
48        if (self.numerador/self.denominador) < x:
49          return True
50        return False
51
52      elif isinstance(x, Fracao):
53        if (self.numerador/self.denominador) < (x.numerador/x
             .denominador):
54          return True
55        return False
56
```

```
57    #Definindo o operador >
58    def __gt__(self, x):
59      if type(x) is int or type(x) is float:
60        if (self.numerador/self.denominador) > x:
61          return True
62        return False
63
64      elif isinstance(x, Fracao):
65        if (self.numerador/self.denominador) > (x.numerador/x
              .denominador):
66          return True
67        return False
68
69    #Definindo a funcao abs()
70    #Retornando o valor absoluto de uma fracao
71    def __abs__(self):
72      valor = self.numerador / self.denominador
73
74      if valor < 0:
75        return valor*-1
76      else:
77        return valor
78
79    #Definindo o operador -
80    def __sub__(self, x):
81        g = lcm(self.denominador, x.denominador)
82
83        termo1 = (g / self.denominador) * self.numerador
84        termo2 = (g / x.denominador) * x.numerador
85
86        return (Fracao(int(termo1 - termo2), g))
87
```

```
88  #Definindo o operador - no caso da fracao estar a direita
        do operador
89  def __rsub__(self, x):
90    return self - Fracao(x)
91
92  #Definindo o operador +
93  def __add__(self, x):
94      if type(x) is int or type(x) is float:
95        return self + Fracao(x)
96      else:
97        g = lcm(self.denominador, x.denominador)
98
99        termo1 = (g / self.denominador) * self.numerador
100       termo2 = (g / x.denominador) * x.numerador
101
102       return (Fracao(int(termo1 + termo2), g))
103
104 #Definindo a funcao int()
105 #Retorna o valor inteiro da fracao
106 def __int__(self):
107     return int(self.numerador/self.denominador)
108
109 #Normalizando a fracao
110 def normalizar(self):
111   g = gcd(self.numerador, self.denominador)
112   if self.denominador < 0:
113     g = -g
114
115   return Fracao((self.numerador // g), (self.denominador
        // g))
116
117 #Definindo a funca str()
```

```
118   #Retorna a fracao como uma string
119   def __str__(self):
120     if self.denominador != 1:
121
122       if self.numerador > self.denominador:
123         resto = Fracao(self.numerador/self.denominador -
              self.numerador//self.denominador)
124         return str(self.numerador//self.denominador) + " +
              " + str(resto.numerador) + str("/") + str(resto.
              denominador)
125
126       else:
127         return str(self.numerador) +  '/' +  str(self.
              denominador)
128
129     else:
130       return str(self.numerador)
```

Código-fonte 2 – Cálculo do número diádico simples

```
1  from math import log
2
3  #Calcula o numero diadico simples entre x e y
4  def nds(x,y):
5
6    #Checando se existe um valor inteiro entre x e y
7    if abs(x) < abs(y):
8      maior = y
9      menor = x
10   else:
11     maior = x
12     menor = y
```

```
13
14  if Fracao(menor).denominador == 1 and abs(menor - maior)
       > 1:
15
16    if menor > 0 and maior < 0 or menor < 0 and maior < 0:
17      return Fracao(menor - 1)
18
19    elif menor < 0 and maior > 0 or menor > 0 and maior >
         0:
20      return Fracao(menor + 1)
21
22  elif Fracao(menor).denominador != 1 and abs(menor - maior
       ) >= 1:
23    if menor > 0 and maior < 0 or menor < 0 and maior < 0:
24      return Fracao(int(menor) - 1)
25
26    elif menor < 0 and maior > 0 or menor > 0 and maior >
         0:
27      return Fracao(int(menor) + 1)
28
29
30  #Caso nao tenha valor inteiro devo encontrar o numero
       diadico simples
31  else:
32
33    x_fracao = Fracao(x)
34    y_fracao = Fracao(y)
35
36    #Devo deixar ambos com o mesmo denominador
37    if x_fracao.denominador < y_fracao.denominador:
38      mult = (abs(y_fracao.denominador)/abs(x_fracao.
           denominador))
```

```
39        x_fracao = Fracao(x_fracao.numerador * mult, x_fracao
              .denominador * mult)
40
41
42      else:
43        mult = (abs(x_fracao.denominador)/abs(y_fracao.
              denominador))
44        y_fracao = Fracao(y_fracao.numerador * mult, y_fracao
              .denominador * mult)
45
46      #Atribuindo valores as variaveis
47      if x_fracao.numerador < y_fracao.numerador:
48        A = x_fracao.numerador
49        B = y_fracao.numerador
50      else:
51        A = y_fracao.numerador
52        B = x_fracao.numerador
53
54      D = x_fracao.denominador
55
56
57      if A == 0  and B == 0:
58        return 0
59
60      if A == B - 1:
61        return Fracao((A+B),2*D).normalizar()
62
63      else:
64
65        N = B - A - 1
66
67        k = int(log(N, 2))
```

```
68
69          C = int(A / 2**k)
70
71
72          return Fracao((C+1)*(2**k), D).normalizar()
```

Código-fonte 3 – Verifica se a posição representada pela string tem ramificação

```
1  #Checa se a string representa uma posicao que tem
       ramificacao
2  #Note que R(R)(B) tem ramificacao
3  #Mas, R(R) nao tem, assim como RR e R(R(R(R(R(R)))))
4
5  def tem_ramificacao(string):
6
7      #cont conta a quantidade de ramos
8      cont = 0
9
10     #pilha que armazena os par nteses que estao sendo
           abertos
11     pilha = []
12
13     #iterador
14     i = 0
15
16     tamanho = len(string)
17
18
19     while i < tamanho:
20         if string[i] == '(':
21
22             #quando ( e encontrado ele e adicionado na
```

```
            pilha
        pilha.append('(')

        #preencher o filho
        filho = ''
        pilha_filho = pilha

        i = i + 1

        #Armazena a string do filho
        while i < tamanho and len(pilha_filho) != 0:
            if string[i] == '(':
                filho = filho + string[i]
                pilha_filho.append('(')

            elif string[i] == ')':
                pilha_filho.pop()

                if len(pilha_filho) != 0:
                    filho = filho + string[i]

            else:
                filho = filho + string[i]

            i = i + 1

        #Checa se o filho tem ramificacao, se ele tiver
            , entao a string tem ramificacao
        if tem_ramificacao(filho):
            return True

        i = i - 1
```

```
53              pilha.append('(')
54
55          #quando um parenteses for fechado ele sera retirado
                da pilha
56          if i < tamanho and string[i] == ')':
57              pilha.pop()
58
59              #se o tamanho da pilha de parenteses for 0
                   significa que uma ramificacao terminou
60              if len(pilha) == 0:
61                  cont = cont + 1
62
63
64          i = i + 1
65
66      #No final da analise se a pilha estiver vazia e a
           quantidade de ramos for maior que um, entao
67      #existe ramificacao
68      if len(pilha) == 0 and cont > 1:
69          return True
70
71      return False
```

Código-fonte 4 – Corta um segmento

```
1  #Apagar todos os caminhos que dependem da aresta que foi
       retirada e esta na posicao i da string
2  def retirar_pos(string, i):
3      saida = ""
4      pilha = ['(']
5      cont = 0
6
```

```
7       for j in range(len(string)):
8           if j == i:
9               saida = saida + ' '
10          else:
11              if j > i:
12                  if cont == 0:
13                      if string[j] == '(':
14                          pilha.append('(')
15
16                      elif string[j] == ')':
17                          pilha.pop()
18                          if len(pilha) == 0:
19                              cont = 1
20
21                      if len(pilha) == 0:
22                          saida = saida + string[j]
23                      else:
24                          saida = saida + ' '
25
26                  else:
27                      saida = saida + string[j]
28
29              else:
30                  saida = saida + string[j]
31
32      sem_vazios = ""
33
34      saida = saida.replace(' ', '')
35
36      j = 0
37
38      while j <  (len(saida)):
```

```
39          if saida[j] == '(' and saida[j+1] == ')':
40              j = j + 1
41          else:
42              sem_vazios = sem_vazios + saida[j]
43
44          j = j + 1
45
46
47      return sem_vazios
```

Código-fonte 5 – Inverte uma posição

```
1  #Recebe uma string que representa uma posicao e retorna a
       posicao invertida
2  #A posicao RB tem valor -1/2 e seu inverso BR tem valor 1/2
3
4  def inverter_posicao(arvore):
5    arvore = arvore.replace('B', 'I')
6    arvore = arvore.replace('R', 'B')
7    arvore = arvore.replace('I', 'R')
8
9    return arvore
```

Código-fonte 6 – Calcula o valor de uma posição que é um caminho

```
1  #Calcula valor de caminhos em tempo linear
2  def calcular_caminho(caminho):
3
4      caminho = caminho.replace(' ', '')
5      caminho = caminho.replace(')', '')
6      caminho = caminho.replace('(', '')
7
```

```
8       valor = 0
9       cor_anterior = ''
10      base = 2
11
12      for cor in (caminho):
13          if cor_anterior == '' or cor == cor_anterior:
14              cor_anterior = cor
15
16              if cor == 'B':
17                  valor = valor + 1
18              elif cor == 'R':
19                  valor = valor - 1
20
21          else:
22              cor_anterior = "N" #nao vai ser levada em
                   consideracao
23
24              if cor == "B":
25                  valor = valor + 1/base
26              elif cor == 'R':
27                  valor = valor - 1/base
28
29              base = base * 2
30
31      return valor
```

Código-fonte 7 – Calcula o conjunto dos valores obtidos após um movimento em uma posição

```
1 #Recebe uma string e retorna dois vetores
2 #O primeiro contem todos os valores das posicoes obtidos
     apos um movimento de Right
3 #O segundo contem todos os valores das posicoes obtidos
```

```
    apos um movimento de Left
def obter_red_blue(arvore):
    qtde_red = 0
    qtde_blue = 0

    blue = []
    red = []

    qtde_blue = arvore.count('B')
    qtde_red = arvore.count('R')


    tamanho = len(arvore)

    i = 0

    while qtde_blue != 0:
        sub = arvore[:]
        while i < tamanho and arvore[i] != 'B':
            i = i + 1

        sub = retirar_pos(sub, i)
        i = i + 1

        blue.append(calcular_arvore(sub))

        qtde_blue = qtde_blue - 1


    i = 0

    while qtde_red != 0:
```

```
35          sub = arvore[:]
36
37          while i < tamanho and arvore[i] != 'R':
38              i = i + 1
39
40          sub = retirar_pos(arvore, i)
41          i = i + 1
42
43          red.append(calcular_arvore(sub))
44
45          qtde_red = qtde_red - 1
46
47      return red, blue
```

Código-fonte 8 – Calcula o valor de uma posição que é uma árvore

```
1  #Armazena os valores de subposi  es j  calculadas em um
       Hash
2  Hash = {}
3
4  #Recebe uma string de uma posicao e retorna o valor
5  def calcular_arvore(arvore):
6      global Hash
7
8      comeca_com_r =  False
9
10     if len(arvore) > 0  and arvore[0] == 'R':
11       comeca_com_r =  True
12       arvore = inverter_posicao(arvore)
13
14     valor = Hash.get(arvore)
15
```

```
16      if valor == None:
17        if tem_ramificacao(arvore) == False:
18            r = Fracao(calcular_caminho(arvore))
19            Hash[arvore] = r
20            return r
21
22        red,blue = obter_red_blue(arvore)
23
24        if len(red) == 0:
25            return Fracao(max(blue) + 1)
26
27        if len(blue) == 0:
28            return Fracao(min(red) - 1)
29
30        r = nds(min(red),max(blue))
31
32        if not isinstance(r, Fracao):
33            r = Fracao(r)
34
35        Hash[arvore] = r
36
37        if comeca_com_r == True:
38          return r * -1
39
40        return r
41
42      else:
43
44        if comeca_com_r == True:
45          return valor * -1
46
47        return valor
```