UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

JESMMER DA SILVEIRA ALVES

Definitividade de Formas Quadráticas

Uma Abordagem Polinomial

Goiânia
2016

sistema de bibliotecas ufg

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[ ] Dissertação** **[x] Tese**

2. Identificação da Tese ou Dissertação

Nome completo do autor: JESMMER DA SILVEIRA ALVES

Título do trabalho: **DEFINITIVIDADE DE FORMAS QUADRÁTICAS – UMA ABORDAGEM POLINOMIAL**

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Jesmmer da Silveira Alves

Assinatura do (a) autor (a)

Data: 12 / 12 / 2016

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

JESMMER DA SILVEIRA ALVES

Definitividade de Formas Quadráticas

Uma Abordagem Polinomial

Tese apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientadora: Profa. Dr.ª Diane Castonguay

Goiânia
2016

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

da Silveira Alves, Jesmmer
Definitividade de Formas Quadráticas [manuscrito] : Uma Abordagem Polinomial / Jesmmer da Silveira Alves. - 2016.
LXIV, 64 f.

Orientador: Profa. Dra. Diane Castonguay; co-orientador Dr. Thomas Brustle.
Tese (Doutorado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2016.
Bibliografia. Apêndice.
Inclui gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. formas quadráticas. 2. formas unitárias. 3. restrições críticas. 4. restrições hipercríticas. 5. algoritmos polinomiais. I. Castonguay, Diane, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

UFMS

Ata de Defesa de Tese de Doutorado

Aos dezoito dias do mês de novembro de dois mil e dezesseis, no horário das catorze horas, foi realizada, nas dependências do Instituto de Informática da UFG, a defesa pública da Tese de Doutorado do aluno Jesmmer da Silveira Alves, matrícula no. 2012 0632, intitulada **"Definitividade de Formas Quadráticas – Uma Abordagem Polinomial"**.

A Banca Examinadora, constituída pelos professores:

Profa. Dra. Diane Castonguay – INF/UFG - orientadora

Profa. Dra. Carmen Cecília Centeno – DC/PUC-GO

Prof. Dr. Edson Ribeiro Álvares – DMAT/UFPR

Prof. Dr. Fábio Henrique Viduani Martinez – FACOM/UFMS

Prof. Dr. Humberto José Longo - INF/UFG

emitiu o resultado:

(X) Aprovado

( ) Aprovado com revisão

(A Banca Examinadora deve definir as exigências a serem cumpridas pelo aluno na revisão, ficando o orientador responsável pela verificação do cumprimento das mesmas.)

( ) Reprovado

com o seguinte parecer: _____

_____

_____

_____

_____
Profa. Dra. Diane Castonguay

_____
Profa. Dra. Carmen Cecília Centeno

_____
Prof. Dr. Edson Ribeiro Álvares

_____
Prof. Dr. Fábio Henrique Viduani Martinez

_____
Prof. Dr. Humberto José Longo

Jesmmer da Silveira Alves

Jesmmer da Silveira Alves é graduado em Tecnologia de Processamento de Dados (2000), especialista em Redes de Computadores (2003) e Mestre em Ciência da Computação (2008) pela Universidade Federal de Goiás. Durante o doutorado, fez um estágio na Universidade de Sherbrooke (2015, com Prof. Dr. Thomas Brüstle), Quebec, Canadá. Desde 2003, é Professor no Instituto Federal Goiano e tem interesses em pesquisas na complexidade de algoritmos, algoritmos para grafos e algoritmos para teoria de representação de algebras.

Aos meus pais, irmão e irmãs.

Agradecimentos

Ao final deste trabalho agradeço primeiramente a Deus, pela oportunidade de estar vivo, com saúde e pela disposição de desenvolver e terminar este doutorado.

Aos meus pais, Norma Tereza da Silveira e Iloí Alves Sobrinho, meu irmão Jefferson, irmãs Elma e Telma, e minha noiva Thalita pelo apoio, carrinho, amor e pelos inúmeros conselhos que me fizeram crescer ainda mais.

A minha orientadora professora Dr.ª Diane Castonguay, pela oportunidade de trabalhar juntos e aprender com suas diferentes formas orientar e ensinar. Seus ensinamentos foram além de formas quadráticas ou álgebra, sobretudo, um exemplo de dedicação, serenidade e profissionalismo.

Ao professor Dr. Thomas Brüstle, pela calorosa hospitalidade ao me receber em Sherbrooke, pelos grandes ensinamentos passados e pelos exemplos de respeito e companheirismo.

Aos amigos e pessoas que de alguma forma incentivaram e colaboraram para o desenvolvimento deste trabalho. Em especial aos colegas de doutorado Elizângela Dias, Halley Wesley, Walid Jradi, e Leila Roling pelos comentários e sugestões de grande relevância.

Aos professores membros da banca examinadora Dr.ª Carmen Centeno, Dr. Edson Ribeiro Alvares, Dr. Fabio Henrique Viduani Martinez e Dr. Humberto José Longo, pelas contribuições sugeridas.

À FAPEG - Fundação de Amparo à Pesquisa do Estado de Goiás, CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior e IF Goiano pelo suporte financeiro durante o doutorado.

"A menos que modifiquemos a nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo"

Albert Einstein.

Resumo

da Silveira Alves, Jesmmer. **Definitividade de Formas Quadráticas**. Goiânia, 2016. 128p. Tese de Doutorado. Instituto de Informática, Universidade Federal de Goiás.

Formas quadráticas são expressões algébricas que têm papel importante em diferentes áreas da ciência da computação, matemática, física, estatística e outras. Abordamos nesta tese formas quadráticas racionais e formas inteiras, com coeficientes racionais e inteiros respectivamente. Os métodos existentes para reconhecimento de formas quadráticas racionais têm complexidade de tempo exponencial ou usam aproximações que deixam o resultado menos confiável. Apresentamos um algoritmo polinomial que aprimora o melhor-caso do reconhecimento de formas quadráticas para tempo constante. Ainda mais, novas estratégias foram usadas para garantir a confiabilidade dos resultados, representando números racionais como frações de inteiros, e para identificar combinações lineares que são linearmente independentes, usando a redução de Gauss. Sobre o reconhecimento de formas inteiras, identificamos que os algoritmos existentes têm complexidade de tempo exponencial para o tipo fracamente não-negativa e polinomial para o tipo fracamente positiva. No entanto, o grau do polinômio depende da dimensão da álgebra e pode ser muito grande. Apresentamos um algoritmo polinomial para o reconhecimento de formas inteiras fracamente positivas. Este algoritmo identifica restrições hipercríticas avaliando todo subgrafo com 9 vértices do grafo associado à forma inteira. Através da busca em profundidade, uma estratégia similar pôde ser usada no reconhecimento do tipo fracamente positiva. Por fim, mostramos que o reconhecimento de formas inteiras pode ser feito através de mutações na matriz de troca relacionada.

Palavras–chave

Formas quadráticas, redução de Gauss, formas inteiras, formas críticas, formas hipercríticas, algoritmo polinomial.

MSC: 11E25, 15A63, 11E04, 16G60, 42A82, 05C85.

Abstract

da Silveira Alves, Jesmmer. **Definiteness of Quadratic Forms - A Polynomial Approach**. Goiânia, 2016. 128p. PhD. Thesis. Instituto de Informática, Universidade Federal de Goiás.

Quadratic forms are algebraic expressions that have important role in different areas of computer science, mathematics, physics, statistics and others. We deal with rational quadratic forms and integral quadratic forms, with rational and integer coefficients respectively. Existing methods for recognition of rational quadratic forms have exponential time complexity or use approximation that weaken the result reliability. We develop a polinomial algorithm that improves the best-case of rational quadratic forms recognition in constant time. In addition, new strategies were used to guarantee the results reliability, by representing rational numbers as a fraction of integers, and to identify linear combinations that are linearly independent, using Gauss reduction. About the recognition of integral quadratic forms, we identified that the existing algorithms have exponential time complexity for weakly nonnegative type and are polynomial for weakly positive type, however the degree of the polynomial depends on the algebra dimension and can be very large. We have introduced a polynomial algorithm for the recognition of weakly nonnegative quadratic forms. The related algorithm identify hypercritical restrictions testing every subgraph of 9 vertices of the quadratic form associated graph. By adding Depth First Search approach, a similar strategy was used in the recognition of weakly positive type. We have also shown that the recognition of integral quadratic forms can be done by mutations in the related exchange matrix.

Keywords

Quadratic forms, Gauss reduction, unit form, critical restrictions, hypercritical restrictions, polynomial algorithm.

MSC: 11E25, 15A63, 11E04, 16G60, 42A82, 05C85.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO 1

Introdução

Formas quadráticas são polinômios homogêneos[1] de grau 2. Tais expressões algébricas podem ter tipos e aplicações diferentes de acordo com os coeficientes no polinômio. Aqui vamos referenciar como formas quadráticas racionais (identificadas aqui pelo símbolo $q_{\mathbb{Q}}$) os polinômios em que os monômios têm como coeficientes números racionais e como formas unitárias (identificadas aqui pelo símbolo $q_{\mathbb{Z}}$) os polinômios em que os monômios têm como coeficientes números inteiros.

As formas quadráticas racionais podem ser classificadas como positivas ou negativas definidas; positivas ou negativas semidefinidas; e indefinidas. As formas inteiras, além dos tipos descritos anteriormente, também podem ser classificadas como fracamente positivas ou fracamente não-negativas (para mais informações referimos [12, 39, 50, 52]). O objetivo deste trabalho é identificar e analisar os diferentes métodos utilizados para determinar o tipo de uma dada forma quadrática (o que nomeamos como "Definitividade de Formas Quadráticas"), bem como, desenvolver algoritmos polinomiais para o mesmo propósito.

A definitividade de formas quadráticas se tornou um tópico importante na álgebra linear (veja algumas definições básicas sobre álgebra no Apêndice A, Seção A.2) porque pode ser utilizada para conseguir informações relevantes sobre estruturas complexas relacionadas à forma quadrática. Por exemplo, podemos associar uma forma quadrática a uma álgebra obtida a partir de um *quiver* sem ciclos orientados. Neste caso, se a álgebra é de representação finita então a forma quadrática é fracamente positiva (para mais informações referimos [10]).

As formas quadráticas racionais têm aplicações em diferentes áreas da ciência da computação, matemática, física quântica, estatística e outros (veja em [9, 24, 29, 46]). Para este tipo de forma quadrática, existe uma variedade de métodos e algoritmos que permitem, para alguns casos em tempo polinomial, determinar o seu. Estes são geralmente divididos em métodos diretos, métodos expansivos e métodos iterativos.

[1] O termo "homogêneo" significa que todos os monômios têm o mesmo grau no polinômio.

A complexidade de tempo dos algoritmos para se determinar o tipo de uma forma quadrática racional depende do método utilizado e pode ser influenciada por casos específicos. Por exemplo, métodos diretos, em geral têm complexidade de tempo exponencial. No entanto, tais métodos são polinomiais para as formas quadráticas positiva definida e negativa definida. Métodos expansivos não têm boa precisão devido à utilização de limites aproximados. A complexidade de tempo de métodos iterativos depende do número de casas decimais consideradas e pode não ser confiável devido à esta limitação.

Um outro problema vem do fato de que métodos iterativos usam operações matemáticas que fracionam as entradas da matriz relacionada à forma quadrática, apresentando um resultado com um grande número de casas decimais[2]. Este tipo de problema é ocasionado pela utilização de operações como divisão e radiciação, que podem receber uma entrada inteira e transformá-la em um valor com várias casas decimais. Entretanto, utilizamos novas estratégias para evitar este fracionamento das entradas e fornecer um resultado confiável, tais como: armazenar as entradas da matriz como quociente de dois inteiros; e usar a redução de Gauss para facilmente identificar combinações lineares de variáveis que são linearmente independentes.

Já as formas inteiras têm papel importante na teoria de representação de álgebra (veja em [4, 11, 16, 32]). Entretanto, para este tipo de forma quadrática, alguns conceitos relacionados a teoria de complexidade computacional ainda não estão bem definidos. Por exemplo, Dean e De La Peña desenvolveram um algoritmo para decidir se uma dada forma inteira é fracamente não-negativa (veja em [12]) e, apesar do algoritmo atender as expectativas pretendidas pelos autores, não foi analisada a complexidade computacional do algoritmo ou avaliado se é possível desenvolver um algoritmo mais rápido.

A estratégia de Dean e De La Peña foi gerar todas as raízes positivas (vetores que satisfazem $q_{\mathbb{Z}}(x) = 1$) através de reflexões e fazer alguns testes para identificar se a forma inteira é fracamente não-negativa (De La Peña também estendeu este trabalho para formas inteiras fracamente positivas [10]). Sabe-se que algumas formas inteiras fracamente positivas têm pelo menos um número quadrático de raízes positivas. Ademais, uma forma inteira fracamente não-negativa pode ter um número infinito de raízes positivas. Portanto, avaliando-se a forma inteira para todas as raízes positivas é impossível conseguir uma solução linear para identificar o tipo da forma inteira.

Uma solução interessante para verificar se uma forma inteira é fracamente não-negativa é através das restrições hipercríticas, definidas por Unger [48]. Um ponto importante é que todas as restrições na lista de Unger têm 9 ou menos vértices. Portanto, uma forma inteira é fracamente não-negativa se e somente se toda restrição $q^J_{\mathbb{Z}}$, a um subgrafo J de cardinalidade 9 da forma inteira, é fracamente não-negativa. Assim, pode-

[2] Veja alguns problemas resultantes em [19].

se utilizar um algoritmo para avaliar todos os subgrafos com até 9 vértices. Desde que cada subgrafo tenha um número fixo de vértices, a avaliação de cada subgrafo é constante e existem $\binom{n}{9}$ subgrafos, um polinômio em n. Esta estratégia resultou em um algoritmo de complexidade $O(n^9)$.

Outro aspecto deixou esta estratégia ainda mais interessante. Todas as restrições hipercríticas na lista de Unger são conexas (através de arestas negativas). Portanto, somente precisam ser considerados os subgrafos que são conexos, o que resulta em um número menor de subgrafos a serem analisados. Com o uso da pesquisa em profundidade, esta estratégia também pôde ser utilizada para verificar se uma forma inteira é fracamente positiva.

Além destes, analisamos a possibilidade de se fazer a definitividade de formas inteiras através de mutações na matriz de troca relacionada. Fomin e Zelevinsky [20] desenvolveram um processo combinatorial recursivo (chamado de processo de mutação) para definir álgebras *cluster*. Toda dinâmica do processo de mutação é encapsulada em uma matriz conhecida como matriz de troca T.

De acordo com Chaves [8], se Q é um *cluster quiver* e todos os c-vetores na c-matriz relacionada possuem a propriedade de coerência de sinais, então os c-vetores são raízes positivas da forma inteira relacionada. Tais raízes são mantidas em uma submatriz $c(T)$ de T (conhecida como c-matriz de T) e, desde que sejam feitas todas as possíveis mutações na matriz de troca relacionada à uma forma inteira $q_{\mathbb{Z}}$, todas as raízes positivas serão geradas e poderão ser testadas, para se determinar o tipo de $q_{\mathbb{Z}}$.

O restante desta tese está organizado da seguinte forma: o Capítulo 2 apresenta os conceitos básicos sobre formas quadráticas racionais, descreve novas estratégias para deixar o resultado confiável e apresenta algoritmos que utilizam a redução de Gauss para fazer a definitividade de formas quadráticas racionais. O Capítulo 3 descreve conceitos básicos sobre as formas inteiras, apresenta alguns métodos para fazer o reconhecimento de formas inteiras e as respectivas análises computacionais, e descreve algoritmos polinomiais para os mesmos propósitos. O Capítulo 4 apresenta alguns conceitos básicos sobre o processo de mutação através de sequências verdes maximais, mostra como a definitividade de formas inteiras pode ser feita através de mutações, e por fim, descreve algumas formas de simplificar a definitividade quando feita através de mutações. Finalmente, o Capítulo 5 descreve trabalhos futuros e as considerações finais do autor.

CAPÍTULO 2

Formas Quadráticas Racionais

Este capítulo descreve como pode ser identificado o tipo de formas quadráticas racionais. As formas quadráticas racionais (identificadas aqui pelo símbolo $q_{\mathbb{Q}}$) são polinômios em que os monômios têm como coeficientes números racionais. Estratégias simples foram utilizadas para deixar o resultado final confiável, representando números racionais como frações de inteiros, e para identificar de maneira precisa o tipo da forma quadrática, através da redução de Gauss.

O capítulo está organizado da seguinte forma: a Seção 2.1 apresenta alguns conceitos básicos sobre formas quadráticas racionais. A Seção 2.2 descreve como números racionais podem ser armazenados como uma fração de inteiros e também descreve como a redução de Gauss pode ser usada para encontrar combinações de variáveis linearmente independentes. A Seção 2.3 apresenta algoritmos que exploram essas estratégias para identificar o tipo formas quadráticas racionais e, na Seção 2.4, são apresentadas as considerações finais do capítulo.

2.1 Conceitos Básicos

Uma **forma quadrática racional** $q_{\mathbb{Q}}$ associada a uma matriz simétrica $A_{n\times n}$ de valores racionais e avaliada em um vetor $x_{n\times 1}$ é definida pela seguinte expressão:

$$q_{\mathbb{Q}}(x) = (x_1,\ldots,x_n)\cdot\begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \ldots & a_{nn} \end{pmatrix}\cdot\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \sum_{1\leqslant i\leqslant j\leqslant n} a_{ij}\cdot x_i\cdot x_j \qquad (2\text{-}1)$$

Toda forma quadrática racional $q_{\mathbb{Q}}(x) = x^tAx$ pode ser representada por uma matriz simétrica. Seja B uma matriz não-simétrica, então B pode ser transformada em uma matriz simétrica através da fórmula $A = \frac{1}{2}(B+B^t)$. Assim, A é simétrica e $q_{\mathbb{Q}}(x) = x^tAx = x^tBx$ [43, 52].

As formas quadráticas podem ser classificadas através do polinômio resultante da expressão $q_{\mathbb{Q}}(x) = x^t Ax$ como: positiva definida; negativa definida; positiva semidefinida; negativa semidefinida; e indefinida.

Definição 2.1 (O Tipo de uma Forma Quadrática) *Sejam $q_{\mathbb{Q}}$ uma forma quadrática racional e $x \in \mathbb{Q}^n$ um vetor não nulo. Se para todo x, $q_{\mathbb{Q}}(x) > 0$ (resp. $q_{\mathbb{Q}}(x) < 0$), então $q_{\mathbb{Q}}$ é* **positiva definida** (*resp.* **negativa definida**). *Se para todo x, $q_{\mathbb{Q}}(x) \geqslant 0$ (resp. $q_{\mathbb{Q}}(x) \leqslant 0$), então $q_{\mathbb{Q}}$ é* **positiva semidefinida** (*resp.* **negativa semidefinida**). *Por fim, se $q_{\mathbb{Q}}(x) > 0$ para um ou mais vetores x e $q_{\mathbb{Q}}(x) < 0$ para outros vetores x, então $q_{\mathbb{Q}}$ é* **indefinida**.

Exemplo 2.2 *A forma quadrática $q_{\mathbb{Q}}(x) = 3x_1^2 - 7x_2^2$ é indefinida. Isto porque facilmente podemos encontrar vetores x tais que $q_{\mathbb{Q}}(x) > 0$ e vetores x tais que $q_{\mathbb{Q}}(x) < 0$ (este é caso para $[1,0]$ e $[0,1]$, respectivamente).*

2.1.1 A Definitividade de Formas Quadráticas Racionais

O termo **definitividade** está relacionado à classificação da forma quadrática. Uma maneira de se verificar se uma forma quadrática é positiva definida é através dos menores principais naturalmente ordenados [7] da matriz simétrica $A_{n\times n}$ relacionada à forma quadrática. Encontrar todos os menores principais naturalmente ordenados têm complexidade $O(n^4)$.

A expansão direta é um método mais completo e pode ser utilizado para se identificar qualquer tipo de forma quadrática. Este utiliza os menores principais da matriz. Seja A uma matriz de ordem n. A submatriz principal de ordem k de A é obtida apagando-se $n-k$ colunas e as respectivas $n-k$ linhas de A. O determinante de uma submatriz principal de ordem k é chamado de **menor principal** de ordem k de A. Neste caso, o número de determinantes de várias ordens é exponencial.

A definitividade de formas quadráticas também pode ser feita extraindo-se o polinômio característico da matriz e identificando-se as raízes deste polinômio. Encontrar as raízes de um polinômio pode ser descrito como o problema de álgebra linear de encontrar todos os números λ que satisfazem a equação $Av = \lambda v$. Neste caso, λ são os auto valores (do inglês *eigenvalues*) e v os correspondentes vetores próprios (do inglês *eigenvectors*). Assim, desde que a equação resultante é um polinômio de grau n em λ, A têm exatamente n auto valores reais, possivelmente repetidos.

Encontrar (com precisão) os auto valores de um polinômio em geral é uma tarefa difícil. Isto porque os auto valores são números reais e, possivelmente, com uma grande quantidade de casas decimais. Uma outra opção é, após identificar o polinômio característico, determinar limites inferiores e superiores para as raízes deste polinômio. Apesar de serem bem interessantes, os métodos atuais fornecem apenas limites aproximados para as raízes do polinômio.

Também pode-se usar métodos iterativos para determinar os auto valores, tais como os algoritmos *QR*, *Jacobi* e a decomposição de *Cholesky* [40, 47]. Estes três métodos têm complexidade $O(n^3)$, sendo que o método de *Jacobi* é considerado o mais confiável atualmente com relação à precisão dos resultados apresentados. No entanto, pela mesma dificuldade descrita anteriormente, métodos iterativos para determinar os auto valores usualmente são implementados com uma aproximação do número de casas decimais.

No Apêndice B, o leitor pode encontrar mais detalhes sobre os métodos usados atualmente para se identificar o tipo de formas quadráticas racionais, citados nesta subseção.

2.2 Novas Estratégias

2.2.1 Números Racionais como Fração de Inteiros

Após analisar os métodos citados na subseção anterior, identificamos que algumas operações (tais como divisão e radiciação) podem aumentar o número de casas decimais e deixar o resultado final menos confiável ou difícil de ser analisado. Pode-se resolver este problema representando-se números racionais como frações de inteiros. Para isto, cada entrada da matriz relacionada à forma quadrática deve ser expressa como quociente de dois inteiros.

Os Algoritmos 2.2–2.5 fazem a adição, subtração, multiplicação e divisão de dois números racionais. Neste caso, a, b e r são vetores com duas posições (numerador e denominador). O Algoritmo 2.1 retorna o máximo divisor comum entre n e d, o qual é usado nos Algoritmos 2.2, 2.3 e 2.4 (e consequentemente no Algoritmo 2.5) para reduzir ao máximo os valores, antes de serem armazenados na matriz. Nestes algoritmos, assumimos que $a[1]$ e $b[1]$ são diferentes de zero. Além disto, um zero deve ser representado por um vetor $[0,1]$ e, ao representar número racionais, o sinal negativo "−" sempre acompanha o numerador (por exemplo, $-\frac{1}{2}$ é representado por [-1,2]).

Algoritmo 2.1: MDC(n,d)

Entrada: Inteiros n e $d \neq 0$.
Saída: MDC de n e d.

```
se (d = 0) então
    mdc = d
senão
    mdc = MDC(d, mod(n,d))
retorna mdc
```

Algoritmo 2.2: SOMA(a,b)

Entrada: Vetores a e b.
Saída: $r = a + b$.

1 $r[1] \leftarrow a[1] \cdot b[1]$
2 $r[0] \leftarrow a[0] \cdot b[1] + b[0] \cdot a[1]$
3 $mdc \leftarrow \text{MDC}(r[0], r[1])$
4 $r[0] \leftarrow r[0]/mdc$
5 $r[1] \leftarrow r[1]/mdc$
6 **retorna** r

Algoritmo 2.3: SUB(a,b)

Entrada: Vetores a e b.
Saída: $r = a - b$.

1 $r[1] \leftarrow a[1] \cdot b[1]$
2 $r[0] \leftarrow a[0] \cdot b[1] - b[0] \cdot a[1]$
3 $mdc \leftarrow \text{MDC}(r[0], r[1])$
4 $r[0] \leftarrow r[0]/mdc$
5 $r[1] \leftarrow r[1]/mdc$
6 **retorna** r

Algoritmo 2.4: MULT(a,b)

Entrada: Vetores a e b.
Saída: $r = a \cdot b$.

1 $r[0] \leftarrow a[0] \cdot b[0]$
2 $r[1] \leftarrow a[1] \cdot b[1]$
3 $mdc \leftarrow \text{MDC}(r[0], r[1])$
4 $r[0] \leftarrow r[0]/mdc$
5 $r[1] \leftarrow r[1]/mdc$
6 **retorna** r

Algoritmo 2.5: DIV(a,b)

Entrada: Vetores a e $b \neq 0$.
Saída: $r = \frac{a}{b}$.

1 $b2[0] \leftarrow b[1]$
2 $b2[1] \leftarrow b[0]$
3 $r \leftarrow \text{MULT}(a, b2)$
4 **retorna** r

2.2.2 Completando Quadrados

Berger [1] mostrou que, para toda forma quadrática $q_{\mathbb{Q}}$ com n variáveis, existem combinações lineares de variáveis ℓ_i linearmente independentes e números c_i tais que $q_{\mathbb{Q}}(x) = \sum_{i=1}^{n} c_i \cdot \ell_i(x)^2$, para $i \leqslant n$. Esta equação irá resultar em quadrados perfeitos afetados pelo sinal "+" (positivos), pelo sinal "−" (negativos) e nulos (quando $c_i = 0$). Desta forma, o tipo de $q_{\mathbb{Q}}$ pode ser identificado como:

- positiva definida, se todos os quadrados são positivos;
- negativa definida, se todos os quadrados são negativos;
- positiva semidefinida, se existem quadrados positivos e os demais são nulos;
- negativa semidefinida, se existem quadrados negativos e os demais são nulos; e
- indefinida, se existem quadrados positivos e quadrados negativos.

A redução de Gauss [25] é um método que pode ser usado para escrever um polinômio de grau 2 em n variáveis como quadrados perfeitos. Seja $q_{\mathbb{Q}}$ uma forma quadrática não nula, a aplicação da redução de Gauss em $q_{\mathbb{Q}}$ pode ser feita como segue:

- Se $a_{ii} \neq 0$ (para algum $i \in \{1, \ldots, n\}$), elimine uma variável. Sem perda de generalidade, supondo que $i = 1$, a redução de Gauss deve proceder da seguinte forma:

$$\begin{aligned} q_{\mathbb{Q}}(x) &= ax_1^2 + x_1 f(x_2,\ldots,x_n) + p(x_2,\ldots,x_n) \\ &= a\left(x_1 + \tfrac{f(x_2,\ldots,x_n)}{2a}\right)^2 - \tfrac{f(x_2,\ldots,x_n)^2}{4a} + p(x_2,\ldots,x_n) \end{aligned} \tag{2-2}$$

onde f é uma combinação linear e p uma forma quadrática. Então, a nova forma quadrática é:

$$q'_{\mathbb{Q}}(x) = p(x_2,\ldots,x_n) - \tfrac{f(x_2,\ldots,x_n)^2}{4a}. \tag{2-3}$$

Exemplo 2.3 *Seja $q_{\mathbb{Q}}(x) = x_2^2 + x_1x_2 + x_1x_3$. A aplicação da redução de Gauss resulta em dois quadrados positivos e um negativo:*

$$\begin{aligned} q_{\mathbb{Q}}(x) &= x_2^2 + x_1x_2 + x_1x_3 \\ &= (x_2 + \tfrac{1}{2}x_1)^2 - \tfrac{1}{4}x_1^2 + x_1x_3 \\ &= (x_2 + \tfrac{1}{2}x_1)^2 - \tfrac{1}{4}(x_1 - 2x_3)^2 + x_3^2. \end{aligned}$$

- Se $a_{ii} = 0$, $a_{ij} \neq 0$ e $a_{jj} = 0$ (para algum $i, j \in \{1,\ldots,n\}$ e $i \neq j$), elimine duas variáveis.[1] Por questão de simplicidade, suponha que $i = 1$ e $j = 2$. Então, a redução de Gauss deve proceder da seguinte forma:

$$\begin{aligned} q_{\mathbb{Q}}(x) &= ax_1x_2 + x_2 f(x_3,\ldots,x_n) + x_1 g(x_3,\ldots,x_n) + p(x_3,\ldots,x_n) \\ &= (ax_1 + f)\cdot(x_2 + \tfrac{g}{a}) - \tfrac{f\cdot g}{a} + p(x_3,\ldots,x_n) \\ &= \tfrac{1}{4}((l_1 + l_2)^2 - (l_1 - l_2)^2) - \tfrac{f\cdot g}{a} + p(x_3,\ldots,x_n) \end{aligned} \tag{2-4}$$

onde f e g são combinações lineares, p é uma forma quadrática, $l_1 = (ax_1 + f)$ e $l_2 = (x_2 + \frac{g}{a})$. A nova forma quadrática é:

$$q'_{\mathbb{Q}}(x) = p(x_3,\ldots,x_n) - \tfrac{1}{a} f(x_3,\ldots,x_n)\cdot g(x_3,\ldots,x_n). \tag{2-5}$$

Neste caso, $q_{\mathbb{Q}}$ é sempre indefinida. Note-se que, se $a_{ii} = 0$ para todo $j = 1\ldots n$, $a_{ij} = 0$.

Exemplo 2.4 *Seja $q_{\mathbb{Q}}(x) = x_1x_2 + x_1x_3 + x_2x_3$. A aplicação da redução de Gauss resulta em um quadrado positivo e dois negativos:*

$$\begin{aligned} q_{\mathbb{Q}}(x) &= x_1x_2 + x_1x_3 + x_2x_3 \\ &= (x_1 + x_3)(x_2 + x_3) - x_3^2 \\ &= \tfrac{1}{4}(x_1 + x_2 + 2x_3)^2 - \tfrac{1}{4}(x_1 - x_2)^2 - x_3^2. \end{aligned}$$

[1] No caso de $a_{jj} \neq 0$, deve-se retornar ao caso anterior (veja Equação 2-2).

2.3 Implementando as Novas Estratégias

O objetivo dos algoritmos apresentados nesta seção é fazer a definitividade de formas quadráticas através da redução de Gauss. O Teorema 2.3.1 e Corolário 2.3.1 são usados para identificar as formas quadráticas positiva definida e positiva semidefinida, respectivamente.

Teorema 2.3.1 *[25] Uma forma quadrática $q_{\mathbb{Q}}$ é positiva definida se e somente se a aplicação da redução de Gauss resulta em n quadrados afetados pelo sinal "+".*

Corolário 2.3.1 *Uma forma quadrática $q_{\mathbb{Q}}$ é positiva semidefinida se a aplicação da redução de Gauss resulta em $k < n$ quadrados, todos afetados pelo sinal "+".*

Prova. Seja $q_{\mathbb{Q}}(x) = \sum_{i=1}^{k} c_i \cdot \ell_i(x)^2$, $c_i \neq 0$. A redução de Gauss elimina uma variável e fornece uma forma linear (afetada um dos sinais "+" ou "−"), ou elimina duas variáveis e fornece duas formas lineares (uma afetada pelo sinal "+" e outra pelo sinal "−"). Assim, desde que $q_{\mathbb{Q}}$ é positiva semidefinida todos os quadrados devem ser positivos e k é menor que n. Caso contrário, $q_{\mathbb{Q}}$ é indefinida (possui pelo menos um quadrado negativo) ou $q_{\mathbb{Q}}$ é positiva definida (de acordo com o Teorema 2.3.1). A volta desta prova segue facilmente. □

Por questão de organização e simplicidade, dividimos a implementação das novas idéias em 3 partes: um modelo de representação através de matrizes para armazenar e atualizar os elementos da forma quadrática; um algoritmo que aperfeiçoa o melhor caso na definitividade de formas quadráticas racionais; e um algoritmo que considera os demais casos.

2.3.1 Modelo de Representação Através de Matrizes

Os Algoritmos 2.6 e 2.7, durante a aplicação da redução de Gauss, usam um modelo de representação através de matrizes para armazenar e atualizar os elementos da forma quadrática.

Este modelo foi desenvolvido com o objetivo de reduzir o tempo de processamento, fazendo com que os algoritmos trabalhem somente com os elementos na diagonal principal e acima desta. Neste caso, $q_{\mathbb{Q}}$ é representada por um matriz triangular superior A, com suas entradas definidas de acordo com os coeficientes de cada termo como frações de inteiros.

Exemplo 2.5 *A forma quadrática* $q_{\mathbb{Q}}(x) = x_1^2 + 2x_2^2 + 5x_3^2 + 2x_1x_2 - 4x_2x_3$*, é representada por uma matriz triangular superior como segue:*

$$A = \begin{bmatrix} \frac{1}{1} & \frac{2}{1} & \frac{0}{1} \\ \frac{0}{1} & \frac{2}{1} & \frac{-4}{1} \\ \frac{0}{1} & \frac{0}{1} & \frac{5}{1} \end{bmatrix}.$$

A Tabela 2.1 mostra a aplicação da redução de Gauss na forma quadrática $q_{\mathbb{Q}}(x) = x_1^2 + 2x_2^2 + 5x_3^2 + 2x_1x_2 - 4x_2x_3$ e a respectiva matriz A. Os quadrados perfeitos estão identificados em negrito e as novas formas quadráticas $q'_{\mathbb{Q}}$ e $q''_{\mathbb{Q}}$ estão representadas pelas novas matrizes A' e A''. As linhas e colunas destacadas com a cor cinza já foram processadas pelo algoritmo. Neste caso, desde que a aplicação da redução de Gauss resulta em três quadrados afetados pelo sinal "+", então $q_{\mathbb{Q}}$ é positiva definida.

$q_{\mathbb{Q}}(x) = x_1^2 + 2x_2^2 + 5x_3^2 + 2x_1x_2 - 4x_2x_3$	$A = \begin{bmatrix} \frac{1}{1} & \frac{2}{1} & \frac{0}{1} \\ \frac{0}{1} & \frac{2}{1} & \frac{-4}{1} \\ \frac{0}{1} & \frac{0}{1} & \frac{5}{1} \end{bmatrix}$
$x_1^2 + 2x_1x_2 = (\mathbf{x_1 + x_2})^\mathbf{2}$ $q'_{\mathbb{Q}}(x) = 2x_2^2 + 5x_3^2 - 4x_2x_3 - x_2^2$ $q'_{\mathbb{Q}}(x) = x_2^2 + 5x_3^2 - 4x_2x_3$	$A' = \begin{bmatrix} \frac{1}{1} & \frac{2}{1} & \frac{0}{1} \\ \frac{0}{1} & \frac{1}{1} & \frac{-4}{1} \\ \frac{0}{1} & \frac{0}{1} & \frac{5}{1} \end{bmatrix}$
$x_2^2 - 4x_2x_3 = (\mathbf{x_2 - 2x_3})^\mathbf{2}$ $q''_{\mathbb{Q}}(x) = 5x_3^2 - 4x_3^2$ $q''_{\mathbb{Q}}(x) = \mathbf{x_3^2}$	$A'' = \begin{bmatrix} \frac{1}{1} & \frac{2}{1} & \frac{0}{1} \\ \frac{0}{1} & \frac{1}{1} & \frac{-4}{1} \\ \frac{0}{1} & \frac{0}{1} & \frac{1}{1} \end{bmatrix}$

Tabela 2.1: *A aplicação da redução de Gauss na forma quadrática* $q_{\mathbb{Q}}(x) = x_1^2 + 2x_2^2 + 5x_3^2 + 2x_1x_2 - 4x_2x_3$ *e a respectiva matriz representação A.*

2.3.2 Um Algoritmo que Otimiza o Melhor Caso

Esta subseção apresenta um algoritmo que explora algumas situações em que a definitividade de formas quadráticas é feita de maneira mais rápida, analisando-se somente os elementos na diagonal principal de A. Os Lemas 2.3.1, 2.3.2 e o Teorema 2.3.2 evidenciam situações em que a forma quadrática é indefinida. No Algoritmo 2.6, descrevemos a implementação destas estratégias.

Lema 2.3.1 *Se* $a_{ii} = 0$, $a_{ij} \neq 0$ *e* $a_{jj} \neq 0$, *para algum* $i, j \in \{1, \ldots, n\}$, *então* $q_{\mathbb{Q}}$ *é indefinida.*

Prova. Suponha sem perda de generalidade que $i = 1$ e $j = 2$. Primeiro elimina-se x_2. Em seguida, o coeficiente x_1^2 é diferente de zero e pode-se eliminar x_1. Segue a redução:

$$\begin{aligned}
q_{\mathbb{Q}}(x) &= bx_2^2 + ax_1x_2 + x_2f(x_3,\ldots,x_n) + x_1g(x_3,\ldots,x_n) + p(x_3,\ldots,x_n)\\
&= bx_2^2 + x_2(ax_1+f) + x_1g + p\\
&= b\left(x_2 + \tfrac{ax_1+f}{2b}\right)^2 - \tfrac{(ax_1+f)^2}{4b} + x_1g + p\\
&= b\left(x_2 + \tfrac{ax_1+f}{2b}\right)^2 - \tfrac{(a^2x_1^2+2ax_1f+f^2)}{4b} + x_1g + p\\
&= b\left(x_2 + \tfrac{ax_1+f}{2b}\right)^2 - \tfrac{a^2}{4b}x_1^2 + x_1\left(\tfrac{-2af}{4b} + g\right) - \tfrac{f^2}{4b} + p\\
&= b\left(x_2 + \tfrac{ax_1+f}{2b}\right)^2 - \tfrac{a^2}{4b}x_1^2 + x_1\left(\tfrac{2bg-af}{2b}\right) - \tfrac{f^2}{4b} + p\\
&= b\left(x_2 + \tfrac{ax_1+f}{2b}\right)^2 - \tfrac{a^2}{4b}\left(x_1 + \frac{\frac{2bg-af}{2b}}{\frac{-2a^2}{4b}}\right)^2 - \frac{\left(\frac{2bg-af}{2b}\right)^2}{-\frac{4a^2}{4b}} - \tfrac{f^2}{4b} + p\\
&= b\left(x_2 + \tfrac{ax_1+f}{2b}\right)^2 - \tfrac{a^2}{4b}\left(x_1 + \tfrac{2bg-af}{a^2}\right)^2 - \frac{\frac{(2bg-af)^2}{4b^2}}{-\frac{4a^2}{4b}} - \tfrac{f^2}{4b} + p\\
&= b\left(x_2 + \tfrac{ax_1+f}{2b}\right)^2 - \tfrac{a^2}{4b}\left(x_1 + \tfrac{2bg-af}{a^2}\right)^2 + \tfrac{(2bg-af)^2}{4a^2b} - \tfrac{f^2}{4b} + p
\end{aligned}$$

onde f e g são combinações lineares, e p é uma forma quadrática. Desde que $sgn(-\frac{a^2}{4b}) = -sgn(b)$, [2] então $q_{\mathbb{Q}}$ é indefinida. □

Lema 2.3.2 *Se $a_{ii} = 0$, $a_{ij} \neq 0$ e $a_{jj} = 0$, para algum $i, j \in \{1,\ldots,n\}$ e $i \neq j$, então $q_{\mathbb{Q}}$ é indefinida.*

Prova. Neste caso, a redução de Gauss vai eliminar duas variáveis e resulta em duas formas lineares, uma afetada pelo sinal "+" e outra pelo sinal "−" (veja Equação 2-4). Portanto, $q_{\mathbb{Q}}$ é indefinida. □

Teorema 2.3.2 *Se $q_{\mathbb{Q}}$ é positiva definida (resp. positiva semidefinida) então $a_{ii} > 0$, para $i \in \{1,\ldots,n\}$ (resp. $a_{ii} \geqslant 0$ e, se $a_{ii} = 0$, então $a_{ij} = 0$, para todo $j \in \{i+1,\ldots,n\}$).*

Prova. Suponha que $a_{ii} \leqslant 0$, para algum $i \in \{1,\ldots,n\}$. Pode-se iniciar por a_{ii} e verificar que a redução de Gauss não resulta em n quadrados positivos. Assim, pelo Teorema 2.3.1 e Corolário 2.3.1, $q_{\mathbb{Q}}$ não é positiva definida. Por outro lado, seja $a_{ii} = 0$. Suponha que existe $j \in \{1,\ldots,n\}$ com $a_{ij} \neq 0$. Pelos Lemas 2.3.1 e 2.3.2, $q_{\mathbb{Q}}$ é indefinida. □

[2] A função *sgn(b)* retorna o sinal predominante na expressão b.

No Algoritmo 2.6, o comando de repetição na linha 4 passa por todos os elementos na diagonal da matriz A e calcula o número de elementos positivos, negativos e iguais a zero. Enquanto isto, se encontrar um elemento negativo e já existir um positivo, ou vice-versa, o processo termina informando que $q_{\mathbb{Q}}$ é indefinida (linhas 9 e 12). Na linha 14, o algoritmo verifica se $q_{\mathbb{Q}}$ têm todos os elementos na diagonal iguais a zero. Finalmente, o Algoritmo 2.6 (VERIFICAR_DIAGONAL) aciona o Algoritmo 2.7 (DEFINITIVIDADE) para analisar as demais possibilidades (linhas 17 e 19).

Algoritmo 2.6: VERIFICAR_DIAGONAL(A)

Entrada: A = matriz da forma quadrática $q_{\mathbb{Q}} \neq 0$.
Saída: Definitividade de $q_{\mathbb{Q}}$.

```
1  pos ← 0
2  neg ← 0
3  zero ← 0
4  para (i ← 1...n) faça
5      se (a_ii = 0) então
6          zero ← zero + 1
7      senão
8          se (a_ii > 0) então
9              se (neg > 0) então retorna "q_Q é Indefinida!"
10             senão pos ← pos + 1
11         senão
12             se (pos > 0) então retorna "q_Q é Indefinida!"
13             senão neg ← neg + 1
14 se (zero = n) então
       // q_Q têm todos os elementos na diagonal iguais a zero
15     retorna "q_Q é Indefinida!"
16 se (pos ≠ 0) então                          // pos ≠ 0 e neg ≠ 0
17     retorna DEFINITIVIDADE(A, "+", 0)
18 senão
19     retorna DEFINITIVIDADE(−A, "−", 0)
```

O Algoritmo 2.6 têm complexidade $O(n^3)$ no pior caso, influenciado principalmente pelo acionamento do Algoritmo 2.7. Entretanto, têm complexidade $O(1)$ no melhor caso. Isto acontece quando a matriz têm elemento(s) negativo(s) e positivo(s) na diagonal de A.

2.3.3 Analisando os Demais Casos

O Algoritmo 2.7 analisa os casos não tratados pelo Algoritmo 2.6. No Algoritmo 2.6, se $pos \neq 0$ (resp. $neg \neq 0$) então $q_{\mathbb{Q}}$ $(resp. -q_{\mathbb{Q}})$ não pode ser negativa definida. Assim, o Algoritmo 2.7 somente verifica se $q_{\mathbb{Q}}$ (resp. $-q_{\mathbb{Q}}$) é positiva definida, positiva semidefinida ou indefinida. Note que, este é o caso para a primeira chamada do Algoritmo 2.7 mas não necessariamente para as demais, e que $q_{\mathbb{Q}}$ é negativa definida (resp. negativa semidefinida) se e somente se $-q_{\mathbb{Q}}$ é positiva definida (resp. positiva semidefinida).

De maneira simples, o Algoritmo 2.7 recebe a matriz A, completa o primeiro quadrado,[3] atualiza a forma quadrática e chama a si mesmo recursivamente (sem a linha e consequentemente a coluna do termo atualizado na forma quadrática) até que A seja um único elemento ou o processo falhe em algum teste.

De acordo com a redução de Gauss, o comando de repetição na linha 8 do Algoritmo 2.7 atualiza os elementos x_{ii} em $q_{\mathbb{Q}}$ e o comando na linha 12 atualiza os elementos x_{ij}, como segue:

$$x_{ii} = a_{ii} - \frac{a_{ii}^2}{4 \cdot a_{11}}, \quad x_{ij} = a_{ij} - \frac{a_{1i} \cdot a_{1j}}{2 \cdot a_{11}}, \tag{2-6}$$

para $2 \leqslant i \leqslant n$ e $i+1 \leqslant j \leqslant n$. Desde que estamos verificando se $q_{\mathbb{Q}}$ é positiva definida, se o algoritmo encontrar um $a_{ii} < 0$ (linha 10), então $q_{\mathbb{Q}}$ é indefinida.

O parâmetro t define se estamos lidando com $q_{\mathbb{Q}}$ ou $-q_{\mathbb{Q}}$ (como "+" ou "−" respectivamente), e o parâmetro *szero* é usado para definir se o primeiro elemento na linha (ou no quadrado perfeito) é igual a zero. O comando de repetição na linha 3 identifica os casos em que $q_{\mathbb{Q}}$ é indefinida, de acordo com os Lemas 2.3.1 e 2.3.2.

Desde que os comandos usados para fazer a redução de Gauss (nas linhas 8 a 13) têm custo $O(n^2)$ e o procedimento pode ser chamado recursivamente até $n-1$ vezes, o Algoritmo 2.7 têm custo $O(n^3)$ no pior caso.

[3] As funções definidas na Seção 2.2 foram usadas calcular as entradas da matriz como frações de inteiros.

Algoritmo 2.7: DEFINITIVIDADE($A, t, szero$)

Entrada: A = matriz da forma quadrática $q_{\mathbb{Q}}$,[a] t =“+” ou “−”.

Saída: Definitividade de $q_{\mathbb{Q}}$.

```
1  se (n ⩾ 2) então
2      se (a_11 = 0) então
3          para (k ← 2...n) faça
4              se (a_1k ≠ 0) então
5                  retorna "q_Q é Indefinida!"   /* Identifica os casos em que q_Q é
                   indefinida, de acordo com os Lemas 2.3.1 e 2.3.2 */
6          szero ← 1 /* Identifica se o primeiro elemento na linha (ou no quadrado
           perfeito) é igual a zero. */
7      senão
8          para (i ← 2...n) faça
9              a_ii ← SUB(a_ii, DIV(MULT(a_1i, a_1i), MULT([4,1], a_11)))
10             se (a_ii < 0) então
11                 retorna "q_Q é Indefinida!"
12             para (j ← i+1...n) faça
13                 a_ij ← SUB(a_ij, DIV(MULT(a_1i, a_1j), MULT([2,1], a_11)))
14     DEFINITIVIDADE(A(2...n, 2...n), t, szero) /* Remove a linha 1 e a coluna 1
       de A e aciona DEFINITIVIDADE */
15 senão
16     se (a_11 = 0) então
17         szero ← 1
18     se (szero = 0) então
19         se (t = "+") então
20             retorna "q_Q é Positiva Definida!"
21         senão
22             retorna "q_Q é Negativa Definida!"
23     senão
24         se (t = "+") então
25             retorna "q_Q é Positiva Semidefinida!"
26         senão
27             retorna "q_Q é Negativa Semidefinida!"
```

[a] Quando acionado através do Algoritmo 2.6 (VERIFICAR_DIAGONAL) $a_{ii} \geqslant 0$ para todo i.

Teorema 2.3.3 *O Algoritmo 2.7 está correto.*

Prova. Primeiro ressaltamos que toda chamada ao Algoritmo 2.7 reduz o número de linhas e colunas de A em 1 (veja linha 14). Assim, eventualmente $n = 1$, o teste na linha 1 é *Falso* e o algoritmo pára. Caso contrário, o processo falha no teste da linha 4 ou no teste da linha 10 e o algoritmo pára. Nós vamos provar a corretude de DEFINITIVIDADE por indução em n. O caso base é quando $n = 1$ e A têm um elemento $a_{11} \geqslant 0$. Se $a_{11} \neq 0$, então $q_{\mathbb{Q}}$ é positiva definida. Senão, $q_{\mathbb{Q}}$ é positiva semidefinida.

Seja A uma matriz de ordem n. Se $a_{11} = 0$, nas linhas 2 a 4, o algoritmo verifica as condições definidas nos Lemas 2.3.1 e 2.3.2 e, caso estes sejam validados, $q_{\mathbb{Q}}$ é indefinida (linha 5). Caso contrário, $a_{1j} = 0$ para todo j e x_1 que não estão em $q_{\mathbb{Q}}$ e $szero = 1$ (linha 6). Na linha 14, usamos o algoritmo para verificar a definitividade de $q'_{\mathbb{Q}} = q_{\mathbb{Q}}$, onde A' é uma matriz $(n-1) \times (n-1)$. Desde que $szero = 1$, podemos concluir que $q_{\mathbb{Q}}$ é indefinida ou positiva semidefinida. Por outro lado, se $a_{11} \neq 0$, as linhas 8 a 13 calculam $q'_{\mathbb{Q}}$ usando a redução de Gauss. Assim, $q_{\mathbb{Q}}(x) = a_{11}l(x)^2 + q'_{\mathbb{Q}}$ para alguma combinação linear l. Na linha 10, verificamos se algum dos novos a_{ii} é menor que zero. Neste caso, $q'_{\mathbb{Q}}$ não é positiva definida nem positiva semidefinida, portanto, $q_{\mathbb{Q}}$ é indefinida (linha 11). Desta forma, quando DEFINITIVIDADE é acionado para $q'_{\mathbb{Q}}$ (linha 14), todos os elementos da diagonal de $q'_{\mathbb{Q}}$ são não-negativos. Ainda mais, $q_{\mathbb{Q}}$ é positiva definida (positiva semidefinida) se e somente se $q'_{\mathbb{Q}}$ é positiva definida (positiva semidefinida). □

Teorema 2.3.4 *O Algoritmo 2.6 está correto.*

Prova. O Algoritmo 2.6 usa o comando na linha 4 para analisar os n elementos na diagonal de A. E se necessário, aciona o Algoritmo 2.7 para casos ainda não resolvidos. Portanto, desde que o Algoritmo 2.7 termina (veja o Teorema 2.3.3), o Algoritmo 2.6 também termina. Para provar que o Algoritmo 2.6 mostra o resultado correto, temos que analisar três diferentes situações de acordo com a redução de Gauss:

1. $zero = n$ (linha 14), $q_{\mathbb{Q}}$ é indefinida. Neste caso, $a_{ii} = 0$ para todo $i \in \{1, \ldots, n\}$. Assumindo que $q_{\mathbb{Q}}$ é uma forma quadrática não nula, o método elimina duas variáveis, mostra dois quadrados e $q_{\mathbb{Q}}$ é indefinida (veja Lema 2.3.2);
2. $a_{ii} > 0$ e $neg > 0$ (linha 8), $q_{\mathbb{Q}}$ é indefinida. Estas condições significam que o processo encontrou um $a_{ii} > 0$ e já havia encontrado pelo menos um $a_{jj} < 0$, para $j \in \{1, \ldots, i-1\}$, em $q_{\mathbb{Q}}$. Como podemos começar a redução de Gauss por a_{ii} ou por a_{jj}, obteremos pelo menos um quadrado afetado pelo sinal "+"e outro pelo sinal de "−". Pelo Teorema 2.3.1 e Corolário 2.3.1, $q_{\mathbb{Q}}$ é indefinida;
3. $a_{ii} < 0$ e $pos > 0$ (linhas 11 e 12), $q_{\mathbb{Q}}$ é indefinida. Segue facilmente como em 2.

□

2.4 Considerações Finais

Os algoritmos apresentados neste capítulo têm complexidade polinomial ($O(1)$ no melhor caso e $O(n^3)$ no pior caso) e são similares aos melhores algoritmos (que são iterativos e expansivos) conhecidos para identificar o tipo de formas quadráticas racionais. Uma questão importante neste capítulo é que os algoritmos apresentados permitiram que os resultados fossem precisos.

A redução de Gauss para completar quadrados permitiu a utilização de um aspecto simples, mas muito eficiente. O sinal que afeta cada quadrado perfeito é uma maneira confiável de identificar formas lineares que são linearmente independentes e, usando estratégias simples para completar quadrados, conseguimos determinar o tipo da forma quadrática.

A estratégia de representação de números racionais como frações de inteiros usa operações básicas (soma, subtração, multiplicação e divisão) que mantém as entradas da matriz como valores inteiros. Este processo é muito importante porque fornece um resultado preciso e de fácil identificação, diferentemente dos resultados fornecidos por algoritmos iterativos.

CAPÍTULO 3

Formas Inteiras

Uma caracterização particular de formas quadráticas acontece quando os coeficientes no polinômio são números inteiros. Neste caso, a forma quadrática é conhecida como forma unitária (referenciada aqui com o símbolo $q_{\mathbb{Z}}$) e têm papel importante na teoria de representação de álgebra. Por exemplo, para certo tipo de álgebra A existe uma forma unitária $q_{\mathbb{Z}}$ relacionada tal que:

- se A é de representação finita, então $q_{\mathbb{Z}}$ é fracamente positiva;
- se A é *tame* (representações parametrizáveis), então $q_{\mathbb{Z}}$ é fracamente não-negativa.

As formas inteiras foram inicialmente apresentadas por Tits[1] e usadas principalmente na representação de álgebra hereditária de representação finita. Não vamos aprofundar neste assunto porque esta tese têm uma abordagem computacional. Para mais informações sugerimos os trabalhos [4, 11, 16, 32].

Neste capítulo, descrevemos inicialmente como é os métodos atuais para identificar o tipo de formas inteiras. Com a análise destes métodos, verificamos que nenhum algoritmo para identificar se uma forma unitária é fracamente não-negativa têm complexidade de tempo exponencial no pior caso. Para melhorar este desempenho computacional, utilizamos estratégias baseadas em grafos no desenvolvimento de um algoritmo polinomial para o mesmo propósito. Com o uso da pesquisa em profundidade, esta estratégia também pôde ser utilizada para verificar se uma forma unitária é fracamente positiva.

O capítulo está organizado da seguinte forma: a Seção 3.1 descreve conceitos básicos com base nos trabalhos de Dean e De La Peña [10, 12]; na Seção 3.2, também com base em [10, 12], são apresentados alguns métodos para fazer o reconhecimento de formas inteiras e as respectivas análises computacionais; a Seção 3.3 apresenta algoritmos polinomiais para os mesmos propósitos; a Seção 3.4 mostra alguns experimentos empíricos com importantes tipos de álgebras e *quivers* criados aleatoriamente; finalmente, na Seção 3.5 são apresentadas a considerações finais sobre o capítulo.

[1] Na literatura, este tipo de forma quadrática é conhecido como *Tits Form*.

3.1 Conceitos Básicos

Uma forma unitária $q_{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$ é definida pela expressão:

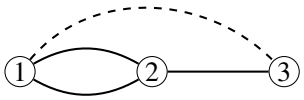
$$q_{\mathbb{Z}}(x) = \sum_{i=1}^{n} x_i^2 + \sum_{i<j} a_{ij} x_i x_j, \ \forall\, x \in \mathbb{Z}^n. \tag{3-1}$$

O vetor $x \in \mathbb{Z}^n$ é considerado **positivo** se $x \neq 0$ e $x_i \geqslant 0$ para todo i. Assim, uma forma unitária $q_{\mathbb{Z}}$ é **fracamente positiva** se $q_{\mathbb{Z}}(x) > 0$ para todo vetor positivo $x \in \mathbb{Z}^n$, ou **fracamente não-negativa** $q_{\mathbb{Z}}(x) \geqslant 0$ para todo vetor positivo $x \in \mathbb{Z}^n$. A forma bilinear simétrica associada a $q_{\mathbb{Z}}$ é dada por $q_{\mathbb{Z}}(-,-) : \mathbb{Z}^n \to \mathbb{Z}$ com $q_{\mathbb{Z}}(x) = \frac{1}{2} q_{\mathbb{Z}}(x,x)$ e, a matriz simétrica A (tal que $a_{ij} = a_{ji}$) de $q_{\mathbb{Z}}$ é a matriz $A = (a_{ij})$ com $a_{ii} = 2$, para $1 \leqslant i \leqslant n$.

Uma maneira conveniente de descrever formas inteiras é através de *quivers*. Um **Quiver** $Q = (Q_0, Q_1)$ é um grafo finito e conexo com um conjunto de vértices $Q_0 = \{1, \ldots, n\}$, um conjunto de arestas $Q_1 = \{\alpha_1, \ldots, \alpha_r\}$ e as funções $s, t : Q_1 \to Q_0$, tal que para cada aresta $\alpha : i \to j$, $s(\alpha) = i$ é a origem e $t(\alpha)$ o destino da aresta. Note-se que a orientação das arestas só é importante ao analisar a álgebra relacionada à forma inteira. Nos nossos exemplos, todos os grafos são não direcionados.

No *quiver* Q relacionado a uma forma unitária $q_{\mathbb{Z}}$, o conjunto de vértices $Q_0 = \{1, \ldots, n\}$ é definido através dos coeficientes a_{ij} de $q_{\mathbb{Z}}$. O conjunto de arestas Q_1 e as funções s e t são definidos como segue: quando $a_{ij} < 0$, conecta-se o vértice i com o vértice j por $-a_{ij}$ arestas sólidas (que chamamos de arestas negativas); e quando $a_{ij} > 0$, conecta-se i com j por a_{ij} arestas tracejadas (arestas positivas). O grafo resultante têm n vértices, possivelmente com múltiplas arestas mas sem laços.

Exemplo 3.1 *Na Figura 3.1, (a) descreve a representação quiver Q da forma unitária $q_{\mathbb{Z}}(x) = \sum_{i=1}^{3} x_i^2 - 2x_1x_2 - x_2x_3 + x_1x_3$ e, (b) a matrix simétrica A associada com $q_{\mathbb{Z}}$. Neste caso, $q_{\mathbb{Z}}$ é fracamente não-negativa mas não é fracamente positiva.*



(a) *Quiver Q.*

$$\begin{bmatrix} 2 & -2 & 1 \\ -2 & 2 & -1 \\ 1 & -1 & 2 \end{bmatrix}$$

(b) *Matrix simétrica A.*

Figura 3.1: *Representações de uma forma unitária $q_{\mathbb{Z}}$.*

3.1.1 Formas Inteiras Críticas e Hipercríticas

Para um conjunto $J = \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$ de índices, denotamos por $s_J : \mathbb{Z}^m \to \mathbb{Z}^n$ a **inclusão** tal que $s_J(e_j) = e_{i_j}$, onde $\{e_i, \ldots, e_k\}$ é a base canônica de $\mathbb{Z}^k$, para $k \in \mathbb{N}$. Então $q^J = q_{s_J}$ é uma **restrição** de $q_{\mathbb{Z}}$ para J.

Definição 3.2 (Formas Críticas e Hipercríticas) *Uma forma unitária $q_{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$ é crítica se toda restrição q^J é fracamente positiva mas $q_{\mathbb{Z}}$ não é. De maneira similar, $q_{\mathbb{Z}}$ é hipercrítica se toda restrição q^J é fracamente não-negativa mas $q_{\mathbb{Z}}$ não é.*

Todas as formas críticas (ou grafos críticos) foram classificadas por von Höhne [50] e todas as formas hipercríticas por Unger [48]. O leitor encontra exemplos de grafos críticos no Apêndice C.1.1 e hipercríticos no Apêndice C.1.2.

Exemplo 3.3 *A forma unitária na Figura 3.2 é hipercrítica. Isto é, $q_{\mathbb{Z}}$ não é fracamente não-negativa mas todas as restrições q^J, para $1 \leqslant j \leqslant n$ são fracamente não-negativas. Ou seja, toda forma unitária definida a partir de qualquer subgrafo conexo de $q_{\mathbb{Z}}$ com menos de n vértices é fracamente não-negativa, mas $q_{\mathbb{Z}}$ não é.*

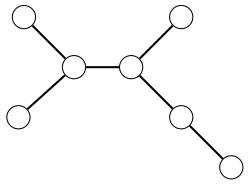


Figura 3.2: *Uma forma unitária hipercrítica $q_{\mathbb{Z}}$.*

Na Seção 3.3 utilizamos o conceito de formas críticas e hipercríticas no desenvolvimento de algoritmos polinomiais para a identificação do tipo de uma forma unitária. Desta forma, é necessário ressaltar algumas características importantes das restrições q^J de uma forma unitária $q_{\mathbb{Z}}$.

Teorema 3.1.1 *Seja $q_{\mathbb{Z}}$ uma forma unitária. Se $q_{\mathbb{Z}}$ é fracamente positiva, então toda restrição q^J também é fracamente positiva.*

Prova. Seja $q_{\mathbb{Z}}$ é fracamente positiva e que existe uma restrição q^J que não é fracamente positiva. Se q^J não é fracamente positiva, então existe um vetor z positivo tal que a forma unitária $q^J(z) \leqslant 0$. Sem perda de generalidade, vamos supor que $q^J(z) = 0$. Agora considere um outro vetor positivo $z' = z \cup x_i$, para $i = 1 \ldots n$, tal que x_i é uma variável de $q_{\mathbb{Z}}$ que não estava incluída em z. Desde que $x_i \geqslant 0$, é claro que $q'^{J'}(z') = 0$ quando $x_i = 0$. Desta forma, pode-se incluir todas as variáveis x_i de $q_{\mathbb{Z}}$ (que ainda não estavam em z) em z' e $q_{\mathbb{Z}}(z') = 0$. Portanto, $q_{\mathbb{Z}}$ não pode ser fracamente positiva. □

Teorema 3.1.2 *Seja $q_{\mathbb{Z}}$ uma forma unitária. Se $q_{\mathbb{Z}}$ é fracamente não-negativa, então toda restrição q^J também é fracamente não-negativa.*

Prova. Esta prova se desenvolve da mesma forma que a prova do Teorema 3.1.1. □

3.1.2 Reconhecimento de Formas Inteiras

O conjunto Σ_q^1 de **raízes positivas** de $q_{\mathbb{Z}}$, $\Sigma_q^1 = \{x \in \mathbb{Z}^n : 0 \leqslant x \text{ e } q_{\mathbb{Z}}(x) = 1\}$, e o conjunto de Σ_q^0 de **raízes nulas** de $q_{\mathbb{Z}}$, $\Sigma_q^0 = \{x \in \mathbb{Z}^n : 0 \leqslant x \text{ e } q_{\mathbb{Z}}(x) = 0\}$, são usados para caracterizar as formas inteiras. Os exemplos de raízes positivas mais facilmente identificados são os vetores e da base canônica de $\mathbb{Z}^n$. Estes são conhecidos como raízes simples.

A transformação linear $\sigma_i : \mathbb{Z}^n \to \mathbb{Z}$ definida por $\sigma_i(x) = x - q_{\mathbb{Z}}(x, e_i) \cdot e_i$ é chamada **reflexão** em i, para $i = 1, \ldots, n$. Esta preserva a forma bilinear, ou seja, $q_{\mathbb{Z}}(\sigma_i(x), \sigma_i(y)) = q_{\mathbb{Z}}(x, y)$. As reflexões foram utilizadas por Dean e De La Peña para encontrar todas as raízes positivas de $q_{\mathbb{Z}}$ e identificar o tipo da forma unitária (veja em [10, 12]).

Pelo resultado de Drozd [17], se uma forma unitária $q_{\mathbb{Z}}$ é fracamente positiva, então $q_{\mathbb{Z}}$ têm um conjunto finito Σ_q^1 de raízes positivas. Durante um certo tempo, acreditava-se que algumas formas inteiras fracamente não-negativas poderiam ter um conjunto infinito de raízes positivas. No entanto, pelo seguinte resultado de Ovsienko, pôde-se identificar um limite superior para os vetores que caracterizam este tipo de forma unitária.

Teorema 3.1.3 *[38] Seja $q_{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$ uma forma unitária. Se $q_{\mathbb{Z}}$ é crítica e $n \geqslant 3$, então $q_{\mathbb{Z}}$ é fracamente não-negativa e existe uma única raiz nula $z \in \mathbb{Z}^n$ tal que z é positivo, chamada de vetor crítico. Um vetor crítico z têm valores $0 \leqslant z(i) \leqslant 6$, para $i = 1, \ldots, n$.*

O Teorema 3.1.4 é usado para verificar se uma forma unitária $q_{\mathbb{Z}}$ é fracamente não-negativa e foi implementado no Algoritmo 3.1. As condições apresentadas neste teorema permitem identificar de forma precisa o conjunto de raízes positivas Σ_q^1 (veja no Item (a)) e o conjunto de raízes nulas Σ_q^0 (veja no Item (b)), de uma forma crítica. As demonstrações destes resultados podem ser vistas em [12] (Seção 1.3 e Seção 1.5).

Teorema 3.1.4 *[12] Considere uma forma bilinear $q_{\mathbb{Z}}(-,-)$ associada a uma forma unitária $q_{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$. Sejam i, v tais que $1 \leqslant i \leqslant n$ e $0 \leqslant v \in \mathbb{Z}^n$. Então,*

(a) Se $q_{\mathbb{Z}}$ é fracamente não-negativa e $v \in \Sigma_q^1$, então $q_{\mathbb{Z}}(v, e_i) > -3$;

(b) Seja $q_{\mathbb{Z}}$ uma forma unitária crítica com $n \geqslant 3$. Seja u tal que $0 \neq u \in \Sigma_q^0$. Então existe um vetor $v \in \Sigma_q^1$ e um índice j tal que $q_{\mathbb{Z}}(v, e_j) = -2$ e $u = v + e_j$.

Para identificar formas inteiras fracamente positivas, De La Peña utilizou o resultado de Ovsienko (veja Teorema 3.1.3) para determinar um limite superior para os vetores considerados no reconhecimento de formas inteiras fracamente positivas. Um outro ponto importante utilizado por De La Peña é o já descrito no item (b) do Teorema 3.1.4 que caracteriza uma forma unitária crítica. O algoritmo de De La Peña para formas inteiras fracamente positivas pode ser visto em [10] (§4, Seção 4.3).

3.2 As Soluções Atuais

3.2.1 Formas Inteiras Fracamente Não-Negativas

As álgebras de caminho KQ, onde Q é um *quiver* conexo e acíclico cujo grafo subjacente é um grafo euclidiano (veja na Figura 3.3) são conhecidas como álgebras de tipo euclidiano (veja mais detalhes sobre este tipo de álgebra em [42]). Os grafos Euclidianos são exemplos de formas inteiras fracamente não-negativas.

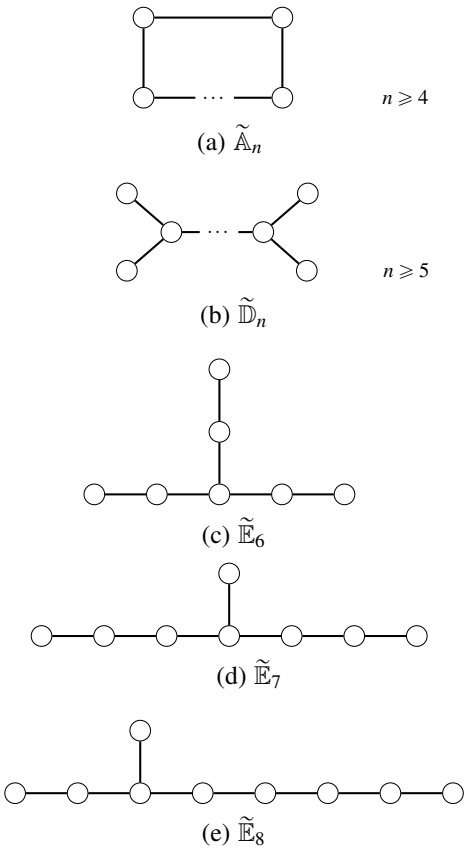


Figura 3.3: *Os grafos Euclidianos são exemplos de formas inteiras fracamente nãonegativas.*

O Algoritmo 3.1 é utilizado para verificar se uma forma unitária é fracamente nãonegativa. Este algoritmo gera todas as raízes positivas usando reflexões e, com base

nas condições descritas na Subseção 3.1.2, faz testes para verificar casos específicos que não caracterizam formas inteiras fracamente nãonegativas. É importante ressaltar que este algoritmo foi desenvolvido por Dean e De La Peña [12] e nosso objetivo é detalhar o funcionamento do algoritmo e identificar a complexidade de tempo[2] do mesmo.

No Algoritmo 3.1, A é uma matriz $n \times n$ da forma unitária $q_{\mathbb{Z}}$, Σ_q^1 é uma lista com raízes positivas de $q_{\mathbb{Z}}$, I é uma lista com as raízes simples de $q_{\mathbb{Z}}$ (base canônica de $q_{\mathbb{Z}}$) sendo que, no início, Σ_q^1 e I são idênticas. Para cada raiz positiva $v \in \Sigma_q^1$ e cada raiz simples $e \in I$ (linhas 3 e 4), o algoritmo avalia a forma unitária (linha 5), tenta gerar novas raízes positivas (linha 19) e, caso consiga, inclui as novas raízes em Σ_q^1 (linha 27).

As condições descritas no Teorema 3.1.4 foram usadas nas linhas 6 a 12 do Algoritmo 3.1 e o processo termina se este falhar em algum destes testes. Se o processo não falhar, o algoritmo tenta gerar todas as raízes positivas através de reflexões com vetores e_i da base canônica (linha 19) e o processo é repetido considerando as novas raízes encontradas. Finalmente, se o processo não falhar em nenhum teste e não encontrar mais raízes positivas através das reflexões, o algoritmo termina retornando *Verdadeiro*, o que significa que a forma unitária é fracamente nãonegativa.

No laço mais externo do Algoritmo 3.1 (linha 2), para cada elemento em Σ_q^1, o algoritmo pode criar até n raízes positivas a mais. As novas raízes também devem ser consideradas e, para cada uma, pode-se também criar n raízes positivas a mais. Isto induziria a um número infinito de raízes positivas. No entanto, desde que um vetor crítico v têm valores $0 \leqslant v_i \leqslant 6$, para $i = 1, \ldots, n$ (veja Teorema 3.1.3), o algoritmo considera 6^n vetores. Portanto, o Algoritmo 3.1 é uma solução exponencial no pior caso.

Em [12] (Seção 2.2), os autores mostraram que a forma unitária é fracamente nãonegativa se e somente se o Algoritmo 3.1 nunca pára (ou não falha nos testes nas linhas 6 a 12), mas termina (gera todas as raízes positivas). De forma geral, o algoritmo só pára se o Item (a) no Teorema 3.1.4 não é satisfeito para algum $v \in \Sigma_q^1$ ou, a condição no Item (b) é satisfeita. Se o algoritmo termina, então para todo vetor crítico $u \in \Sigma_q^0$, $u \cdot A \geqslant 0$ e $q_{\mathbb{Z}}$ é fracamente nãonegativa.

[2] Talvez por não se tratar de uma publicação na área de complexidade de algoritmos, Dean e De La Peña não deram informações relacionadas à complexidade de tempo do algoritmo apresentado.

Algoritmo 3.1: FNN_TESTE(A)

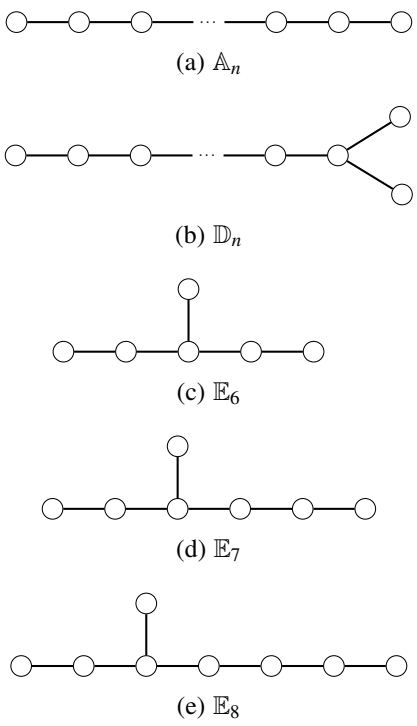
Entrada: A = matriz $n \times n$ da forma unitária $q_{\mathbb{Z}} \neq 0$

Saída: Resposta se $q_{\mathbb{Z}}$ é fracamente nãonegativa ou não (Verdadeiro ou Falso).

```
1  Σ_q^1 ← I ← matriz identidade
2  enquanto (Σ_q^1 ≠ ∅) faça
3      v ← vetor em Σ_q^1                          // v é uma raiz positiva de Σ_q^1
4      para cada (e ∈ I) faça
5          q_ℤ ← v·A·e^t                           // Avalia a forma unitária q_ℤ
6          se (q_ℤ ⩽ −3) então
7              retorna Falso                       // q_ℤ falhou para alguma raiz positiva
8          se (q_ℤ = −2) então
9              t ← (v + e)·A
10             para (i ← 1 ... n) faça
11                 se (t(i) < 0) então
12                     retorna Falso               // q_ℤ falhou para alguma raiz positiva
13         se (q_ℤ = −1) então
14             sair ← 0
15             para (i ← 1 ... n) faça
16                 se (v(i) ⩾ 6) então
17                     sair ← 1
18             se (sair = 0) então
19                 σ ← v − (v·A·e^t)·e             // Gera uma nova raiz positiva
20                 add ← 1
21                 i ← 1
22                 enquanto (i ⩽ n e add = 1) faça
                       /* Verifica se todo elemento em σ é maior que zero      */
23                     se (σ(i) < 0) então
24                         add ← 0
25                     i ← i + 1
26                 se (add = 1) então
27                     Σ_q^1 ← [Σ_q^1, σ]          // Inclui σ em Σ_q^1
28 retorna Verdadeiro                              // q_ℤ não falhou para todas as raízes positivas
```

3.2.2 Formas Inteiras Fracamente Positivas

Os grafos Dynkin, ou diagramas Dynkin (veja na Figura 3.4), foram utilizados por Gabriel [21] para mostrar que uma álgebra (ou *quiver* relacionado) é de representação finita se e somente se é um grafo Dynkin. Estes grafos são exemplos de formas inteiras fracamente positivas.



(a) $\mathbb{A}_n$

(b) $\mathbb{D}_n$

(c) $\mathbb{E}_6$

(d) $\mathbb{E}_7$

(e) $\mathbb{E}_8$

Figura 3.4: *Os grafos Dynkin, apresentados por Gabriel [21], são exemplos de formas inteiras fracamente positivas.*

O Algoritmo 3.1, com as condições descritas na Subseção 3.1.2, foi utilizado por De La Peña (veja em [10] §4, Seção 4.3) para verificar se uma forma unitária é fracamente positiva (mostramos na Seção 3.3.2 que esta é uma solução polinomial). Uma outra opção para o reconhecimento de formas inteiras fracamente positivas foi obtida através dos menores principais da matriz associada à forma unitária. Para isto, precisamos destacar alguns fatos da álgebra linear.

A matriz $A^{(ij)}$ é obtida da eliminação da i-ésima linha e j-ésima colunas de uma matriz A. O ij-ésimo menor principal de A é o determinante de $A^{(ij)}$. O determinante de A pode ser recursivamente definido em termos de seus menores principais. A matriz $ad(A)$ na qual a entrada (i, j) é $(-1)^{i+j} \cdot det\ A^{(ij)}$, é chamada de matriz adjunta de A. Esta têm a propriedade $A \cdot ad(A) = (det\ A) \cdot I_n = ad(A) \cdot A$.

Teorema 3.2.1 *[10] (Cap. 3, Seção 1.2) Seja $q_{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$ uma forma unitária e A a matriz simétrica associada. Se $q_{\mathbb{Z}}$ é fracamente positiva, então para toda submatriz principal B de A, $det\ B > 0$ ou $ad(B)$ não é positiva (isto é, têm uma entrada nãonegativa).*

O Teorema 3.2.1 foi usado por De La Peña para verificar se uma forma unitária é fracamente positiva. Neste caso, existem $\begin{pmatrix} n \\ k \end{pmatrix}$ menores principais de tamanho k em A, portanto, existem $2^n - 1$ menores principais e esta solução é claramente exponencial.

3.3 Uma Solução Polinomial

Esta seção mostra como verificar se uma forma unitária é fracamente nãonegativa ou fracamente positiva através de uma algoritmo polinomial que implementa estratégias baseadas em grafos.

3.3.1 Fracamente Nãonegativa Polinomial

O Teorema 3.3.1 e o Corolário 3.3.1 dão suporte ao algoritmo polinomial para o reconhecimento de formas inteiras fracamente nãonegativas.

Teorema 3.3.1 *[10] Seja $q_{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$ uma forma unitária com $n \geqslant 3$, então $q_{\mathbb{Z}}$ é fracamente nãonegativa se e somente se não existem restrições hipercríticas q^J.*

Corolário 3.3.1 *Uma forma unitária $q_{\mathbb{Z}}$ é fracamente nãonegativa se e somente se toda restrição q^J a um subconjunto J de cardinalidade 9 é fracamente nãonegativa.*

Prova. Todas as formas inteiras hipercríticas foram classificadas por Unger [48] e independentemente por Hoenhe (a maioria destas podem ser vistas em [51] e algumas relacionadas a *Tits forms* em [10]). Um ponto importante é que todas as formas inteiras na lista de Unger têm 9 ou menos vértices. Portanto, se todas as restrições q^J de subconjuntos com 9 vértices são fracamente nãonegativas, então $q_{\mathbb{Z}}$ é fracamente nãonegativa. □

Desta forma, podemos utilizar um algoritmo para testar todo *subquiver* com 9 vértices do *quiver* Q de $q_{\mathbb{Z}}$ (conforme Teorema 3.1.2). Desde que têm um número fixo de vértices, testar cada *subquiver* é constante e existem $\binom{n}{9}$ destes *subquivers* - um polinômio de grau 9 em n. Isto mostra que o problema de decidir se $q_{\mathbb{Z}}$ é fracamente nãonegativa está em P. O Algoritmo 3.2 descreve este processo. Neste caso, o resultado do Algoritmo 3.1 é *Falso* quando q^J não é fracamente nãonegativa, ou *Verdadeiro* quando q^J é fracamente nãonegativa (conforme Teorema 3.1.2).

Algoritmo 3.2: FNN_POLINOMIAL(A)

Entrada: A = matriz $n \times n$ da forma unitária $q_{\mathbb{Z}} \neq 0$.

Saída: Resposta se $q_{\mathbb{Z}}$ é fracamente nãonegativa ou não (Verdadeiro ou Falso).

```
1 L^J ← todos os subconjuntos q^J com 9 vértices de A.
2 para cada (q^J ∈ L_J) faça
                                   // testa cada q^J com o Algoritmo 3.1
3     se (FNN_TESTE(q^J) = Falso) então
4         retorna Falso            // q_Z falhou para algum q^J de A
5 retorna Verdadeiro               // q_Z não falhou para todos q^J de A
```

Para gerar todas as combinações com 9 vértices, o Algoritmo 3.2 têm complexidade de tempo polinomial ($O(n^9 \cdot c)$), onde c é uma constante usada para testar cada restrição q^J. Todas as restrições hipercríticas na lista de Unger são conectadas por arestas negativas. Assim, desde que todo *subquiver* hipercrítico está contido em um *subquiver* conexo com 9 vértices, podemos verificar se $q_{\mathbb{Z}}$ é fracamente nãonegativa testando todos os *subquivers* conexos de 9 vértices. Portanto, consideramos que toda entrada é conexa.

Isto justifica uma importante mudança no Algoritmo 3.2. Quando se estiver computando os *subquivers* conexos de $q_{\mathbb{Z}}$, é suficiente definir "conexo" somente usando arestas negativas. Na prática, isto resulta em menos *subquivers* a serem considerados porque, em geral, o *quiver* associado à forma unitária têm um grau pequeno para todos os vértices. Um exemplo de algoritmo que gera todas os *subquivers* conexos pode ser visto em [34].

Embora o Algoritmo 3.2 tenha complexidade polinomial, para *quivers* com quantidade muito grande de vértices (por exemplo, acima de 300 vértices[3]), também deverá ser considerada uma quantidade muito grande de *subquivers*. Neste caso, sugerimos utilizar uma implementação paralela do Algoritmo 3.2 para testar todos os *subquivers* de forma mais rápida.

3.3.2 Fracamente Positiva Polinomial

De La Peña [10] (§4, Seção 4.3) apresentou um algoritmo para verificar se uma dada forma unitária é fracamente positiva. A estratégia é similar à descrita no Algoritmo 3.1, no entanto, com testes diferentes. Pelo resultado de Drozd [17], uma forma unitária fracamente positiva $q_{\mathbb{Z}}$ têm um conjunto finito de raízes positivas Σ_q^1. No entanto, pelo Corolário 3.3.2, podemos concluir que o número de raízes positivas em uma forma unitária fracamente positiva deve ser polinomial.

[3] Note que este tipo de situação não é comum na teoria de representação de álgebra.

Corolário 3.3.2 *Seja Q o quiver da forma unitária $q_{\mathbb{Z}}$, associado à álgebra Λ. Se $q_{\mathbb{Z}}$ é fracamente positiva então $q_{\mathbb{Z}}$ têm um número polinomial de raízes positivas.*

Prova. Observação importante: desde que esta tese têm uma abordagem computacional, não vamos detalhar as notações algébricas utilizadas nesta prova. Para mais informações, recomendamos que os leitores vejam o Apêndice A e [3, 4, 10, 21].

Seja $\Lambda = k[Q]/I$ uma k-álgebra de dimensão finita, básica e conexa. De acordo com [10] (Capítulo II, §2, Teorema 2.3), se $q_{\mathbb{Z}}$ é fracamente positiva então $q_{\mathbb{Z}}$ é de representação finita e, existe uma bijeção $X \mapsto dimX$ entre isoclasses de módulos-Λ não-decomponíveis $i(\Lambda, n)$ e raízes positivas de $q_{\mathbb{Z}}$. O problema com componentes projetivos ocorre quando existem ciclos em um *quiver*. Podemos escolher uma orientação na qual não existem ciclos em Q. A álgebra é então chamada triangular e satisfaz as condições requeridas pelo teorema de De La Peña. Agora, de acordo com [3] (Proposição I e II, página 325), uma álgebra é de representação finita se e somente se existe um polinômio $P \in \mathbb{R}[x]$, tal que $i(\Lambda, n) \leqslant P(n)$ para todo $n \in \mathbb{N}$. Portanto, $i(\Lambda, n)$ é limitado superiormente e $q_{\mathbb{Z}}$ têm um número polinomial de raízes positivas. □

Um ponto crítico com relação à solução apresentada por De La Peña é que o grau do polinômio $P(n)$ é definido a partir da dimensão da álgebra relacionada e pode ser muito grande. Uma solução alternativa para o reconhecimento de formas inteiras fracamente positivas faz uso da Busca em Profundidade (do inglês *Depth First Search - DFS*). Para mais informações sobre DFS veja veja Apêndice A e [18].

Decidir se uma forma unitária é fracamente positiva é equivalente a excluir todos os subgrafos que são restrições críticas descritos por von Höhne [50]. Existem 1717 grafos críticos de tipo $\widetilde{\mathbb{E}}_8$, 142 de tipo $\widetilde{\mathbb{E}}_7$, 17 de tipo $\widetilde{\mathbb{E}}_6$ e algumas famílias infinitas de tipo $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$.

Com relação ao número de vértices, restrições críticas e hipercríticas são similares, ou seja, exceto pelos grafos $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$, todas as restrições críticas também têm menos que 10 vértices. Assim, utilizamos o Algoritmo 3.3 para identificar se uma forma unitária $q_{\mathbb{Z}}$ é fracamente positiva. Neste algoritmo, foi implementada a busca em profundidade (através dos Algoritmos 3.4–3.9) para identificar as restrições críticas do tipo $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$ em $q_{\mathbb{Z}}$ (linha 1). Se nenhuma restrição for encontrada, utilizamos o algoritmo FP_TESTE[4] para verificar se algum dos *subquivers* com 9 vértices do *quiver* Q de $q_{\mathbb{Z}}$ falha no teste para fracamente positiva (linha 5). Se não falhar até esta etapa, $q_{\mathbb{Z}}$ é fracamente positiva, conforme Teorema 3.1.1, e o algoritmo retorna *Verdadeiro* (linha 7).

[4] Algoritmo de De La Peña para formas inteiras fracamente positivas [10] (§4, Seção 4.3).

Algoritmo 3.3: FP_POLINOMIAL(A)

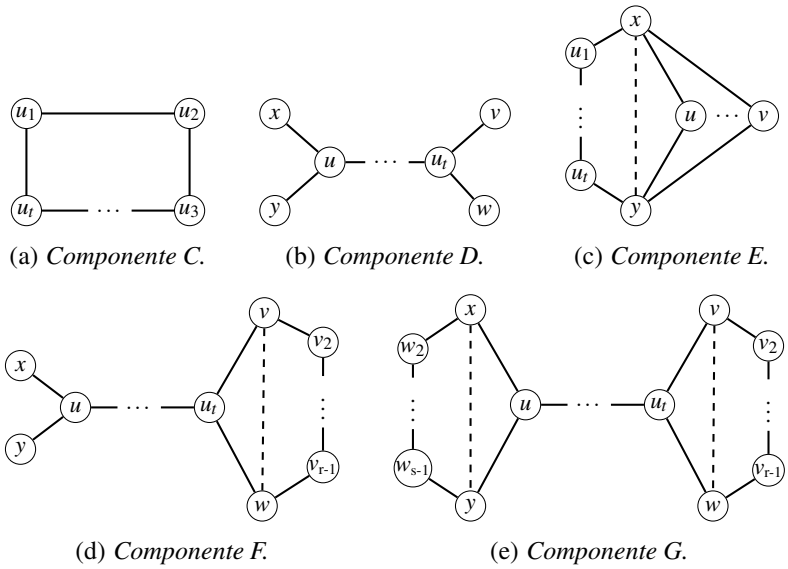
Entrada: A = matriz $n \times n$ da forma unitária $q_{\mathbb{Z}} \neq 0$.

Saída: Resposta se $q_{\mathbb{Z}}$ é fracamente positiva ou não (Verdadeiro ou Falso).

```
1  rest𝔸̃ₙ𝔻̃ₙ ← DFS_COMPS(Q, Adj_p, Adj_n)                    // Algoritmos 3.4-3.9
2  se (rest𝔸̃ₙ𝔻̃ₙ = Falso) então
3      L^J ← todos os subconjuntos q^J com 9 vértices de A.
4      para cada (q^J ∈ L^J) faça
           /* testa cada q^J com o algoritmo do De La Peña para formas inteiras
              fracamente positivas                                          */
5          se (FP_TESTE(q^J) = Falso) então
6              retorna Falso                          // q_Z falhou para algum q^J de A
7      retorna Verdadeiro                                  // q_Z é fracamente positiva
8  senão
9      retorna Falso                          // q_Z têm restrições do tipo 𝔸̃ₙ ou 𝔻̃ₙ
```

Os Algoritmos 3.4–3.9 utilizam a busca em profundidade para identificar as restrições críticas do tipo $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$ em $q_{\mathbb{Z}}$. As principais idéias nestes algoritmos são baseadas no artigo de Dias *et al.* [15] para identificação de ciclos sem corda.

As restrições críticas de tipo $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$ serão identificadas através dos componentes na Figura 3.5. O Componente C representa as restrições críticas de tipo $\widetilde{\mathbb{A}}_n$ e os Componentes D, E, F e G, as restrições críticas de tipo $\widetilde{\mathbb{D}}_n$.



(a) *Componente C.* (b) *Componente D.* (c) *Componente E.*

(d) *Componente F.* (e) *Componente G.*

Figura 3.5: *Componentes que representam restrições críticas do tipo $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$.*

O ponto de partida é o Algoritmo 3.6 (DFS_COMPS(Q, Adj_p, Adj_n)). A entrada Q é o quiver relacionado a $q_{\mathbb{Z}}$. As matrizes Adj_n e Adj_p são matrizes de adjacências de arestas positivas e de arestas negativas, respectivamente.

Inicialmente, este algoritmo pega todas as triplas $\langle x, u, y \rangle$ e tenta encontrar algum Componente C através do CC_VISIT na linha 7. Se não encontrar, tenta identificar o Componente D (através do DC_VISIT na linha 12), o Componente E (através do CC_VISIT na linha 22), o Componente F (através do DC_VISIT e depois CC_VISIT), e o Componente G (através do DC_VISIT e depois GC_VISIT). Se Q têm um dos componentes presentes na Figura 3.5, então o algoritmo retorna *Verdadeiro*. Caso contrário, retorna *Falso*.

Os Algoritmos 3.4 e 3.5 foram usados para bloquear e desbloquear vizinhos de um vértice v e evitar situações que não caracterizam componentes da Figura 3.5. Por exemplo, para evitar a construção de um Componente C ou D com arestas positivas entre os vértices no caminho. O conjunto Q_{1_p} é usado para verificar se uma aresta é positiva e o conjunto Q_{1_n} se é negativa.

Algoritmo 3.4: BLOQUEIA_VIZINHOS$(v, bloqueados)$

Entrada: Vértice $v \in Q$ e array global *bloqueados*.
Saída: Todos vizinhos de v bloqueados.

```
1 para cada (u ∈ Adj_n(v) ∪ Adj_p(v)) faça
2     bloqueados(u) ← bloqueados(u) + 1
```

Algoritmo 3.5: DESBLOQUEIA_VIZINHOS$(v, bloqueados)$

Entrada: Vértice $v \in Q$ e array global *bloqueados*.
Saída: Todos vizinhos de v bloqueados.

```
1 para cada (u ∈ Adj_n(v) ∪ Adj_p(v)) faça
2     bloqueados(u) ← bloqueados(u) − 1
```

Algoritmo 3.6: DFS_COMPS(Q, Adj_p, Adj_n)

Entrada: u = vértice inicial; Q = *quiver* relacionado a $q_{\mathbb{Z}}$; Adj_p = matriz de adjacência de arestas positivas; Adj_n = matriz de adjacência de arestas negativas.

Saída: Resposta se Q têm os Componentes C, D, F ou G (Verdadeiro/Falso).

```
1  para cada (u ∈ Q_0) faça
2      para cada ((x,y) ∈ Adj_n(u)) faça
3          se ((x,y) ∈ Q_{1_n}) então retorna Verdadeiro
4          se ((x,y) ∉ Q_{1_p}) então
5              BLOQUEIA_VIZINHOS(u)
6              p ← ⟨x,u,y⟩
7              CC_VISIT(p, bloqueados)
8              DESBLOQUEIA_VIZINHOS(u)
9              BLOQUEIA_VIZINHOS(x)
10             BLOQUEIA_VIZINHOS(y)
11             p ← ⟨u⟩
12             DC_VISIT(p, bloqueados, ⟨x,u,y⟩)
13             DESBLOQUEIA_VIZINHOS(x)
14             DESBLOQUEIA_VIZINHOS(y)
15         senão
16             BLOQUEIA_VIZINHOS(u)
17             bloqueados(u) ← 1
18             para cada (v ∈ Adj_n(x) ∩ Adj_n(y)) faça
19                 se (bloqueados(v) = 1) então
20                     p ← ⟨x,v,y⟩
21                     BLOQUEIA_VIZINHOS(v)
22                     CC_VISIT(p, bloqueados)
23                     DESBLOQUEIA_VIZINHOS(v)
24             bloqueados(u) ← 0
25             DESBLOQUEIA_VIZINHOS(u)
26             BLOQUEIA_VIZINHOS(x)
27             BLOQUEIA_VIZINHOS(y)
28             p ← ⟨u⟩
29             DC_VISIT(p, bloqueados, ⟨x,u,y⟩)
30             DESBLOQUEIA_VIZINHOS(x)
31             DESBLOQUEIA_VIZINHOS(y)
32 retorna Falso          // Q não têm os Componentes C, D, E, F e G.
```

Algoritmo 3.7: CC_VISIT(p, *bloqueados*)

Entrada: Caminho $p = \langle u_1, u_2, \ldots, u_t \rangle$ tal que p é um ciclo sem corda; array global *bloqueados*.

Saída: Resposta se Q têm os Componentes C, E ou F.

```
1  BLOQUEIA_VIZINHOS(u_t)
2  para (v ∈ Adj_n(u_t)) faça
3      se (bloqueados(v) = 1) então
4          se (u_1, v ∈ Q_{1_n}) então
5              retorna Verdadeiro                    // Q têm os Componentes C, E ou F.
6          p' ← ⟨p, v⟩
7          CC_VISIT(p', bloqueados)
8  DESBLOQUEIA_VIZINHOS(u_t)
```

Algoritmo 3.8: DC_VISIT($p, \mathit{bloqueados}, p_0$)

Entrada: Caminho $p = \langle u_1, u_2, \ldots, u_t \rangle$ tal que p é um ciclo sem corda; array global *bloqueados*; $p_0 = 0$ se $x, y \notin Q_{1_p}$ na tripla $\langle x, u, y \rangle$, caso contrário $p_0 = \langle x, u, y \rangle$.

Saída: Resposta se Q têm o Componente D.

```
1  BLOQUEIA_VIZINHOS(u_t)
2  para (v ∈ Adj_n(u_t)) faça
3      se (bloqueados(v) = 1) então
4          para (w ∈ Adj_n(u_t)) faça
5              se (bloqueados(w) = 1) então
6                  se ((v, w) ∉ Q_{1_p}) então
7                      se (p_0 = 0) então
8                          retorna Verdadeiro            // Q têm o Componente D.
9                      senão
10                         CC_VISIT(p_0, bloqueados)
11             se ((v, w) ∈ Q_{1_p}) então
12                 p' ← ⟨v, u_t, w⟩
13                 se (p_0 = 0) então
14                     CC_VISIT(p', bloqueados)
15                 senão
16                     GC_VISIT(p_0, p', bloqueados)
17         p' ← ⟨p, v⟩
18         DC_VISIT(p', bloqueados, p_0)
19 DESBLOQUEIA_VIZINHOS(u_t)
```

Algoritmo 3.9: GC_VISIT$(p, q, bloqueados)$

Entrada: Caminhos $p = \langle u_1, u_2, \ldots, u_t \rangle$ e $q = \langle v_1, v_2, \ldots, v_s \rangle$, tal que ambos são ciclos sem corda; array global *bloqueados*.

Saída: Resposta se Q têm o Componente G.

```
1 BLOQUEIA_VIZINHOS(u_t)
2 para (v ∈ Adj_n(u_t)) faça
3     se (bloqueados(v) = 1) então
4         se ((v, u_1) ∈ Q_{1,n}) então
5             CC_VISIT(q, bloqueados)
6         senão
7             p' ← ⟨p, v⟩
8             GC_VISIT(p', q, bloqueados)
9 DESBLOQUEIA_VIZINHOS(u_t)
```

Para mostrar que a identificação de restrições através da busca em profundidade funciona, destacamos cinco situações[5] que caracterizam as cinco restrições críticas em um *quiver* Q, são elas:

1. Q têm um subgrafo induzido pela sequência $p = \langle u_1, \ldots, u_t, u_1 \rangle$, que representa o Componente C. O algoritmo começa com alguma tripla $p = \langle x, u, y \rangle$ e aumenta p adicionando um vértice não-bloqueado no final de p. Eventualmente, o algoritmo encontra um vértice v que é adjacente a x e falha na linha 5 de CC_VISIT.
2. Q têm um grafo induzido com as triplas $\langle x, u, y \rangle, \langle v, u_t, w \rangle$ e o caminho $\langle u = u_1, \ldots, u_t \rangle$, que representa o Componente D. O algoritmo começa com $p = \langle u \rangle$ e, em DC_VISIT, aumenta p adicionando um novo vértice não-bloqueado v no final de p. Eventualmente, encontra dois vizinhos não-bloqueados v e w do vértice u_t no final de p, que não são vizinhos por arestas positivas. Então o algoritmo falha na linha 8 de DC_VISIT.
3. Q têm um grafo induzido com as triplas $\langle x, u, y \rangle, \langle x, v, y \rangle$, e um caminho $p = \langle x = u_1, u_2, \ldots, u_t = y \rangle$ com $(x, y) \in Q_{1_p}$, que representa o Componente E. O algoritmo começa com a tripla $\langle x, u, y \rangle$ e irá procurar por $\langle x, v, y \rangle$, na linha 18 de DFS_COMPS$(Q,\ Adj_p,\ Adj_n)$. Na linha 22, aciona CC_VISIT para encontrar o caminho $\langle u_1, u_2, \ldots, u_t \rangle$ e falha na linha 5.
4. Q têm um grafo induzido com as triplas $\langle x, u, y \rangle, \langle x, v, y \rangle$ e os caminhos $\langle u = u_1, u_2, \ldots, u_t \rangle$, $\langle v = v_1, v_2, \ldots, v_r = w \rangle$, com $(v, w) \in Q_{1_p}$, que representa o Componente F. O algoritmo começa com a tripla $\langle x, u, y \rangle$. Da mesma maneira que aconteceu com o Componente D, o algoritmo aumenta o caminho p (começando com

[5] Todos os caminhos e triplas citadas são conectadas por arestas negativas.

$p = \langle u \rangle$) até encontrar dois vizinhos não-bloqueados v e w, do vértices u_t no final de p. Neste caso, $(v,w) \in Q_{1_p}$ e o algoritmo DC_VISIT aciona CC_VISIT para $p = \langle v, u_t, w \rangle$. Este vai falhar depois que encontrar um caminho entre v e w, na linha 5.

5. Q têm um grafo induzido com as triplas $\langle x,u,y \rangle, \langle v,u_t,w \rangle$, os caminhos $\langle u = u_1, u_2, \ldots, u_t \rangle$, $\langle v = v_1, v_2, \ldots, v_r = w \rangle$ e $\langle x = w_1, w_2, \ldots, w_s = y \rangle$ com (x,y) e $(v,w) \in Q_{1_p}$, que representa o Componente G. O algoritmo começa com a tripla $\langle x,u,y \rangle$. Desde que alcança a linha 24, não têm o Componente E. Da mesma forma que aconteceu com o Componente D, o algoritmo aumenta o caminho p começando com $(p = \langle u \rangle)$, até encontrar dois vizinhos não-bloqueados v e w do vértice u_t no final de p. Neste caso, $(v,w) \in Q_{1_p}$, aciona GC_VISIT e encontra os caminhos $\langle v_1, v_2, \ldots, v_r \rangle$ e $\langle w_1, w_2, \ldots, w_s \rangle$. O algoritmo falha na linha 5 de CC_VISIT.

3.3.3 Identificando Casos Conhecidos

A estratégia de encontrar e testar todos os *subquivers* conexos com 9 vértices q^J de um *quiver* Q, descrita na Subseção 3.3.1, revelou novas estratégias que não poderiam ser aplicadas ao *quiver* original Q. Ou seja, a partir do momento que passamos a considerar *quivers* com 9 vértices, identificamos que muitos destes são *quivers* "simples" (caminhos ou ciclos) e com grande possibilidade de serem isomórficos entre si. Desta forma, através do grau máximo[6] de cada *subquiver* de 9 vértices em Q, identificamos os *subquivers* que são relacionados a formas inteiras fracamente positivas e, consequentemente, não precisam ser testados.

Corolário 3.3.3 *Seja Q um quiver que não contém os Componentes C, D, E, F e G e q^J um subquiver conexo com 9 vértices de Q. Se o grau máximo em q^J é 2, então q^J é um subquiver relacionado a uma forma unitária fracamente positiva.*

Prova. Vamos supor que todos os vértices em q^J têm grau 2 exceto dois vértices, que têm grau 1 e estão nas extremidades de q^J. Neste caso, q^J é um caminho, ou um grafo Dynkin do tipo $\mathbb{A}_n$, e é relacionado a uma forma unitária fracamente positiva. Uma outra possibilidade é todos os vértices terem grau 2. Neste caso, q^J é um ciclo, ou um grafo euclidiano do tipo $\widetilde{\mathbb{A}}_n$. Os grafos $\widetilde{\mathbb{A}}_n$ são relacionados a formas inteiras fracamente nãonegativas. No entanto, desde que Q não têm um Componente C, q^J só pode ser um ciclo com corda positiva e é relacionada a uma forma unitária fracamente positiva. □

[6] Ao computar o grau máximo do *subquiver*, é necessário considerar somente arestas negativas.

Corolário 3.3.4 *Seja Q um quiver que não contém os Componentes C, D, E, F e G. Seja q^J um subquiver conexo com 9 vértices de Q. Se todos os vértices em q^J têm grau 2, exceto dois vértices v_1, v_2 com grau 3 (cada um com dois vizinhos de grau 1), então q^J é um subquiver relacionado a uma forma unitária fracamente positiva.*

Prova. Vamos supor que todos os vértices em q^J têm grau 2 e existe um único vértice v com grau 3 (com dois vizinhos de grau 1). Desta forma, q^J é um grafo Dynkin do tipo $\mathbb{D}_n$ e é relacionado a uma forma unitária fracamente positiva. Se existem dois vértices v_1, v_2 com grau 3 (cada um com dois vizinhos de grau 1), então q^J é um grafo Euclidiano do tipo $\widetilde{\mathbb{D}}_n$ relacionado a uma forma unitária fracamente nãonegativa. No entanto, desde que Q não têm um Componente D, q^J têm pelo menos uma corda positiva e é relacionado a uma forma unitária fracamente positiva. □

3.4 Experimentos Numéricos

Os resultados apresentados nesta seção foram conseguidos através de uma implementação Matlab em um computador com Intel Core i5 1.3GHz e 4GB RAM. O objetivo foi comparar os resultados da execução do Algoritmo 3.1 (FNN_TESTE) com os resultados da execução Algoritmo 3.2 (FNN_POLINOMIAL), no reconhecimento de formas inteiras fracamente nãonegativas. Ainda mais, para comparar os resultados do Algoritmo de De La Peña para formas inteiras fracamente positivas (FP_TESTE)[7] com os resultados do Algoritmo 3.3 (FP_POLINOMIAL), no reconhecimento de formas inteiras fracamente positivas.

As subseções seguintes descrevem dois tipos de experimentos realizados. O primeiro tipo foi feito com grafos Dynkin e grafos Euclidianos. Estes estão diretamente relacionados com importantes álgebras. O segundo tipo foi feito com *quivers* construídos de maneira aleatória. Este tipo de *quiver* não têm relação com importantes álgebras, entretanto, é uma forma interessante de analisar o comportamento dos algoritmos com *quivers* gerais.

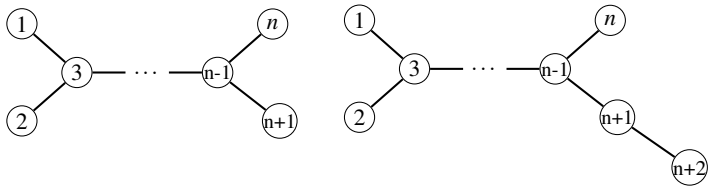
3.4.1 Experimentos com grafos Dynkin

A Figura 3.6(a) é um exemplo de grafo[8] Euclidiano do tipo $\widetilde{\mathbb{D}}_n$ e estes estão relacionados a formas inteiras fracamente nãonegativas. A Figura 3.6(b) descreve o

[7] Este algoritmo pode ser visto em [10] (§4, Seção 4.3).
[8] Os números dos vértices somente representam uma sequência numérica.

mesmo exemplo mas com um vértice e uma aresta a mais, correspondente a uma forma unitária não fracamente nãonegativa.

A Tabela 3.1 mostra os resultados conseguidos executando FNN_TESTE e FNN_POLINOMIAL com as formas inteiras da Figura 3.6 (formas inteiras fracamente nãonegativas e não fracamente nãonegativas), para diversos tamanhos de n (número de vértices no *quiver*). Nesta tabela, $N(G)$ descreve o número de vértices; $|\,\Sigma_q^1\,|$ o número de raízes positivas; *tempo* o tempo em segundos; $|\,sub\,|$ o número de subgrafos; e *subgrafos* são os subgrafos que falharam durante a execução de FNN_POLINOMIAL.



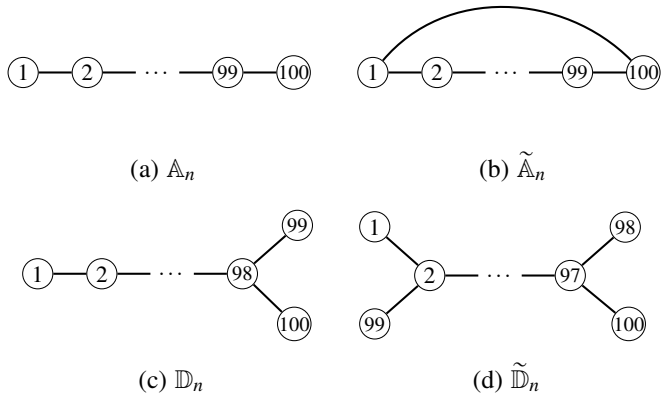
(a) *fracamente nãonegativa* (b) *não fracamente nãonegativa*

Figura 3.6: *Exemplo de forma unitária que é fracamente nãonegativa (a) e que não é fracamente nãonegativa (b).*

$N(G)$	Fracamente nãonegativas				(NÃO) Fracamente nãonegativas			
	FNN_TESTE		FNN_POLINOMIAL		FNN_TESTE		FNN_POLINOMIAL	
	$\|\,\Sigma_q^1\,\|$	tempo	$\|\,sub\,\|$	tempo	$\|\,\Sigma_q^1\,\|$	tempo	*subgrafos*	tempo
19|20	612	0.16	12	0.34	2754	2.10	{10, 11,...,20}	0.50
39|40	2812	3.02	32	0.86	3281	8.23	{30, 31,...,40}	1.05
59|60	2366	3.10	43	1.05	2177	5.47	{50, 51,...,50}	1.56
79|80	3186	9.70	63	1.70	4046	38.80	{70, 71,...,80}	2.03
99|100	4806	33.20	83	2.19	4646	65.34	{90, 91,...,100}	3.05

Tabela 3.1: *Resultados conseguidos executando* FNN_TESTE *e* FNN_POLINOMIAL *com as formas inteiras da Figura 3.6.*

A Tabela 3.2 mostra os resultados com os grafos Dynkin ($\mathbb{A}_n$, $\mathbb{D}_n$) e com grafos Euclidianos ($\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$), para $n = 100$, descritos na Figura 3.7. Nesta tabela, *comp* descreve o componente identificado pelos Algoritmos 3.4–3.9.



(a) $\mathbb{A}_n$ (b) $\widetilde{\mathbb{A}}_n$

(c) $\mathbb{D}_n$ (d) $\widetilde{\mathbb{D}}_n$

Figura 3.7: *Grafos Dynkin ($\mathbb{A}_n$, $\mathbb{D}_n$) e Grafos Euclidianos ($\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$), para $n = 100$*

$n = 100$	Fracamente nãonegativas				Fracamente Positivas			
	FNN_TESTE		FNN_POLINOMIAL		FP_TESTE		FP_POLINOMIAL	
	$\lvert \Sigma_q^1 \rvert$	tempo	$\lvert sub \rvert$	tempo	$\lvert \Sigma_q^1 \rvert$	tempo	comp	tempo
$\mathbb{A}_n$	5050	28.90	91	2.18	5050	27.48	-	0.84
$\widetilde{\mathbb{A}}_n$	9900	83.10	100	2.35	9900	85.30	C	0.34
$\mathbb{D}_n$	9900	106.46	92	2.24	9900	103.18	-	0.33
$\widetilde{\mathbb{D}}_n$	19404	318.18	93	2.25	19404	291.94	D	0.32

Tabela 3.2: *Resultados com os grafos Dynkin ($\mathbb{A}_n$ e $\mathbb{D}_n$) e com grafos Euclidianos ($\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$), para $n = 100$.*

Note que, apesar de pequena diferença no número de vértices, em grafos Euclidianos o número de raízes positivas Σ_q^1 pode ser bem diferente. Por exemplo, o grafo Euclidiano $\widetilde{\mathbb{D}}_n$ com 99 vértices na Tabela 3.1 têm 4806 raízes positivas e, na Tabela 3.2, o mesmo $\widetilde{\mathbb{D}}_n$ com 100 vértices têm 19404 raízes positivas.

A análise dos resultados na Tabela 3.1 mostrou que FNN_POLINOMIAL conseguiu identificar o tipo da forma unitária em um tempo bem menor que o tempo utilizado por FNN_TESTE. Sobretudo, no reconhecimento de formas inteiras fracamente positivas, os resultados com FP_POLINOMIAL foram consideravelmente menores. Isto porque a maioria das restrições foram identificadas com a estratégia DFS.

Os experimentos mostraram que a estratégia de testar todos os subgrafos com 9 vértices é mais eficiente que gerar todas as raízes positivas da forma quadrática relacionada. No entanto, para analisar o desempenho dos algoritmos com grafos gerais, fizemos experimentos com *quivers* gerados de forma aleatória.

3.4.2 Experimentos com *Quivers* Aleatórios

Os experimentos[9] com *quivers* criados através de uma distribuição aleatória de vértices e arestas foram feitos através da média de 20 testes sobre formas inteiras fracamente positivas com n vértices, da seguinte forma: o primeiro exemplo têm n arestas negativas e $2 \cdot n$ arestas positivas; o segundo $n+1$ e $2 \cdot n$; e o último $n+19$ arestas negativas e $2 \cdot n$ arestas positivas.

Para melhorar a confiabilidade da média calculada, o *quiver* com maior tempo foi desconsiderado. Isto foi necessário porque, para uma quantidade pequena de exemplos, os algoritmos utilizaram um tempo muito maior do que o tempo utilizado com os demais exemplos. Tais procedimentos também foram executados nos experimentos com formas inteiras que não são fracamente positivas, que são fracamente nãonegativas e que não são fracamente nãonegativas.

Para formas inteiras que são fracamente positivas, os resultados conseguidos com o método FP_POLINOMIAL foram bem melhores que os conseguidos com FP_TESTE (veja na Figura 3.8). Um ponto importante nestes resultados é que a maioria das restrições críticas foram identificas pelo método DFS na implementação dos Algoritmos 3.4–3.9.

Os resultados conseguidos com *quivers* aleatórios para formas inteiras que não são fracamente positivas ou fracamente nãonegativas, foram similares aos conseguidos com grafos Dynkin e grafos Euclidianos. Ou seja, os métodos FP_POLINOMIAL e FNN_POLINOMIAL foram mais rápidos que os métodos FP_TESTE e FNN_TESTE (veja nas Figuras 3.9 e 3.10).

[9] O leitor pode ver as tabelas com os dados coletados no Apêndice C, Seção C.2.

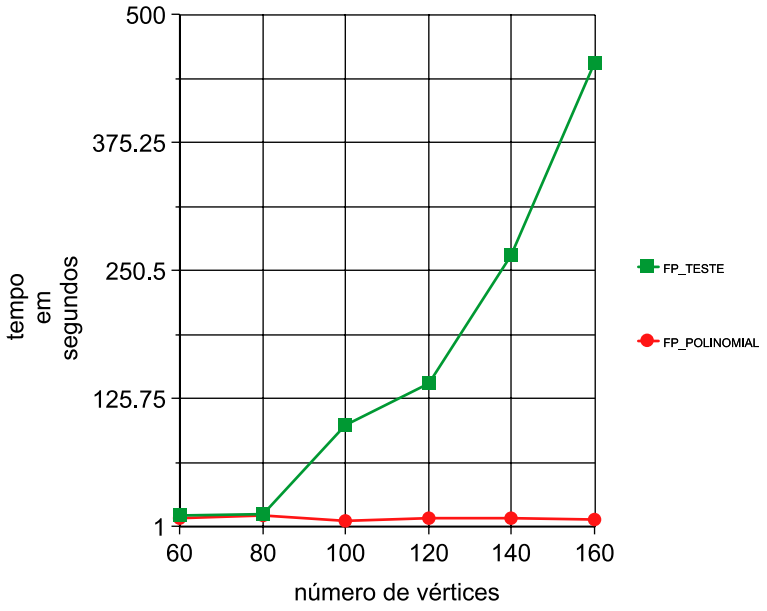


Figura 3.8: *Experimentos com quivers aleatórios para formas inteiras fracamente positivas.*

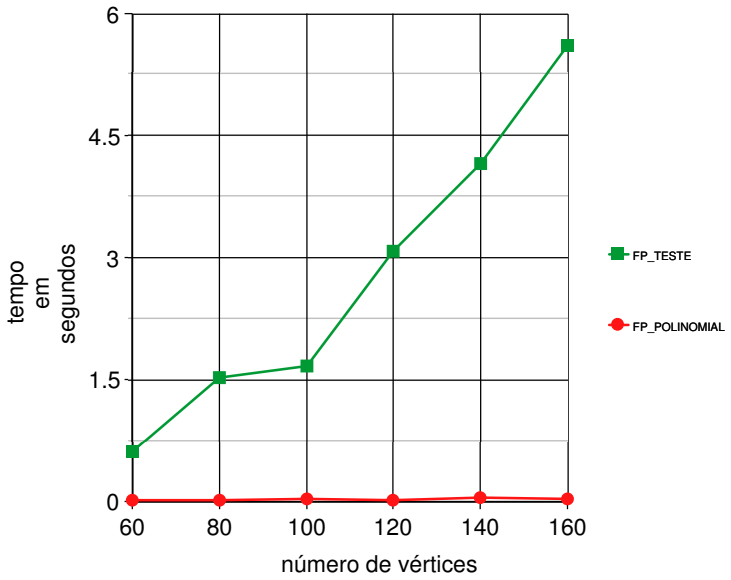


Figura 3.9: *Experimentos com quivers aleatórios para formas inteiras que não são fracamente positivas.*

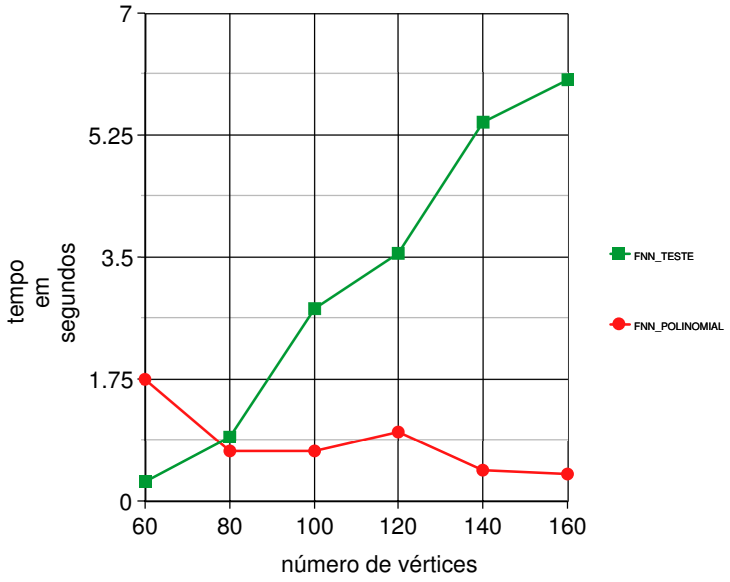


Figura 3.10: *Experimentos com quivers aleatórios para formas inteiras que não são fracamente nãonegativas.*

3.5 Considerações Finais

Os métodos desenvolvidos por Dean e De La Peña (FP_TESTE e FNN_TESTE)[10] funcionam muito bem para álgebras com *quivers* relacionados a grafos Dynkin com número pequeno de vértices. Entretanto, se o *quiver* têm um número grande de vértices, isto pode se tornar um problema.

Em geral, para valores grandes de n (*quivers* com 40 ou mais vértices), nossos algoritmos FP_POLINOMIAL e FNN_POLINOMIAL conseguiram resultados corretos e de maneira mais rápida que os algoritmos que geram todas as raízes positivas (FP_TESTE e FNN_TESTE), no reconhecimento de formas inteiras fracamente positivas e fracamente nãonegativas. Por outro lado, para *quivers* com 10 ou menos vértices, foram mais lentos na maioria dos exemplos testados.

Isto aconteceu porque, para alguns casos específicos (como em *quivers* com número reduzido de vértices), existe um número muito pequeno de raízes positivas. Entretanto, de acordo com os resultados apresentados, para *quivers* com um grande número de vértices (por exemplo, com 300 vértices ou mais) será impraticável gerar todas as raízes positivas.

Em uma análise de corretude, os algoritmos FP_POLINOMIAL e FNN_POLINOMIAL mostraram os mesmos resultados apresentados pelos algoritmos FP_TESTE e FNN_TESTE. Ainda mais, identificaram corretamente as restrições críticas e hipercríticas descritas por Unger e Von Höhne.

[10] Métodos que geram todas as raízes positivas da forma unitária $q_{\mathbb{Z}}$.

CAPÍTULO 4

Definitividade Através de Mutações

Uma outra possibilidade para se fazer a definitividade de formas inteiras é através de mutações. Fomin e Zelevinsky [20] desenvolveram um processo combinatorial recursivo (chamado de processo de mutação) para definir álgebras cluster. A teoria de álgebras cluster está relacionada a vários campos e, portanto, pode ser interpretada de diferentes maneiras. Aqui vamos destacar a descrição combinatorial do processo de mutação de matriz de troca e mutação de *quiver*, com o objetivo de mostrar que a definitividade de formas inteiras pode ser feita através de sequências de mutações com características específicas.

O processo de mutação de matrizes descreve ações que estão diretamente relacionadas a reflexões entre raízes positivas da forma inteira relacionada. A dinâmica do processo de mutação é encapsulada em uma matriz conhecida como matriz de troca T e, as raízes positivas são mantidas em uma submatriz $c(T)$ de T (conhecida como c-matriz de T). Assim, mostramos que, desde que sejam feitas todas as possíveis mutações na matriz de troca relacionada a uma forma inteira $q_{\mathbb{Z}}$, todas as raízes positivas serão geradas e poderão ser testadas, para se determinar o tipo de $q_{\mathbb{Z}}$. Sobretudo, várias estratégias podem ser usadas para simplificar este processo, permitindo que todas as raízes sejam encontradas sem a necessidade de se fazer todas as mutações possíveis.

O capítulo está organizado da seguinte forma: a Seção 4.1 descreve alguns conceitos básicos sobre o processo de mutação. Esta etapa inicial foi dividida em duas partes, a Subseção 4.1.1 destaca conceitos gerais com base nos artigos em [5, 20] e, da Subseção 4.1.2 em diante, consideramos mutações em vértices verdes a partir de um tipo de *quiver* específico. Isto nos permitiu garantir os resultados apresentados neste capítulo; na Seção 4.2 descrevemos, através de um algoritmo que utiliza a busca em largura - BFS (para mais informações sobre BFS veja Apêndice A e [18]), como pode ser feita a definitividade através de mutações. Por fim, na Seção 4.3 apresentamos algumas estratégias para simplificar a definitividade através de mutações e descrevemos uma forma de implementação das mesmas.

4.1 Conceitos Básicos

4.1.1 Processo de Mutação

A dinâmica do processo de mutação é encapsulada em uma matriz conhecida como matriz de troca. De forma geral, o processo de mutação começa com uma **semente**, que inclui um conjunto inicial de n geradores distintos (chamados variáveis cluster) e uma matriz de troca. Assim, a partir de um processo iterativo (chamado de mutação de matriz de troca), são produzidas as demais variáveis cluster. Aqui vamos destacar principalmente a descrição combinatorial do processo de mutação de matriz de troca e mutação de *quiver*. O leitor pode encontrar mais informações sobre álgebras cluster e variáveis cluster em [5, 20].

Definição 4.1 (Matriz de Troca) *Uma matriz de troca (do inglês exchange matrix) é uma matriz $T = (t_{ij}) \in M_{n,n+m}(\mathbb{Z})$ para $n,m \geqslant 0$ tal que a submatriz quadrada $T^0 = (t_{ij})_{1\leqslant i,j\leqslant n} \in M_n(\mathbb{Z})$ é anti-simetrizável, ou seja, existem inteiros positivos d_i tais que $d_i a_{ij} = -d_i a_{ji}$, para $1 \leqslant i,\ j \leqslant n$.*

Definição 4.2 (Mutação de Matriz de Troca) *Seja $T \in M_{n,n+m}(\mathbb{Z})$ uma matriz de troca. Então para qualquer $1 \leqslant k \leqslant n$, a mutação de T em direção a k é uma matriz de troca $\mu_k(T) = (t'_{ij}) \in M_{n,n+m}(\mathbb{Z})$ dada por:*

$$t'_{ij} = \begin{cases} -t_{ij} & \text{se } i = k \text{ ou } j = k, \\ t_{ij} + [t_{i,k}]_+ \cdot [t_{k,j}]_+ - [t_{i,k}]_- \cdot [t_{k,j}]_-, & \text{caso contrário.} \end{cases}$$

onde $[x]_+ = max(x,0)$ e $[x]_- = min(x,0)$, para qualquer $x \in \mathbb{Z}$.

A matriz de troca e o processo de mutação de matriz de troca podem ser representados através de *quivers*. Neste caso, um *quiver* congelado (do inglês *ice quiver*) é um *cluster quiver*,[1] que naturalmente tem uma matriz de troca associada, e a mutação de matriz de troca pode ser representada através da mutação de *quiver* congelado.

Definição 4.3 (*Quiver* Congelado) *Um quiver congelado $\widetilde{Q}$ é um par (Q,C) onde Q é um cluster quiver e $C \subset Q_0$ é um subconjunto possivelmente vazio de vértices conhecidos como vértices congelados, tal que não existem arestas entre eles.*

Para melhor representar os conjuntos de vértices não-congelados e congelados, vamos considerar um *quiver* congelado $\widetilde{Q} = Q_0 \cup C$, onde $Q_0 = \{1,\ldots,n\}$, $C = \{n+1,\ldots,m\}$. Sendo que os vértices não-congelados estão em Q_0 e os congelados em C.

[1] Um *cluster quiver* é um *quiver* sem laços e sem 2-ciclos orientados.

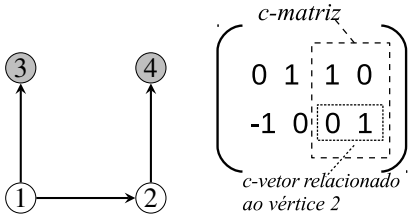
Definição 4.4 (Matriz de Troca de um *Quiver* Congelado) *Seja $\widetilde{Q}$ um quiver congelado. A matriz de troca $T(\widetilde{Q}) = (t_{ij}) \in M_{n,n+m}(\mathbb{Z})$ associada ao $\widetilde{Q}$ é definida por:*

$$t_{ij} = |\{i \to j \in Q_1\}| - |\{j \to i \in Q_1\}|.$$

O mapeamento $\widetilde{Q} \mapsto T(\widetilde{Q})$ induz uma bijeção do conjunto de quivers congelados ao conjunto de matrizes de troca.

Por questão de clareza, sempre que possível, vamos referenciar a matriz de troca de um *quiver* congelado simplesmente como T. A submatriz $c(T) = (t_{i,n+j})_{1 \leqslant i,j \leqslant n}$, da matriz de troca T, guarda informações importantes que descrevem a relação de cada vértice não-congelado em Q_0 com os vértices congelados em C. No caso, $c(T)$ é chamada **c-matriz** de T e, para qualquer vértice $i \in Q_0$, a i-ésima linha de $c(T)$ é chamada de **c-vetor** relacionado ao vértice i.

Exemplo 4.5 *A Figura 4.1(a) descreve um exemplo de quiver congelado $\widetilde{Q}$ e a Figura 4.1(b) a respectiva matriz de troca T. A matriz de troca T descreve a relação entre os vértices em $\widetilde{Q}$ ($t_{ij} = 0$, se não existem arestas entre os vértices i e j; $t_{ij} = x$, se existem x arestas de i para j; e $t_{ij} = -x$, se existem x arestas de j para i).*



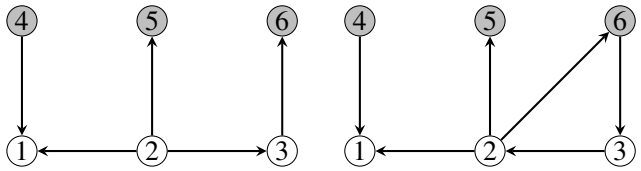
(a) $\widetilde{Q}$ (b) T

Figura 4.1: *Em (a) quiver congelado $\widetilde{Q}$ e em (b) a matriz de troca relacionada T.*

Definição 4.6 (Mutação de *Quiver* Congelado) *Seja $\widetilde{Q}$ um quiver congelado e $k \in Q_0$ um vértice não-congelado. A mutação de $\widetilde{Q}$ em k é definida como o quiver congelado $\mu_k(\widetilde{Q})$ obtido de $\widetilde{Q}$ aplicando as seguintes modificações:*

1. *Cada par $i \xrightarrow{a} k \xrightarrow{b} j$ em $\widetilde{Q}$ corresponde a $i \xrightarrow{ab^*} j$ em $\mu_k(\widetilde{Q})$;*
2. *Substituir $i \xrightarrow{a} k$ por $i \xleftarrow{a^*} k$;*
3. *Substituir $k \xrightarrow{b} j$ por $k \xleftarrow{b^*} j$;*
4. *Todos os 2-ciclos devem ser removidos.*

Exemplo 4.7 *A Figura 4.2(a) apresenta um exemplo de quiver congelado $\widetilde{Q}$, sendo que, os vértices congelados estão na cor cinza e os vértices não-congelados na cor branca. A Figura 4.2(b) descreve o quiver congelado $\mu_3(\widetilde{Q})$, definido a partir da mutação de $\widetilde{Q}$ no vértice* 3.



(a) $\widetilde{Q}$ (b) $\mu_3(\widetilde{Q})$

Figura 4.2: *Em (a) quiver congelado $\widetilde{Q}$ e em (b) quiver congelado $\mu_3(\widetilde{Q})$, definido a partir da mutação de $\widetilde{Q}$ no vértice 3.*

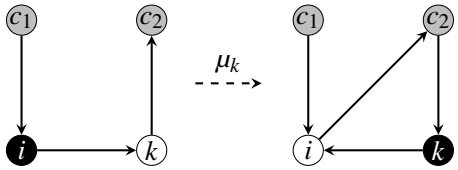
4.1.2 As Sequências de Mutações em Vértices Verdes

Dentro do processo de mutação de *quiver* congelado, existe um forma de coloração de vértices não-congelados que é usada para definir o conjunto de vértices habilitados a receberem mutações (chamados de vértices verdes). Os demais vértices são conhecidos como vértices vermelhos.

Definição 4.8 (Vértices Verdes e Vermelhos) *Seja $\widetilde{Q}$ um quiver congelado e $i \in Q_0$ um vértice não-congelado. O vértice i é considerado verde (resp. vermelho) se todas as arestas entre vértices congelados e i estão saindo de (resp. chegando em) i.*

Para um *quiver* congelado qualquer, nem sempre estas cores são bem definidas (veja no Exemplo 4.9). Veremos na Definição 4.12 que as cores dos vértices em um *quiver* congelado são bem definidas quando todas as mutações são feitas em vértices verdes a partir de um *quiver* específico, o *quiver* emoldurado.

Exemplo 4.9 *A Figura 4.3 descreve um quiver congelado em que a cor de um dos vértices não-congelados não é bem definida. Ou seja, após a mutação de $\widetilde{Q}$ em direção a k, a cor do vértice i em $\mu_k(\widetilde{Q})$ não é bem definida. Neste exemplo e nos demais exemplos a partir daqui, os vértices verdes serão representados na cor branca, os vermelhos na cor preta e os congelados na cor cinza.*[2]



$\widetilde{Q}$ $\mu_k(\widetilde{Q})$

Figura 4.3: *O quiver congelado $\mu_k(\widetilde{Q})$, resultante da mutação de $\widetilde{Q}$ em k, tem um vértice em que a cor não é bem definida.*

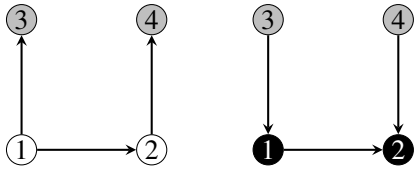
[2] As cores foram definidas desta forma para facilitar a identificação do tipo de vértice quando a impressão for feita em preto e branco.

Antes de definir uma sequência de mutações em vértices verdes, vamos destacar dois tipos de *quivers* congelados que são usados na caracterização das sequências verdes. Estes *quivers* aparecem quando analisamos a relação entre vértices congelados e vértices não-congelados.

Definição 4.10 **(*Quiver* Emoldurado e Co-emoldurado de Q)** *Seja Q um cluster quiver, um quiver Emoldurado $\widehat{Q}$ (resp. Co-emoldurado $\breve{Q}$) obtido a partir de Q é definido como segue:*

1. *Adicione um vértice congelado i' para cada vértice i em Q_0;*
2. *Adicione uma aresta $i \to i'$ (resp. $i' \to i$) para cada vértice i em Q_0.*

Exemplo 4.11 *A Figura 4.4(a) descreve um exemplo de quiver emoldurado e a Figura 4.4(b) um co-emoldurado.*



(a) $\widehat{Q}$ (b) $\breve{Q}$

Figura 4.4: *Em (a) um exemplo de quiver emoldurado $\widehat{Q}$ e em (b) um quiver co-emoldurado $\breve{Q}$.*

Desde que Q é um *cluster quiver*, ambos $\widehat{Q}$ e $\breve{Q}$ são *quivers* congelados. A matriz de troca de um *quiver* emoldurado é dada por $T(\widehat{Q}) = (t_{ij}) \in M_{n,2\cdot n}(\mathbb{Z})$. Mais adiante, mostraremos que estes *quivers* são únicos em uma sequência verde.

Definição 4.12 **(Classe de Mutação de um *Quiver*)** *Seja $\widehat{Q}$ um quiver emoldurado. A classe de mutação $Mut(\widehat{Q})$ é o conjunto de todos os possíveis quivers congelados obtidos através de sequências de mutações (em vértices verdes) a partir de $\widehat{Q}$.*

Note que uma classe de mutação é definida a partir de mutações em vértices verdes. Essa restrição é para garantir que os c-vetores, em uma c-matriz, tenham a propriedade de coerência de sinais (do inglês *sign coherence* [14]). Isto quer dizer que, para todo c-vetor na c-matriz, todas as entradas são nãonegativas ou são não-positivas.

No contexto de um *quiver* congelado, se um vértice $i \in Q_0$ é relacionado a um c-vetor que tem as entradas nãonegativas, então i é verde. Caso contrário, i é vermelho. Ainda mais, a propriedade de coerência de sinais garante que a cor de todo vértice não-congelado i seja bem definida. Isto é, todas as arestas entre vértices congelados e i, estão saindo de i ou estão chegando em i.

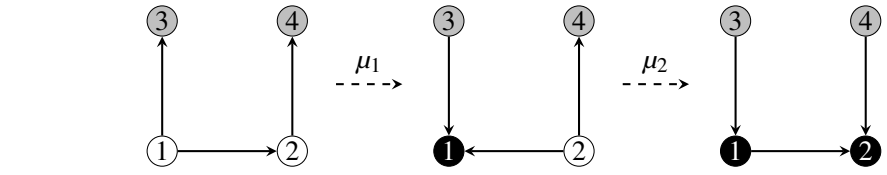
Teorema 4.1.1 *[5] Seja Q um cluster quiver e $\widetilde{Q} \in Mut(\widehat{Q})$. Então todo vértice não congelado em $\widetilde{Q}$ é verde ou vermelho.*

Tal propriedade é necessária na relação de c-vetores com as raízes positivas da forma inteira relacionada (veja mais detalhes em [8]). Portanto, sempre que nos referirmos a uma mutação, a mesma é em um vértice verde.

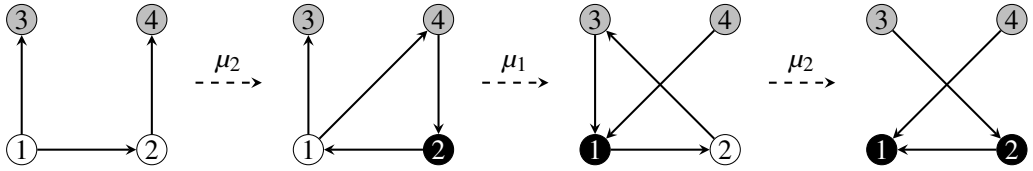
Definição 4.13 (Sequências Verdes) *Uma sequência $seq_v(i) = \langle i_1, \ldots, i_c \rangle \subset Q_0$ é considerada uma sequência verde se para todo $2 \leqslant k \leqslant c$, o vértice i_k é verde em $\mu_{i_{k-1}} \circ \cdots \circ \mu_{i_1}(\widehat{Q})$. O comprimento da sequência $seq_v(i)$ é denotado por $c(i)$.*

A sequência verde $seq_v(i) = \langle i_1, \ldots, i_c \rangle$ é chamada maximal, denotada $Seq_M(i)$, se todo vértice não-congelado em $\mu_i(\widetilde{Q})$ é vermelho, onde $\mu_i(\widetilde{Q}) = \mu_{i_c} \circ \cdots \circ \mu_{i_1}(\widehat{Q})$. Note-se que, uma sequência verde sempre começa com um *quiver* emoldurado e, se a sequência verde for maximal, sempre termina com um *quiver* co-emoldurado. Daqui em diante, sequências verdes maximais serão nomeadas simplesmente como sequências maximais.

Exemplo 4.14 *A Figura 4.5 mostra as sequências maximais com relação ao quiver congelado da Figura 4.1(a). Para este exemplo, existem duas sequências maximais, $\langle 1,2 \rangle$ e $\langle 2,1,2 \rangle$.*



(a) *Sequência maximal $\langle 1,2 \rangle$.*



(b) *Sequência maximal $\langle 2,1,2 \rangle$.*

Figura 4.5: *Sequências maximais com relação ao quiver congelado da Figura 4.1(a).*

Para a classe $Mut(\widehat{Q})$ de alguns *quivers* congelados, a quantidade de sequências maximais pode ser consideravelmente grande. No entanto, existem classes de mutações que não possuem sequências maximais. Na Tabela 4.1, apresentamos alguns exemplos de *quivers*, com os respectivos *quivers* congelados, as sequências maximais Seq_M, o número de sequências maximais $|Seq_M|$ e o comprimento máximo c_M em vértices das sequências maximais. Note que, o último *quiver* congelado na tabela não possui sequências maximais porque o comprimento máximo é infinito.

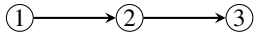
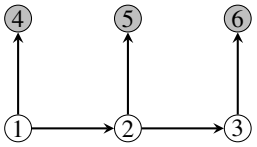
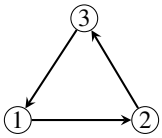
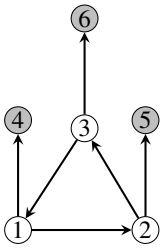
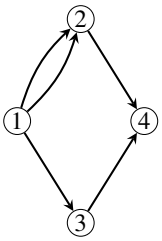
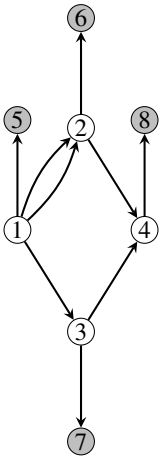
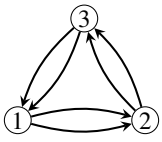
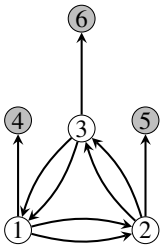
Q	$\widehat{Q}$	Seq_M	$\lvert Seq_M\rvert$	c_M
		(1, 2, 3) (1, 3, 2, 3) (2, 1, 2, 3) (2, 1, 3, 2) (2, 3, 1, 3, 2) (3, 1, 2, 3) (3, 2, 1, 2, 3) (3, 2, 1, 3, 2, 3) (3, 2, 3, 1, 2, 3)	9	6
		(1, 2, 3, 1) (1, 3, 1, 2, 1) (1, 3, 2, 1) (2, 1, 2, 3, 2) (2, 1, 3, 2) (2, 3, 1, 2) (3, 1, 2, 3) (3, 2, 1, 3) (3, 2, 3, 1, 3)	9	5
		(1, 2, 3, 4) (1, 2, 3, 4) (1, 2, 4, 3, 4) (1, 3, 2, 4) (1, 3, 4, 2, 4) (1, 4, 2, 3, 4, 2, 3) (1, 4, 2, 3, 4, 3, 2) (1, 4, 2, 4, 3, 2) (1, 4, 3, 2, 4, 2, 3) (1, 4, 3, 2, 4, 3, 2) ⋮ (4, 3, 4, 1, 3, 4, 2, 1)	48	8
		–	–	∞

Tabela 4.1: *Exemplos de quivers com os respectivos quivers congelados, sequências maximais Seq_M, número $\lvert Seq_M\rvert$ e comprimento máximo c_M das sequências maximais.*

Um outro ponto importante é que um *quiver* congelado pode ter sequências verdes finitas (ou sequências maximais) e sequências verdes infinitas. Por exemplo, o *quiver* congelado da Figura 4.6 possui três sequências maximais ($\langle 1,2,3\rangle$, $\langle 2,1,2,3\rangle$, $\langle 2,1,3,2\rangle$). No entanto, todas as sequências verdes iniciadas a partir do vértice 3 são infinitas. Veremos na Seção 4.3.1 como identificar quando a mutação em um vértice $k \in Q_0$ leva a uma sequência infinita.

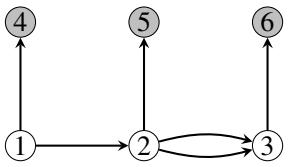


Figura 4.6: *Quiver congelado com sequências maximais e sequências infinitas.*

O Algoritmo 4.1 mostra como encontrar uma sequência maximal. Na linha 1, é definida a matriz de troca T, relacionada ao *quiver* emoldurado $\widehat{Q}$, a partir do *cluster quiver* Q. A lista Seq_M armazena os vértices da sequência maximal. A função SELECIONAR_VERTICE(T) seleciona um vértice verde de $\widetilde{Q}$ (a partir de T) e informa quando a sequência maximal deve ser finalizada (quando $v = 0$). A função MUTAÇÃO(T,v) faz a mutação da matriz T em v. O leitor pode ver as funções SELECIONAR_VERTICE(T) e MUTAÇÃO(T,v) no Apêndice D, Algoritmos D.2 e D.3 respectivamente.

A escolha do vértice verde para mutação em T tem o propósito de selecionar o vértice com menor número de arestas incidentes de vértices não-congelados. Esta condição direciona o processo para a construção da menor sequência maximal (veja o Teorema 4.1.2). Um algoritmo mais completo, que faz todas as mutações em vértices verdes e encontra todas as sequências maximais é apresentado na seção seguinte.

Algoritmo 4.1: MENOR_SM(Q)

Entrada: Q = *cluster quiver*.

Saída: Seq_M = sequência maximal com menor tamanho.

```
1 T ← [Q, identidade(Q)]          // Define a matriz de troca T a partir de Q
2 Seq_M ← ∅
3 v ←SELECIONAR_VERTICE(T)        // Seleciona um vértice (Algoritmo D.2)
4 enquanto (v ≠ 0) faça
5 |   T ←MUTAÇÃO(T, v)            // Faz a mutação μ_v(T) (Algoritmo D.3)
6 |   Seq_M ← ⟨Seq_M, v⟩          // Inclui v na lista Seq_M
7 |   v ← SELECIONAR_VERTICE(T)
8 fim
9 retorna Seq_M
```

Lema 4.15 *Seja $\widetilde{Q} \in Mut(\widehat{Q})$ um quiver congelado e $i \in Q_0$ um vértice não-congelado. Se a cor de i é verde, então nenhuma mutação em um vértice $k \in Q_0, k \neq i$, pode mudar a cor de i para vermelha.*

Prova. Desde que i é verde, existe a aresta $(i \xrightarrow{a} c_1)$ em Q_1, $c_1 \in C$ e $a > 0$. Desta forma, para que a mutação em um vértice $k \in Q_0$ mude a cor de i é necessário existir o caminho $c_1 \xrightarrow{b} k \xrightarrow{c} i$, tal que $a \leqslant b \cdot c$ (pela modificação 1 na Definição 4.6). No entanto, a cor de k é verde e não pode existir arestas de c_1 para k. □

Teorema 4.1.2 *Seja $\widetilde{Q} \in Mut(\widehat{Q})$ um quiver congelado e $k \in Q_0$ um vértice não-congelado. Se k é o (um dos) vértice(s) com menor número de arestas incidentes de vértices não-congelados em $\widetilde{Q}$, então a mutação em k direciona o processo para a construção de uma menor sequência maximal.*[3]

Prova. Note que uma sequência maximal tem comprimento mínimo n (mutações sucessivas em n vértices) e que o tamanho de uma sequência maximal é diretamente influenciado pela mudança da cor de vértices não-congelados na sequência. Ou seja, considere a sequência Seq_M. Para que o comprimento c de Seq_M seja maior que n, é necessário que aconteça a mudança de cor (da vermelha para verde) de pelo menos um vértice em Seq_M.

Pela Definição 4.6, Teorema 4.1.1 e Lema 4.15, a única forma de mudança da cor de um vértice $i \in Q_0$ (da vermelha para verde) é através do caminho $c_1 \xrightarrow{a} i \xrightarrow{b} k$ e a

[3] A menor sequência maximal é a sequência com menor número de vértices. Obviamente, podem existir várias sequências maximais com menor número de vértices.

existência da aresta $k \xrightarrow{c} c_1$ (tal que $i,k \in Q_0$, $c_1 \in C$ e $a \leqslant b \cdot c$).[4] Caso contrário, a cor de i não é bem definida. Veja na Figura 4.7, para $a = b = c = 1$.

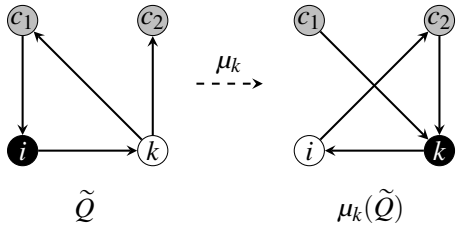


Figura 4.7: *Mudança da cor de um vértice não-congelado (da vermelha para verde).*

Portanto, quanto menor o número de arestas incidentes (de vértices não-congelados) em k, menor será o número de mudanças de cores de vértices não-congelados e a mutação em k direciona o processo para a construção de uma menor Seq_M. □

4.2 A Definitividade Através de Mutações

O objetivo desta seção é fazer o reconhecimento de formas inteiras através de sequências de mutações na matriz de troca relacionada à forma inteira. Para isto, mostramos que as raízes positivas da forma inteira estão na c-matriz da matriz de troca relacionada. Em seguida, com o auxílio da busca em largura (*Breadth First Search - BFS*), apresentamos um algoritmo que faz o reconhecimento de formas inteiras através de mutações.

4.2.1 As Raízes Positivas na c-Matriz

O Teorema 4.2.1 destaca a relação dos c-vetores em uma c-matriz com as raízes positivas da forma inteira e serve de base para a implementação do Algoritmo 4.2.

Teorema 4.2.1 *Seja $\widehat{Q}$ o quiver emoldurado definido a partir de um cluster quiver Q associado a $q_{\mathbb{Z}}$. Se para todo $\widetilde{Q} \in Mut(\widehat{Q})$, todo c-vetor positivo x, temos $q_{\mathbb{Z}}(x) \geqslant 0$ $(resp.\ q_{\mathbb{Z}}(x) > 0)$, então $q_{\mathbb{Z}}$ é fracamente nãonegativa (resp. fracamente positiva).*

Prova. De acordo com Chaves [8], se Q é um *cluster quiver* e todos os c-vetores na c-matriz relacionada possuem a propriedade de coerência de sinais, então os c-vetores são raízes positivas da forma inteira relacionada $q_{\mathbb{Z}}$ (ou o oposto destas raízes, p. ex. [100]

[4] Note que, se $a = b \cdot c$, pelo Teorema 4.1.1 k deve enviar uma ou mais arestas para outros vértices não-congelados.

ou $[-100]$). Agora considere a classe de mutação $Mut(\widehat{Q})$ de um *quiver* emoldurado $\widehat{Q}$ definido a partir de *cluster quiver* Q. Desde que $Mut(\widehat{Q})$ corresponde a todas as mutações possíveis em vértices verdes a partir de $\widehat{Q}$, então $Mut(\widehat{Q})$ contém todas as c-matrizes geradas a partir de $\widehat{Q}$ e todos os c-vetores em c-matrizes na classe $Mut(\widehat{Q})$ possuem a propriedade de coerência de sinais (conforme Teorema 4.1.1). Portanto, pode-se testar todo c-vetor de toda c-matriz em $Mut(\widehat{Q})$ para definir o tipo da forma inteira relacionada $q_{\mathbb{Z}}$. □

4.2.2 Uma Implementação BFS

O Algoritmo 4.2 descreve como podem ser feitas todas as mutações a partir da matriz de troca T de um *quiver* Q. O algoritmo implementa a busca em largura enfileirando as matrizes de troca da classe $Mut(\widehat{Q})$ na lista L_{mat}, da seguinte forma: primeiro adiciona T em L_{mat}; ao retirar T de L_{mat}, inclui em L_{mat} todas as matrizes que são geradas a partir de mutações em T (ou todos os *quivers* gerados a partir de mutações em vértices verdes); a próxima matriz a ser processada[5] será a primeira filha de T adicionada em L_{mat} (a próxima em L_{mat} depois de T); em seguida é processada a segunda filha e assim por diante.

A lista L_{seq} foi utilizada para guardar a sequência verde seq_v que inclui a matriz atual. Assim, sempre que o algoritmo retira uma matriz de L_{mat}, também retira a respectiva sequência verde de L_{seq} (linhas 9 e 10). A função LISTA_VERDES(T_a) (veja Algoritmo D.1) preenche a lista L_v com vértices verdes da matriz T_a sendo processada. Desta forma, sempre que L_v é vazia, a sequência verde seq_v é uma sequência maximal e é colocada na lista de sequências maximais L_{sm} (linha 12).

A lista de raízes positivas L_{rp} é utilizada para guardar os c-vetores positivos de cada nova matriz T_n gerada a partir da mutação[6] em um vértice verde v de L_v. Por fim, a variável $cont_{mut}$ é usada para permitir, através da entrada max_{mut}, o controle da quantidade de mutações realizadas pelo algoritmo.

[5] Retirada de L_{mat} e executadas todas as mutações que são relacionadas a vértices verdes do respectivo *quiver* congelado.

[6] Veja o Algoritmo D.3 no Apêndice D.

Algoritmo 4.2: BFS_MUT(Q, max_{mut})

Entrada: Q = *quiver*; max_{mut} = número máximo de mutações.

Saída: L_{rp} = lista de raízes positivas; $cont_{seq}$ = número de sequências maximais; L_{sm} = lista de sequências maximais.

```
1  T ← [Q, identidade(Q)]                    // Define a matriz de troca T a partir de Q
2  L_mat.inserir(T)
3  seq_v ← ∅
4  L_seq.inserir(seq_v)
5  L_rp ← c-vetores de T                     // inclui os c-vetores positivos de T em L_rp
6  cont_mut ← 0
7  cont_seq ← 0
8  enquanto ((L_mat ≠ ∅) e (cont_mut ⩽ max_mut)) faça
9      T_a ← matriz de troca de L_mat        // T_a é a primeira matriz na lista L_mat
10     seq_v ← sequência verde de T_a em L_seq
11     L_v ← LISTA_VERDES(T_a)               // Vértices verdes de T_a (Algoritmo D.1)
12     se (L_v = ∅) então
13         L_sm.inserir(seq_v)               // A sequência seq_v de T_a é maximal
14         cont_seq ← cont_seq + 1
15     senão
16         enquanto ((L_v ≠ ∅) e (cont_mut ⩽ max_mut)) faça
17             v ← vértice verde de T_a em L_v
18             T_n ← MUTAÇÃO(T_a, v)         // Faz a mutação de T_a em v (Algoritmo D.3)
19             L_rp ← c-vetores de T_n       // Inclui os c-vetores positivos de T_n em L_rp
20             L_mat.inserir(T_n)
21             seq_n ← [seq_v, v]            // Atualiza a sequência verde de T_n
22             L_seq.inserir(seq_n)
23             cont_mut ← cont_mut + 1
24         fim
25     fim
26 fim
27 retorna L_rp, cont_seq, L_sm
```

Um ponto importante na execução do algoritmo é o enfileiramento e a retirada simultâneas da matriz de troca e da respectiva sequência verde nas listas L_{mat} e L_{seq}. Seja T_a uma matriz de L_{mat} e seq_v a respectiva sequência verde. Todas as sequências verdes de matrizes geradas a partir de mutações em vértices verdes de T_a serão compostas por seq_v mais o índice do vértice que direcionou a mutação em T_a (linha 21).

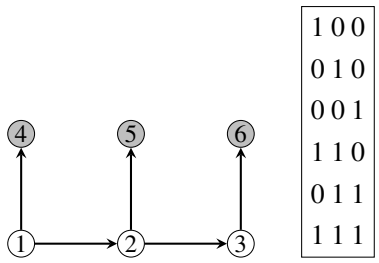
Se considerarmos todas as possíveis mutações a partir de um *quiver* emoldurado $\widehat{Q}$, o Algoritmo 4.2 tem custo fatorial. Isto porque são $n!$ mutações possíveis no pior caso. Ou seja, após cada mutação efetuada, pode-se ter $n-1$ vértices verdes habilitados a receberem mutações. No entanto, este algoritmo se diferencia do apresentado por Brüstle, Dupont e Pérolin (veja em [5]) pelo fato de percorrer a árvore de mutações na largura. Isto permite, por exemplo, encontrar inicialmente as sequências maximais de menor tamanho. Sobretudo, max_{mut} pode ser usada para evitar que o algoritmo entre em um laço infinito (o que ocorre quando a sequência verde é infinita) e não apresente qualquer tipo de resultado.

Teorema 4.2.2 *O Algoritmo 4.2 está correto.*

Prova. O algoritmo coloca todas as matrizes que podem sofrer mutações na lista L_{mat} e, a cada iteração do comando de repetição na linha 8, remove uma matriz desta lista. Se não existem sequências infinitas em $Mut(\widehat{Q})$, então as sequências verdes serão sequências maximais, em algum momento a lista L_{mat} estará vazia e o algoritmo pára. Caso o algoritmo encontre uma sequência infinita em $Mut(\widehat{Q})$, a variável max_{mut} é utilizada para interromper o processo e o algoritmo pára. Desde que todas as matrizes de troca geradas a partir de mutações em vértices verdes serão incluídas em L_{mat}, o Algoritmo 4.2 faz todas as possíveis mutações em vértices verdes a partir de $\widehat{Q}$. □

O Exemplo 4.16 descreve a classe de mutação $Mut(\widehat{Q})$ gerada com a execução do Algoritmo 4.2 sob o *quiver* emoldurado da Figura 4.8(a). Na classe de mutação $Mut(\widehat{Q})$, as raízes positivas (ou os c-vetores relacionados a vértices verdes) estão destacadas com colchetes na c-matriz de cada matriz de troca. O Algoritmo 4.2 faz todas as possíveis mutações em vértices verdes a partir do *quiver* emoldurado $\widehat{Q}$ (31 mutações e 31 novas matrizes de troca). No entanto, como pode ser observado na figura, todas as raízes positivas são encontradas logo no início do processo.

Exemplo 4.16 *A Figura 4.8(a) descreve um quiver emoldurado com 3 vértices e a Figura 4.8(b) as raízes positivas da forma inteira relacionada. A classe de mutação $Mut(\widehat{Q})$ gerada com a execução do Algoritmo 4.2 sob o quiver emoldurado da Figura 4.8(a) é descrita na Figura 4.9.*



(a) $\widehat{Q}$ (b) Σ_q^1

Figura 4.8: *Em (a) quiver emoldurado $\widehat{Q}$ e em (b) as raízes positivas Σ_q^1 da forma inteira relacionada a $\widehat{Q}$.*

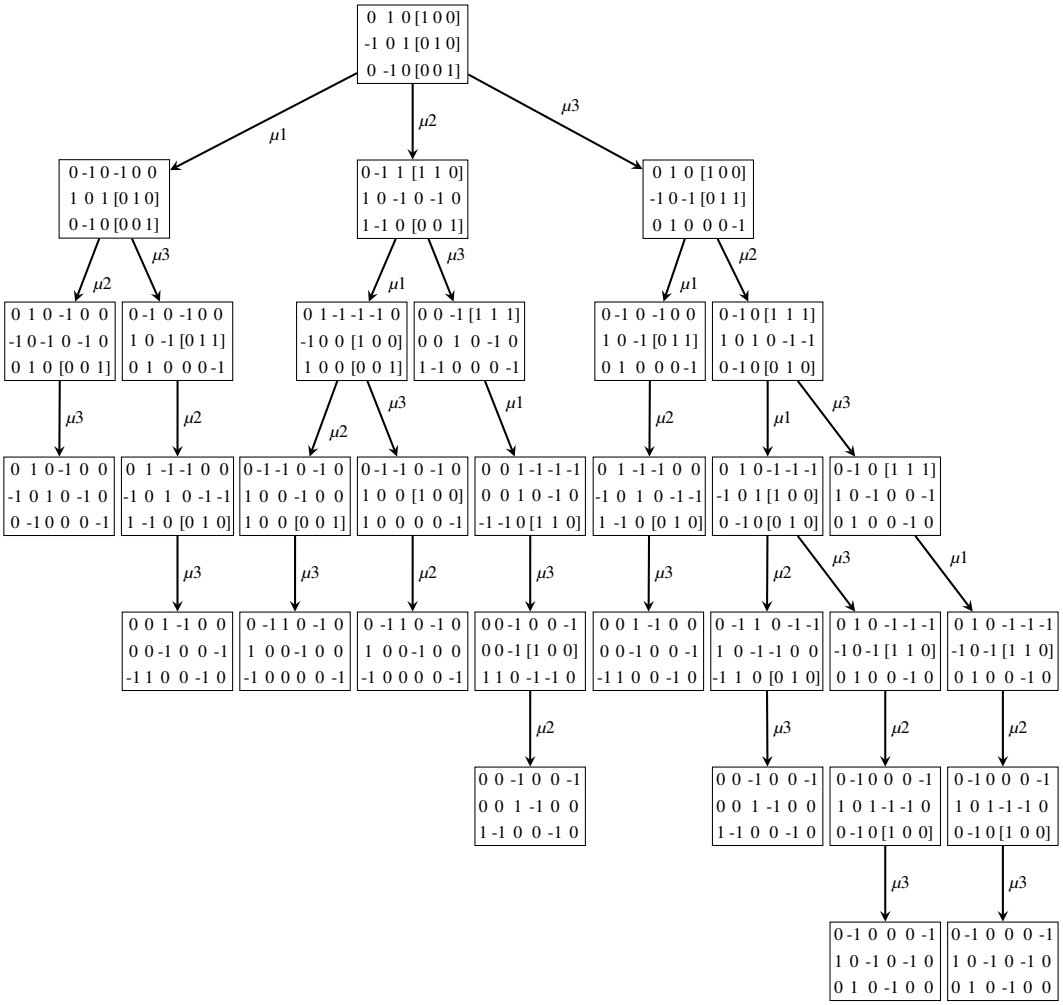


Figura 4.9: *Classe de mutação $Mut(\widehat{Q})$ gerada com a execução do Algoritmo 4.2 sob o quiver da Figura 4.8(a).*

4.3 Simplificando o Processo de Mutação

Esta seção apresenta algumas técnicas que podem ser usadas para simplificar o processo de definitividade de formas inteiras através de mutações. Ainda mais, introduzimos um modelo (*frame*) que permite a identificação de sequências equivalentes em $Mut(\widehat{Q})$ e apresentamos um algoritmo que implementa as técnicas descritas nesta seção.

4.3.1 O $\triangle$infinito

Na classe de mutação $Mut(\widehat{Q})$ de um *quiver* emoldurado $\widehat{Q}$, pode existir um tipo de *quiver* congelado (ou *subquiver* de um *quiver* congelado $\widetilde{Q}$) que leva uma sequência verde em $Mut(\widehat{Q})$ ser infinita. Vamos chamar este *quiver* congelado de $\triangle$*infinito*.

Definição 4.17 ($\triangle$***infinito***) *Seja $\widetilde{Q} \in Mut(\widehat{Q})$ um quiver congelado com $v_1, v_2 \in Q_0$ e $c_1 \in C$. Se $\widetilde{Q}$ possui x arestas de v_1 para v_2, y arestas de v_2 para c_1 e z arestas de c_1 para v_1 (tal que $x \geqslant 2, z > 0$ e $y \geqslant z$) então $\widetilde{Q}$ tem um $\triangle$infinito.*

Teorema 4.3.1 *Seja $\widetilde{Q} \in Mut(\widehat{Q})$ um quiver congelado. Se $\widetilde{Q}$ tem um $\triangle$infinito, então $Mut(\widehat{Q})$ tem uma sequência infinita.*

Prova. Seja $\widetilde{Q} \in Mut(\widehat{Q})$ um *quiver* congelado que tem um $\triangle$*infinito* conforme descrito na Definição 4.17. Vamos mostrar que a mutação no vértice verde v_2 resulta em um outro *quiver* congelado $\mu_{v2}(\widetilde{Q})$ com as mesmas características de $\widetilde{Q}$.

De acordo com a Definição 4.6, o *quiver* resultante $\mu_{v2}(\widetilde{Q})$ (veja Figura 4.10) terá x' arestas de v_2 para v_1 (tal que $x' \geqslant 2$), z' arestas de c_1 para v_2 (tal que $z' > 0$) e $y' = x \cdot y - z$ arestas de v_1 e c_1 . Agora basta verificar que $y' \geqslant z'$.

$$\begin{aligned} y' &= x \cdot y - z \\ &\geqslant 2 \cdot y - z,\ x \geqslant 2 \\ &\geqslant y,\ y \geqslant z \\ &\geqslant z',\ y = z' \end{aligned} \tag{4-1}$$

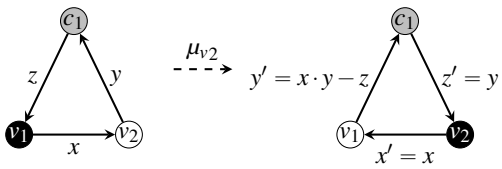


Figura 4.10: *Mutação em um $\triangle$infinito.*

□

Exemplo 4.18 *A Figura 4.11 descreve uma sequência de três mutações em um △infinito, para $x = 2$, $y = 2$ e $z = 1$. Neste caso, o número nas arestas define a quantidade de arestas entre dois vértices. Note-se que, a mutação no vértice verde inverte o sentido das arestas e alterna a coloração dos vértices congelados no quiver congelado.*

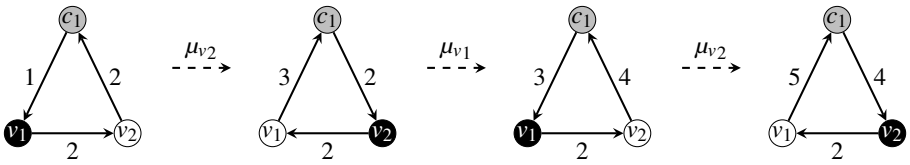


Figura 4.11: *Exemplo de sequência de três mutações em um △infinito.*

Teorema 4.3.2 *Seja $\widetilde{Q} \in Mut(\widehat{Q})$ um quiver congelado. Se existe em $\widetilde{Q}$ o caminho $v_1 \xrightarrow{x} v_2 \xrightarrow{y} c_1$, tal que $v_1, v_2 \in Q_0$ são verdes, $c_1 \in C$, $x \geqslant 2$ e $y > 0$. Então a mutação em v_2 cria um △infinito.*

Prova. Vamos analisar a mutação em v_2. De acordo com a Definição 4.2, a mutação em v_2 inverte as arestas que passam por v_2 (deixando $x' = x \geqslant 2$ arestas de v_2 para v_1 e $z' = y > 0$ arestas de c_1 para v_2) e insere $y' = z + x \cdot y$ arestas de v_1 para c_1 (veja Figura 4.12). Assim, desde que $y' = z + x \cdot y \geqslant 2 \cdot y \geqslant z'$, a mutação em v_2 cria um △*infinito*.

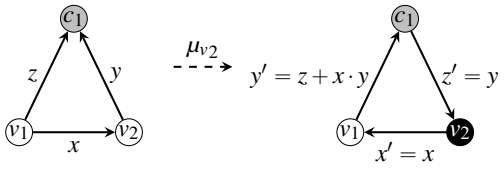


Figura 4.12: *A mutação em v_2 cria um △infinito.*

□

4.3.2 Sequências Equivalentes

Masson e Morissett [33] mostraram que podem existir em $Mut(\widehat{Q})$ sequências de mutações que resultam na mesma matriz de troca e podem ser evitadas durante o processo de mutação.

Teorema 4.3.3 *[33] Seja $\widetilde{Q}$ um quiver congelado. Sejam $i, j \in Q_0$. Se **não existem** arestas entre i e j, então: $\mu i \cdot \mu j = \mu j \cdot \mu i$.*

Teorema 4.3.4 *[33] Seja $\widetilde{Q}$ um quiver congelado. Sejam $i, j \in Q_0$. Se **existem** arestas entre i e j, então: $\mu i \cdot \mu j = \mu j \cdot \mu i \cdot \mu j$.*

O Algoritmo 4.3 utiliza um modelo de implementação (*frame*) que permitirá identificar em $Mut(\widehat{Q})$ o conjunto de matrizes de troca que poderiam ter sequências equivalentes de acordo com os Teoremas 4.3.3 e 4.3.4. De maneira simples, se considerarmos $Mut(\widehat{Q})$ uma árvore (grafo conexo e acíclico), então um *frame* é um subgrafo de $Mut(\widehat{Q})$ composto por 3 níveis:

- o nível (A) contém uma matriz de troca T_{pai};
- o nível (B) contém as matrizes que foram geradas a partir de mutações em T_{pai}; e
- o nível (C) contém as matrizes que foram geradas a partir de mutações em uma matriz do nível (B).

A matriz de troca T_a que está sendo processada será sempre uma matriz no nível (B) (exceto quando esta matriz é raiz[7] em $Mut(\widehat{Q})$). A Figura 4.13, descreve os níveis (A), (B) e (C) para os *frames* das matrizes T_2, T_3 e T_4 da classe $Mut(\widehat{Q})$ do *quiver* da Figura 4.8. Para estas matrizes, o *frame* é composto por T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9 e T_{10}. Um *frame* nem sempre é composto por todas as matrizes no mesmo nível de $Mut(\widehat{Q})$. Por exemplo, o *frame* das matrizes T_{13} e T_{14} é composto por $T_7, T_{13}, T_{14}, T_{20}$ e T_{21}.

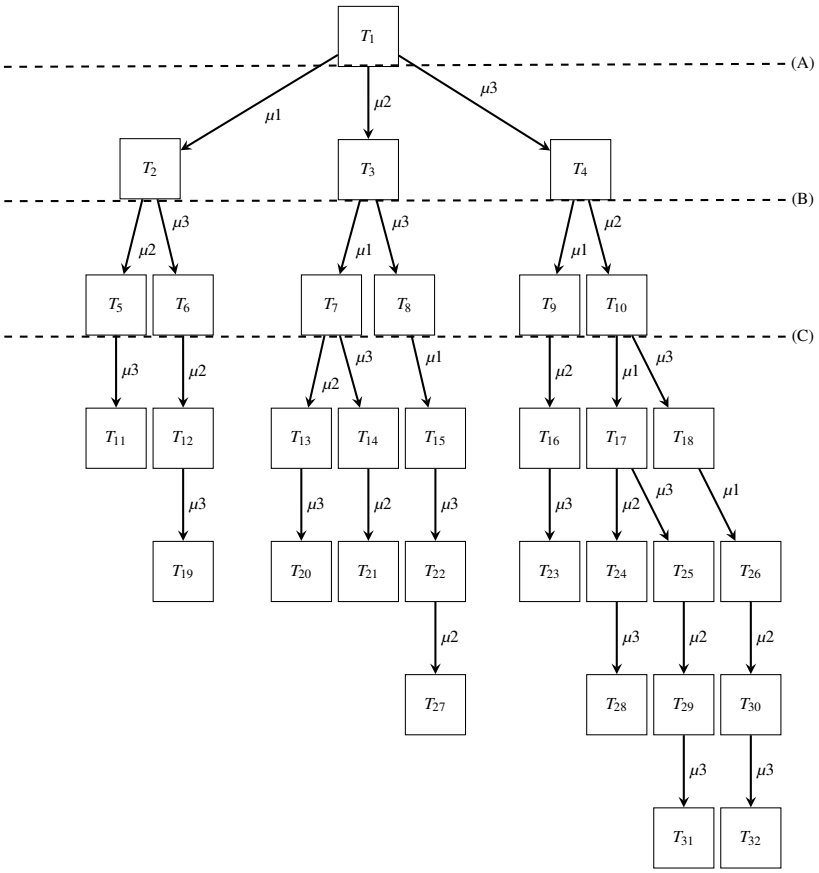


Figura 4.13: *Níveis (A), (B) e (C) para os frames das matrizes T_2, T_3 e T_4 da classe $Mut(\widehat{Q})$.*

[7] Neste caso, o algoritmo somente faz todas as mutações em vértices verdes a partir de T_a.

Esta estrutura foi utilizada no Algoritmo 4.3 para identificar as sequências na forma $\mu_i \cdot \mu_j = \mu_j \cdot \mu_i$, de acordo com o Teorema 4.3.3. Suponha que o algoritmo está trabalhando com uma matriz T_x. Identifica-se a última mutação μ_i em T_x e faz-se uma mutação μ_j a partir de T_x. A sequência $\mu_j \cdot \mu_i$ é então armazenada em uma matriz de sequências do *frame* M_f utilizando os índices j,i. Assim, quando o algoritmo estiver trabalhando com outra matriz T_y no mesmo *frame*, pode-se utilizar M_f para identificar a sequência μ_i, μ_j.

A mesma estrutura foi utilizada para identificar as sequências na forma $\mu_i \cdot \mu_j = \mu_j \cdot \mu_i \cdot \mu_j$. No entanto, vamos ver na subseção 4.3.4 que é feita uma mutação μ_j a mais após μ_i (com relação ao caso anterior) e, para processar a matriz de troca $T(\mu_j(\widetilde{Q}))$ na sequência correta, foi necessário utilizar uma lista auxiliar LA_{mat}.

4.3.3 As Mutações que Geram Raízes Positivas

Teorema 4.3.5 *Seja $\widetilde{Q} \in Mut(\widehat{Q})$, $i,k \in Q_0$ e $c \in C$. Se i é verde e existe o caminho $i \xrightarrow{a} k \xrightarrow{b} c$ em $\widetilde{Q}$, a mutação em k gera uma raiz positiva em $\mu_k(\widetilde{Q})$.*

Prova. As raízes positivas são geradas a partir da inclusão de arestas em $\widetilde{Q}$. De acordo com a Definição 4.6, após a mutação em k, o par de arestas $i \xrightarrow{a} k \xrightarrow{b} c$ corresponde a $i \xrightarrow{ab} c$ em $\mu_k(\widetilde{Q})$. Agora considere a c-matriz de $T(\mu_k(\widetilde{Q}))$. O c-vetor relacionado ao vértice i deve receber na posição c o valor ab. Desde que o valor ab é positivo, a mutação em k gera uma raiz positiva relacionada ao vértice i em $T(\mu_k(\widetilde{Q}))$. □

Note-se que o Teorema 4.3.5 descreve a única forma de gerar raízes positivas (diferentes das raízes simples e que ainda não foram consideradas) a partir de mutações em vértices verdes de $\widetilde{Q}$. Isto porque as demais modificações descritas na Definição 4.6 não geram novas arestas, somente invertem o sentido das arestas existentes. É claro que, se o c-vetor relacionado ao vértice i possui entradas negativas, a mutação em k pode gerar uma raiz positiva. No entanto, esta raiz já foi considerada no processo.

4.3.4 Um Algoritmo Mais Eficiente

O Algoritmo 4.3 implementa os Teoremas 4.3.2, 4.3.3, 4.3.4 e 4.3.5, para simplificar o processo de identificar o tipo de formas inteiras através de mutações. A condição implementada através do Teorema 4.3.2 tem como objetivo evitar a criação de $\triangle$*infinito*. As condições implementadas através dos Teoremas 4.3.3 e 4.3.4 evitam sequências verdes que resultam em *quivers* congelados relacionados à mesma matriz de troca. Já a condição no Teorema 4.3.5 faz as mutações que podem gerar raízes positivas.

De maneira similar ao funcionamento do Algoritmo 4.2, o Algoritmo 4.3 utiliza a lista L_{mat} para enfileirar as matrizes de troca, no entanto, utiliza a lista L_{conf} para guardar

informações importantes sobre cada matriz em L_{mat}. Cada elemento em L_{conf} contém as seguintes informações sobre a matriz T_a sendo processada (atual): identificação da matriz que gerou a matriz atual (chamada de *pai*); identificação da matriz atual (id); última mutação na matriz atual (mut); e cores dos vértices no *quiver* relacionado à matriz atual (cores). As informações em L_{conf} são utilizadas para:

- Mudar o *frame* atual. Se o algoritmo retirar de L_{mat} uma matriz que tem o *pai* diferente do *frame* atual, o algoritmo muda o *frame* (definido através do *pai* da matriz atual), retira a matriz T_{pai} relacionada à matriz atual da lista L_{pai}, e inicializa a matriz M_f que armazena as informações sobre as sequências neste *frame* (veja nas linhas 6 a 10);
- Identificar a cor dos vértices. O vetor cores contém as cores de cada vértice no *quiver* relacionado à matriz atual. Este vetor pode ser usado para verificar se um vértice é verde ou vermelho (quando a cor é igual a 1 ou 0, respectivamente), e também para identificar se um vértice verde tem uma restrição (quando a cor do vértice é igual a 1^*);
- Identificar as matrizes em $Mut(\widehat{Q})$. O índice id é usado para identificar cada matriz em $Mut(\widehat{Q})$.

A implementação dos Teoremas 4.3.3 e 4.3.4 foi feita nas linhas 13 à 26 do algoritmo da seguinte forma: dados um vértice verde μ_j e a última mutação μ_i feita na matriz atual. O algoritmo primeiro verifica, através da matriz de sequências M_f, se a sequência ainda não foi executada no *frame* (linha 14). Caso não, pode guardar a sequência em M_f se estiver de acordo com os Teoremas 4.3.3 (linha 18) e 4.3.4 (linha 22).

A matriz T_{pai}, que gerou a matriz atual T_a, é usada para verificar se existem arestas entre i e j (linhas 18 e 22), de acordo com os Teoremas 4.3.3 e 4.3.4. Note-se que, T_a somente é inserida na lista L_{pai} quando todas as matrizes geradas a partir de T_a forem incluídas em L_{mat} (caso não aconteça mutações em T_a, T_{pai} não é inserida em L_{pai}). Esta é uma questão importante para o funcionamento da troca de *frames*. Por exemplo, seja T_2 a matriz atual na $Mut(\widehat{Q})$ da Figura 4.13. Após incluir T_6 em L_{mat}, T_2 será incluída em L_{pai} e será removida quando T_5 for retirada de L_{mat}.

A função ATUALIZA_LISTAS (veja Algoritmo D.4) faz a mutação da matriz atual em direção a μ_i ou μ_j e atualiza as listas utilizadas no algoritmo. A lista LA_{mat} é uma lista auxiliar para a lista L_{mat}. Esta foi necessária porque quando o algoritmo faz a terceira mutação μ_j na sequência $\mu_j \cdot \mu_i \cdot \mu_j$ (veja Teorema 4.3.4), a matriz $T(\mu_j(\widetilde{Q}))$ não pode ser colocada em L_{mat} após a matriz $T(\mu_i(\widetilde{Q}))$. Se isto acontecer, a identificação do *frame* fica comprometida para as próximas matrizes a serem processadas. Portanto, esta matriz é guardada em LA_{mat} e definida uma restrição para μ_j (veja no Algoritmo D.4, linha

8). Quando esta restrição for identificada, a matriz $T(\mu_j(\widetilde{Q}))$ vai para L_{mat} na sequência correta.

A Figura 4.14 descreve esta situação. Seja T_2 a matriz atual sendo processada pelo algoritmo. O algoritmo vai analisar a matriz T_5 gerada a partir de T_2 (T_5 deve ser incluída em L_{mat}). Se a sequência $\mu_j \cdot \mu_i$ não estiver de acordo com o Teorema 4.3.3 (linha 18), então o algoritmo faz a mutação μ_j gerando T_{11}. Se a sequência $\mu_j \cdot \mu_i \cdot \mu_j$ estiver de acordo com o Teorema 4.3.4 (linha 22), a sequência é guardada em M_f e o processo continua. No entanto, se T_{11} for colocada em L_{mat} logo após T_5, irá acontecer uma mudança de *frame* incorreta[8] quando T_{11} for processada pelo algoritmo. Para corrigir isto, LA_{mat} mantém a matriz T_{11} até a restrição colocada para a cor μ_j ser identificada em T_5.

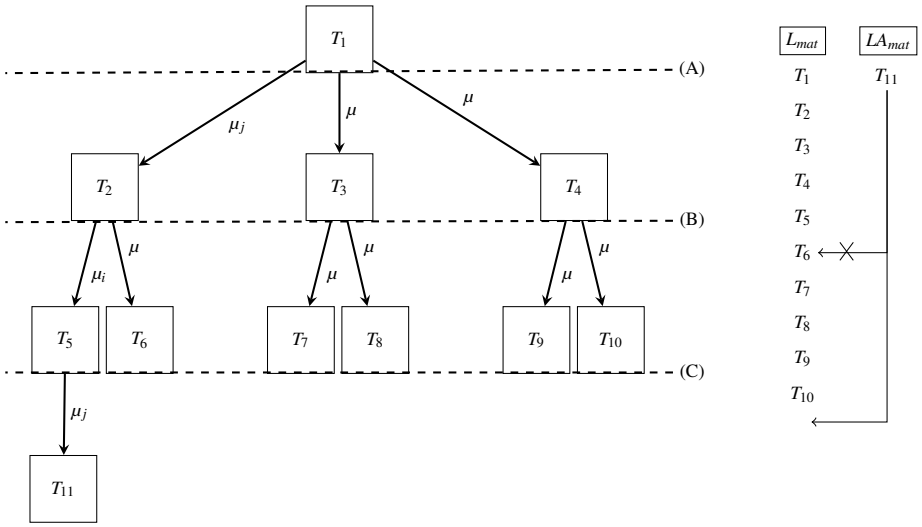


Figura 4.14: *Exemplo de inclusão da matriz T_{11} na sequência correta em L_{mat} com o auxílio da lista LA_{mat}.*

Por fim, na linha 11, a função OBTER_VERTICE (veja Algoritmo D.5) seleciona um vértice verde para mutação. Neste caso, o vértice selecionado não pode gerar um $\triangle$*infinito* (veja Teorema 4.3.2) e deve receber uma ou mais arestas de vértices não-congelados (Teorema 4.3.5). A função retornará o vértice para mutação ou, caso não seja selecionado nenhum vértice, retornará 0 (zero).

[8] Isto porque a matriz T_6 está no mesmo *frame* de T_5.

Algoritmo 4.3: BFS_SMUT(Q)

Entrada: Q = *cluster quiver*.

Saída: L_{rp} = lista de raízes positivas.

1 $T \leftarrow [Q; identidade(Q)], frame \leftarrow 0$

2 $L_{mat}.inserir(T), L_{conf}.inserir(0,1,0,[1,\ldots,1]_n)$
// $L_{conf} \leftarrow$[*pai* de T, id. de T, últ. *mut*. em T, n cores em $T(\widehat{Q})$]

3 **enquanto** $(L_{mat} \neq \varnothing)$ **faça**

4 $T_a \leftarrow$ matriz de troca de L_{mat} // matriz de troca (atual) no início de L_{mat}

5 $C_a \leftarrow$ configuração de L_{conf} // configuração da matriz de troca atual

6 **se** $(C_a.pai \neq frame$ **e** $C_a.pai \neq 0)$ **então**

7 $frame \leftarrow C_a.pai$

8 $T_{pai} \leftarrow$ matriz de troca da lista L_{pai}

9 $M_f \leftarrow zeros(n \times n)$ // Inicializa a matriz de sequências do *frame*

10 **fim**

11 **enquanto** ($\mu_j \leftarrow$OBTER_VERTICE$(T_a, C_a.cores)$ **faça**

/* Algoritmo D.5. Se μ_j possui uma restrição ($C_a.cores(\mu_j) = 1^*$), remover uma matriz de LA_{mat}, incluir em L_{mat} e analisar o próximo μ_j verde */

12 $\mu_i \leftarrow C_a.mut$ // Identifica a última mutação realizada em T_a

13 **se** $(\mu_i \neq 0)$ **então**

14 **se** $(M_f(\mu_j, \mu_i) = 1)$ **então**

15 $M_f(\mu_j, \mu_i) \leftarrow 0$ // sequência μ_i,μ_j ou μ_j,μ_i,μ_j já realizada

16 **senão**

17 $T_n, id_{Tn}, cores_n \leftarrow$ATUALIZA_LISTAS$(T_a, \mu_j, C_a.id, `L_{mat}')$
// Alg. D.4

18 **se** $(cores_n(\mu_i) = 0$ **e** $T_{pai}(\mu_j, \mu_i) = 0)$ **então**

19 $M_f(\mu_i, \mu_j) \leftarrow 1$ // armazena sequência μ_i,μ_j em M_f

20 **senão**

21 $T_{n2}, cores_{n2} \leftarrow$ATUALIZA_LISTAS$(T_n, \mu_i, id_{Tn}, `LA_{mat}')$

22 **se** $(cores_{n2}(\mu_j) = 0$ **e** $T_{pai}(\mu_j, \mu_i) = 1)$ **então**

23 $M_f(\mu_i, \mu_j) \leftarrow 1$ // armazena sequência μ_j,μ_i,μ_j em M_f

24 **fim**

25 **fim**

26 **fim**

27 **senão**

28 ATUALIZA_LISTAS$(T_a, \mu_j, C_a.id, `L_{mat}')$ // T_a é raiz de $Mut(\widehat{Q})$

29 **fim**

30 **fim**

31 $L_{pai}.inserir(T_a)$ // inclui T_a em L_{pai} SE fizer mutações nas linhas 17 ou 21

32 **fim**

33 **retorna** L_{rp}

Teorema 4.3.6 *O Algoritmo 4.3 está correto.*

Prova. O algoritmo coloca todas as matrizes que podem sofrer mutações na lista L_{mat} e, a cada iteração do comando de repetição na linha 3, remove uma matriz desta lista. O mesmo acontece com a lista L_{conf}. Desde que a seleção de cada vértice que irá direcionar a mutação na matriz atual é feita de forma a evitar sequências infinitas (veja Teorema 4.3.2), em algum momento a lista L_{mat} estará vazia e o algoritmo pára. Exceto para matrizes que são equivalentes (de acordo com os Teoremas 4.3.3 e 4.3.4) ou que não geram raízes positivas (de acordo com Teorema 4.3.5), todas as matrizes de troca geradas a partir de mutações em vértices verdes serão incluídas em L_{mat}. Portanto, o Algoritmo 4.3 faz todas as possíveis mutações em vértices verdes a partir de $\widehat{Q}$. □

A Figura 4.15 descreve a classe $Mut(\widehat{Q})$ depois da execução do Algoritmo 4.3 sob o *quiver* emoldurado da Figura 4.8(a). As matrizes na cor cinza não foram geradas devido à aplicação dos Teoremas 4.3.3, 4.3.4 e 4.3.5. Note que, a condição descrita no Teorema 4.3.2 ($\triangle$*infinito*) não foi aplicada porque não existem sequências infinitas neste exemplo.

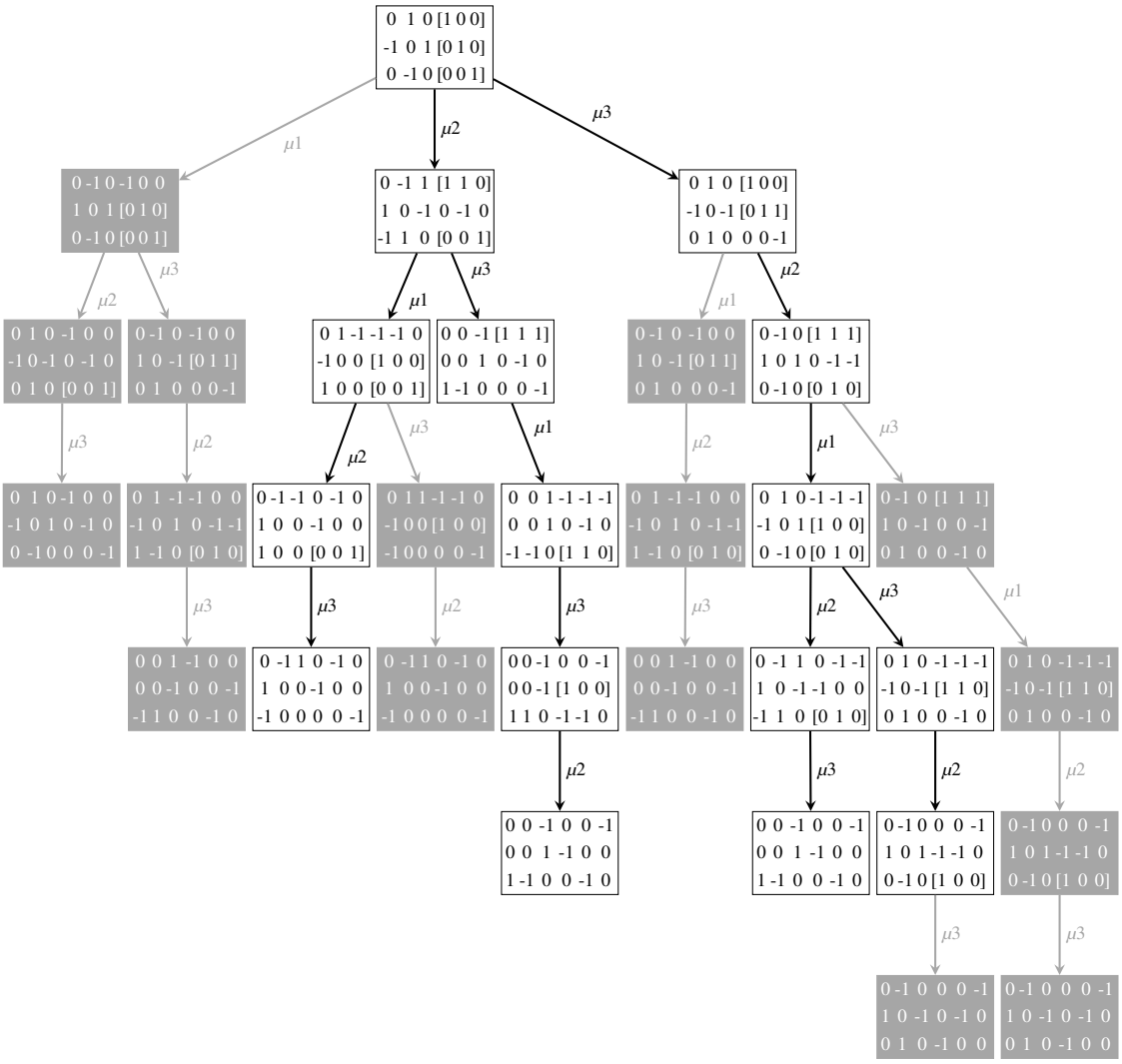


Figura 4.15: *Classe de mutação $Mut(\widehat{Q})$ gerada com a execução do Algoritmo 4.3 sob o quiver emoldurado da Figura 4.8(a). As matrizes na cor cinza não foram geradas devido a aplicação dos Teoremas 4.3.3, 4.3.4 e 4.3.5.*

4.4 Considerações Finais

Este capítulo teve como objetivo mostrar que a definitividade de formas inteiras pode ser feita através de sequências de mutações com características específicas. No início deste estudo, identificamos que a propriedade de coerência de sinais é garantida quando todas as mutações são feitas em vértices verdes a partir de um *quiver* específico, o *quiver* emoldurado. Esta definição inicial foi necessária para garantir que os c-vetores das c-matrizes são equivalentes às raízes positivas da forma inteira relacionada.

Alguns resultados foram importantes para esclarecer propriedades básicas sobre o processo de mutação em *quivers*. Por exemplo, a existência da propriedade de coerência de sinais em *quivers* congelados (Teorema 4.1.1) e a construção da menor sequência maximal (Algoritmo 4.1 e Teorema 4.1.2). O Algoritmo 4.2 permite encontrar e testar todas as raízes positivas da forma inteira relacionada. Apesar de ter custo fatorial, o algoritmo é uma evolução significativa quando comparado ao algoritmo original desenvolvido por Brüstle, Dupont e Pérolin. Isto porque, permite encontrar inicialmente as sequências maximais de menor tamanho e evita que o algoritmo entre em um laço infinito.

A seção mais importante do capítulo certamente é a Seção 4.3. Nesta seção, descrevemos algumas estratégias que podem ser aplicadas na definitividade de formas inteiras através de mutações e, bem como, na solução de qualquer problema que tem como base o processo de mutação. Estas estratégias foram implementadas no Algoritmo 4.3 e resultaram em uma diminuição grande no tempo de processamento quando comparado com o Algoritmo 4.2. No entanto, identificamos que para grandes instâncias (por exemplo, *quivers* com mais de 100 vértices), os algoritmos apresentados nos capítulos anteriores ainda são mais rápidos.

Acreditamos que o Algoritmo 4.3 pode ser ainda mais rápido se for identificada uma forma de interromper o processo quando todas as raízes positivas forem geradas. No entanto, um limite superior para o número de raízes positivas de uma forma inteira ainda é desconhecido. Uma outra forma de deixar o algoritmo mais rápido é fazer somente as mutações que podem gerar novas raízes positivas.

CAPÍTULO 5

Considerações Finais Sobre a Tese

Esta tese apresentou um estudo sobre a definitividade de formas quadráticas racionais e inteiras. Apesar de os primeiros algoritmos para este objetivo terem sido desenvolvidos há mais de 20 anos, muitas questões relacionadas à complexidade computacional não foram bem definidas. Desta forma, o objetivo principal desta tese foi identificar e entender como estava sendo feita a definitividade de formas quadráticas e, com estas informações, desenvolver métodos mais rápidos para o mesmo propósito. Entretanto, era preciso definir uma meta a ser atingida durante o desenvolvimento desta tese.

Uma meta comum quando se trabalha com complexidade computacional é desenvolver algoritmos polinomiais. Existem métodos polinomiais (diretos, expansivos e iterativos) para formas quadráticas racionais, no entanto, estes são de baixa precisão (utilizam limites aproximados ou dependem do número de casas decimais consideradas). Desta forma, no Capítulo 2, utilizamos uma representação de números racionais como fração de inteiros (veja Subseção 2.2.1) e, através da redução de Gauss, desenvolvemos algoritmos polinomiais ($O(1)$ no melhor caso e $O(n^3)$ no pior caso) para fazer a definitividade de formas quadráticas racionais (veja a Subseção 2.2.2 e os Algoritmos 2.6 e 2.7).

Com uma análise inicial do artigo de Dean e De La Peña [12], identificamos que o algoritmo desenvolvido para reconhecer formas inteiras fracamente nãonegativas era eficiente para propósitos matemáticos,[1] no entanto, têm custo exponencial. No Capítulo 3, utilizamos estratégias baseadas em grafos para fazer a definitividade de formas inteiras utilizando algoritmos polinomiais. Uma destas estratégias foi baseada no fato que as formas hipercríticas classificadas por Unger têm 9 ou menos vértices e são conexas através de arestas negativas. Desta forma, desenvolvemos um algoritmo polinomial que identifica se uma forma inteira é fracamente nãonegativa avaliando todos os *subquivers* conexos com 9 vértices (veja Algoritmo 3.2).

No caso das formas inteiras fracamente positivas, identificamos que o número de raízes positivas é polinomial (veja Corolário 3.3.2). No entanto, o grau do polinômio

[1] O algoritmo conseguiu identificar o tipo das formas inteiras consideradas pelos autores.

depende da dimensão da álgebra e pode ser consideravelmente grande. Resolvemos então utilizar a busca em profundidade para desenvolver uma solução mais eficiente.

Com relação ao número de vértices, as restrições críticas e hipercríticas são similares. Ou seja, exceto pelos diagramas $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$, todas as restrições críticas também têm 9 ou menos vértices. Assim, com algoritmos baseados na busca em profundidade, identificamos restrições na forma $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$ (veja os Algoritmos 3.4–3.9) e, se nenhuma restrição for encontrada, testamos todos os *subquivers* com 9 vértices do *quiver* Q de $q_{\mathbb{Z}}$ (veja Algoritmo 3.3). Esta estratégia apresentou resultados muito melhores, mesmo para *quivers* gerados através de uma distribuição aleatória de vértices e arestas (veja Subseção 3.4).

Os resultados conseguidos até este momento atendiam as expectativas que buscávamos durante o doutorado (desenvolver algoritmos polinomiais para formas quadráticas racionais e inteiras). No entanto, durante um estágio feito com o Professor Thomas Brüstle na Universidade de Sherbrooke, surgiu a possibilidade de se identificar o tipo de formas inteiras através de mutações na matriz de troca relacionada. O Capítulo 4 foi desenvolvido com o objetivo de mostrar que a identificação do tipo de formas inteiras pode ser feita através de sequências de mutações com características específicas.

Nesse capítulo, tentamos descrever de forma clara e simples[2] os principais conceitos sobre sequências de mutações no *quiver* relacionado à forma inteira. Destacamos resultados importantes que podem auxiliar o estudo e o desenvolvimento destes conceitos (veja os Teoremas 4.1.1, 4.1.2 e 4.3.2). Também foi descrito como gerar todas as raízes positivas através de sequências de mutações no *quiver* e apresentadas algumas técnicas para simplificar o processo (veja na Seção 4.3).

Ainda sobre os resultados desta tese, pretendemos publicar em revistas e apresentar em conferências internacionais artigos relacionados aos conteúdos dos Capítulos 2, 3 e 4. Um artigo intitulado *Definiteness of Quadratic Forms* foi submetido à revista *Computational & Applied Mathematics* e estamos aguardando o resultado das revisões. Outro artigo intitulado *A Polynomial Recognition of Unit Forms* foi apresentado no *Cologne Twente Workshop 2016*, em Milão, Itália. Uma versão mais completa deste artigo está sendo finalizada e também será submetida para uma revista. Pretendemos ainda apresentar os resultados conseguidos no Capítulo 4 em conferências internacionais e submeter versões mais completas a revistas da área relacionada.

[2] Utilizando conceitos bem conhecidos e evitando notações algébricas que poderiam dificultar a compreensão por parte de leitores da ciência da computação e áreas afins.

Como trabalhos futuros, sugerimos principalmente o desenvolvimento dos conceitos apresentados no Capítulo 4. Acreditamos que novas estratégias podem ser utilizadas para simplificar ainda mais a definitividade de formas inteiras através de mutações, por exemplo: todas as raízes positivas podem ser encontradas a partir de somente uma sequência de mutações em vértices verdes com características específicas; desenvolver um limite superior para o número de raízes positivas da forma inteira; fazer somente as mutações que podem gerar novas raízes positivas.

Uma questão importante[3] que também deve ser tratada como trabalho futuro é a criação do grafo de troca (do inglês *Exchange Graph*, veja mais sobre este assunto em [5, 20]). Este grafo descreve o fluxo de mutações em uma classe $Mut(\widehat{Q})$ e pode ser usado para tirar conclusões sobre a álgebra relacionada. Esta é uma questão importante porque para diversas álgebrass ainda não se conhece o grafo de troca. O Algoritmo 4.3 pode ser usado para gerar o grafo de troca.

[3] Isto não foi considerado nesta tese por se tratar de uma abordagem que não está diretamente relacionada com a definitividade das formas quadráticas.

Referências Bibliográficas

[1] BERGER, M. **Géométrie**. CEDIC/Fernand Nathan, 1990.

[2] BITTINGER, M. **Intermediate algebra**. Pearson, 11th edition, 2010.

[3] BONGARTZ, K. **Treue einfach zusammenhängende algebren i**. *Commentarii mathematici Helvetici*, 57(1):282–330, 1982.

[4] BRÜSTLE, T.; DE LA PEÑA, J.; SKOWROŃSKI, A. **Tame algebras and tits quadratic forms**. *Advances in Mathematics*, 226(1):887–951, 2011.

[5] BRÜSTLE, T.; DUPONT, G.; PÉROTIN, M. **On maximal green sequences**. *International Mathematics Research Notices*, 2014(16):4547–4586, 2014.

[6] BURDEN, R. L.; FAIRES, J. D. **Numerical analysis**. Brooks/Cole, 9 edition, 2011.

[7] BURDEN, R.; FAIRES, J. **Numerical analysis ninth edition**. International Thomson Publishing, ninth edition, 2010.

[8] CHÁVEZ, A. N. **On the c-vectors of an acyclic cluster algebra**. *International Mathematics Research Notices*, p. rnt264, 2013.

[9] COSENTINO, A.; SEVERINI, S. **Weight of quadratic forms and graph states**. *Physical Review A*, 80(5):052309, 2009.

[10] DE LA PEÑA, J. **Quadratic forms and the representation type of an algebra**, volume 343. Sonderforschungsber, Univ. Bielefeld, 1990.

[11] DE LA PEÑA, J.; SKOWROŃSKI, A. **Algebras whose tits form accepts a maximal omnipresent root**. *Transactions of the American Mathematical Society*, 366(1):395–417, 2014.

[12] DEAN, A.; DE LA PEÑA, J. **Algorithms for weakly nonnegative quadratic forms**. *Linear algebra and its applications*, 235:35–46, 1996.

[13] DERKSEN, H.; WEYMAN, J. **Quiver representations**. *Notices of the AMS*, 52(2):200–206, 2005.

[14] DERKSEN, H.; WEYMAN, J.; ZELEVINSKY, A. **Quivers with potentials and their representations ii: applications to cluster algebras**. *Journal of the American Mathematical Society*, 23(3):749–790, 2010.

[15] DIAS, E. S.; CASTONGUAY, D.; LONGO, H.; JRADI, W. A. R. **Efficient enumeration of chordless cycles**. *arXiv preprint arXiv:1309.1051*, 2013.

[16] DRÄXLER, P.; DE LA PENA, J. **Tree algebras with non-negative tits form**. *Communications in Algebra*, 28(8):3993–4012, 2000.

[17] DROZD, Y. A. **Coxeter transformations and representations of partially ordered sets**. *Functional Analysis and its Applications*, 8(3):219–225, 1974.

[18] EVEN, S. **Graph algorithms**. Cambridge University Press, 2011.

[19] FEOFILOFF, P. **Algoritmos de programação linear**. *Editora da Universidade de São Paulo*, 1999.

[20] FOMIN, S.; ZELEVINSKY, A. **Cluster algebras i: foundations**. *Journal of the American Mathematical Society*, 15(2):497–529, 2002.

[21] GABRIEL, P.; ROITER, A. V. **Representations of finite-Dimensional algebras**, volume 73. Springer Science & Business Media, 1997.

[22] GONÇALVES, A. **Introdução à Álgebra**, volume Coleção Projeto Euclides. Associação Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, 5 edition, 2013.

[23] GOWER, J. C.; GROENEN, P. J. **Applications of the modified Leverrier-Faddeev algorithm for the construction of explicit matrix spectral decompositons and inverses**. University of Leiden, 1990.

[24] GRASSL, M.; KLAPPENECKER, A.; RÖTTELER, M. **Graphs, quadratic forms, and quantum codes**. *arXiv preprint quant-ph/0703112*, 2007.

[25] GRIFONE, J. **Algebre linéaire**. Cepadues-éditions, 1990.

[26] HAZEWINKEL, M.; GUBARENI, N.; KIRICHENKO, V. **Algebras, Rings and Modules**, volume 1. Springer, 2004.

[27] HELMBERG, G.; WAGNER, P.; VELTKAMP, G. **On faddeev-leverrier's method for the computation of the characteristic polynomial of a matrix and of eigenvectors**. *Linear algebra and its applications*, 185:219–233, 1993.

[28] HORN, R.; JOHNSON, C. **Matrix analysis**. Cambridge University Press, 2012.

[29] LERARIO, A. **Convex pencils of real quadratic forms**. *Discrete & Computational Geometry*, 48(4):1025–1047, 2012.

[30] LIMA, E. L. **Álgebra Linear**, volume Coleção Matemática Universitária. Associação Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, 6 edition, 2003.

[31] MACLANE, S.; BIRKHOFF, G. **Algebra**. AMS Chelsea Publishing, 2010.

[32] MARCZAK, G.; POLAK, A.; SIMSON, D. **P-critical integral quadratic forms and positive unit forms: an algorithmic approach**. *Linear Algebra and its Applications*, 433(11):1873–1888, 2010.

[33] MASSON, C.; MORISSETTE, J.-P. **Quadrilatères et pentagones dans le graphe d'échange**. 2015.

[34] MAXWELL, S.; CHANCE, M. R.; KOYUTÜRK, M. **Efficiently enumerating all connected induced subgraphs of a large molecular network**. In: *Algorithms for Computational Biology*, p. 171–182. Springer, 2014.

[35] MEYER, C. **Matrix analysis and applied linear algebra**, volume 2. Siam, 2000.

[36] O'LEARY, D. **Scientific computing with case studies**. SIAM, 2009.

[37] ORE, O.; ORE, Y. **Theory of graphs**, volume 38. American Mathematical Society Providence, 1962.

[38] OVSIENKO, S. **Integral weakly positive forms**. *Schur Matrix Problems and Quadratic Forms*, 78:3–17, 1978.

[39] PARLETT, B. **The symmetric eigenvalue problem**, volume 7. SIAM, 1980.

[40] RUTISHAUSER, H. **The jacobi method for real symmetric matrices**. *Numerische Mathematik*, 9(1):1–10, 1966.

[41] SADUN, L. **Applied linear algebra: the decoupling principle**. American Mathematical Society, 2008.

[42] SHIMON, E. **Tame algebras and integral quadratic forms**. *Lectures Notes in Mathematics*, 1984.

[43] SHORES, T. **Applied linear algebra and matrix analysis**. Springer Science & Business Media, 2007.

[44] STOTHERS, A. J. **On the complexity of matrix multiplication**. The University of Edinburgh, 2010.

[45] TARJAN, R. **Depth first search an linear graph algorithms**. *SIAM Journal on Computing*, 1:146–160, 1972.

[46] TONG, L.; YANG, J.; COOPER, R. **Efficient calculation of p-value and power for quadratic form statistics in multilocus association testing**. *Annals of human genetics*, 74(3):275–285, 2010.

[47] TREFETHEN, L.; BAU III, D. **Numerical linear algebra**, volume 50. Siam, 1997.

[48] UNGER, L. **The concealed algebras of the minimal wild, hereditary algebras**. *Bayreuther Mathematische Schriften*, 31:145–154, 1990.

[49] VIGKLAS, P. S. **Upper bounds on the values of the positive roots of polynomials**. PhD thesis, University of Thessaly, School of Engineering, Volos, Greece, 2010.

[50] VON HÖHNE, H.-J. **On weakly positive unit forms**. *Commentarii Mathematici Helvetici*, 63(1):312–336, 1988.

[51] VON HÖHNE, H.-J. **On weakly non-negative unit forms and tame algebras**. *Proc. London Math. Soc*, 73:47–67, 1996.

[52] WILKINSON, J.; WILKINSON, J.; WILKINSON, J. **The algebraic eigenvalue problem**, volume 87. Clarendon Press Oxford, 1965.

APÊNDICE A

Definições Complementares

Este apêndice têm como objetivo disponibilizar alguns informações básicas sobre grafos e os algoritmos DFS e BFS (Seção A.1). Além disto, disponibilizar alguns informações básicas sobre álgebra, *quivers* e representações *quivers* (Seção A.2). Neste apêndice, as noções sobre grafos tiveram como base os trabalhos [37, 15] e as noções sobre álgebra os trabalhos [13, 22, 30].

A.1 Noções sobre grafos

Um **grafo** G consiste de dois conjuntos finitos: um de vértices $V(G)$ e um de arestas $E(G)$. Ao longo do texto, um grafo $G = (V,E)$ é finito e não orientado, sem laços nem arestas múltiplas. Dado um grafo G, consideramos $n = |V(G)|$ e $m = |E(G)|$.

Um **caminho simples** ou **elementar**, denotado por P_k, é uma sequência finita de vértices $\langle v_1, v_2, \ldots, v_k \rangle$ tal que $(v_i, v_{i+1}) \in E(G)$ para todo $i \in \{1, \ldots, K-1\}$ e sem vértice repetido na sequência, isto é, $v_i \neq v_j$ para todo $j \in \{1, \ldots, K\}$ com $j \neq i$. A Figura A.1, exemplifica o conceito.

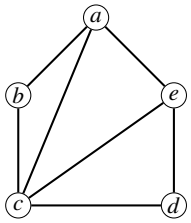
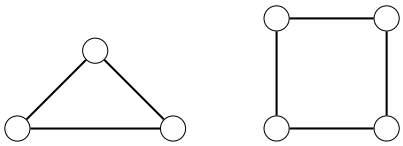


Figura A.1: $\langle a,e,d,c \rangle$ *é um caminho simples, mas* $\langle a,e,c,a,b \rangle$ *não.*

Um **ciclo** é um caminho simples $\langle v_1, v_2, \ldots, v_k \rangle$ tal que $(v_k, v_1) \in E(G)$ e $K \geqslant 3$. Denotamos um grafo composto por um único ciclo de K vértices por C_k. Nesse tipo de grafo, o número de arestas é igual a número de vértices e cada vértice possui grau dois (Figura A.2).

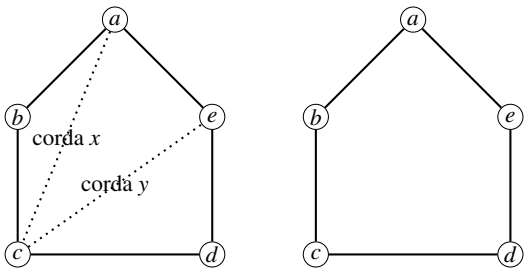


(a) C_3. (b) C_4.

Figura A.2: *Exemplos de grafos C_n.*

Esta definição de ciclo não repete o primeiro vértice no final da sequência, como usualmente é feito. Um ciclo definido desta forma simplifica a representação de uma versão rotacionada dos ciclos. Note que se $\langle v_1, v_2, \ldots, v_k \rangle$ é um ciclo, então ele também pode ser representado por $\langle v_i, v_{i+1} \ldots, v_k, v_1, v_2, \ldots, v_{i-1} \rangle$ e $\langle v_i, v_{i-1}, \ldots, v_2, v_1, v_k, \ldots, v_{i+1} \rangle$, para todo $i = 1, \ldots, K$.

Uma **corda** de um caminho (resp. ciclo) é uma aresta entre dois vértices do caminho (ciclo) que não é parte dele. Podemos ver duas possíveis cordas no ciclo $\langle a, b, c, d, e \rangle$ da Figura A.3(a).



(a) (b)

Figura A.3: *Vértices $\langle a, b, c, d, e \rangle$ formando em (a) um ciclo com corda e em (b) um ciclo sem corda.*

Um caminho (ciclo), de comprimento pelo menos 4, que não possui corda é chamado **caminho sem corda (ciclo sem corda)**. Observe que um caminho (ciclo) sem corda é um subgrafo induzido isomorfo a um P_k (resp. C_k). Para mais informações sobre grafos e ciclos sem corda veja [37] e [15].

A.1.1 Busca em profundidade (DFS)

A busca em profundidade (*Depth-First Search* – DFS) em um grafo tem a estratégia de procurar "mais fundo" no grafo sempre que possível. As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas inexploradas saindo dele. Quando todas as demais arestas de v são exploradas, a busca "regressa" para explorar as arestas que deixam o vértice a partir do qual v foi descoberto.

Esse processo continua até que todos os vértices acessíveis a partir do vértice origem inicial sejam descobertos. Se restarem quaisquer vértices não descobertos, então

um deles será selecionado como uma nova origem e a pesquisa se repetirá a partir daquela origem. Esse processo inteiro será repetido até que todos os vértices sejam descobertos.

Sempre que um vértice v é descoberto durante uma varredura na lista de adjacências de um vértice já descoberto u, a busca em profundidade registra isso definindo o campo **predecessor** de v como $\pi(v) = u$.

O subgrafo predecessor produzido pela pesquisa pode ser composto por várias árvores, porque a pesquisa pode ser repetida a partir de várias origens, formando uma floresta. Arestas de árvore são aquelas descobertas quando os vértices ainda não foram visitados (coloridos). Formalmente, temos que $E_\pi = \{(\pi(v), v) : v \in V(G) \text{ e } \pi(v) \neq NIL\}$.

Os vértices são identificados por marcação de tempo, sendo que cada vértice v possui dois marcadores de tempo: o primeiro, denominado $d(v)$, registra quando v é descoberto pela primeira vez (e acinzentado) e o segundo marcador, denominado $f(v)$, registra quando a busca termina de examinar a lista de adjacência de v, colorindo-o de PRETO. Para todo vértice v, $d(v)$ é menor que $f(v)$. O vértice v é BRANCO antes do tempo $d(v)$, CINZA entre os tempos $d(v)$ e $f(v)$ e depois é marcado com PRETO.

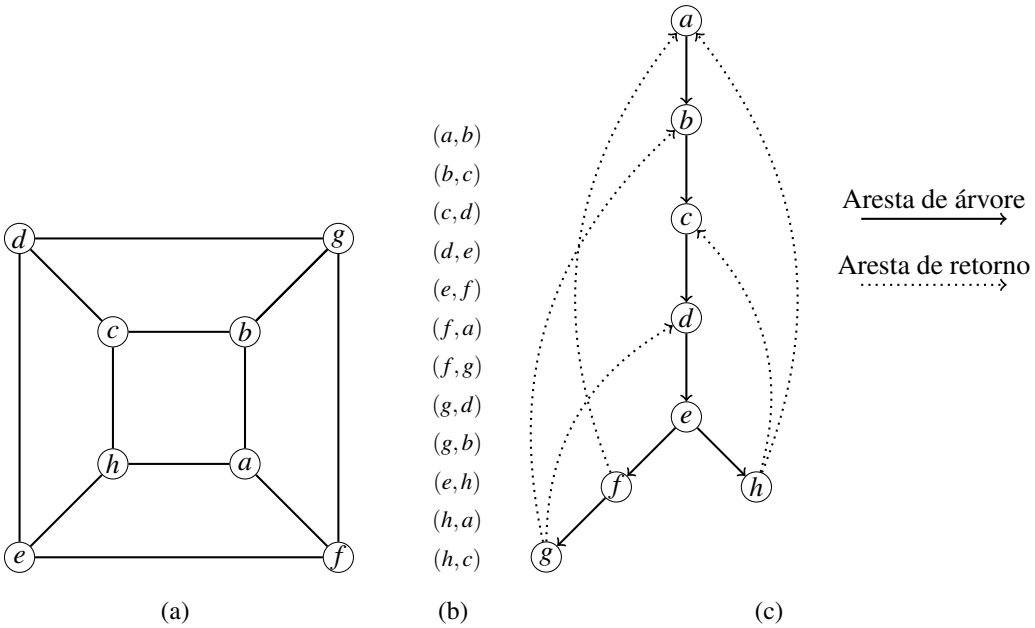


Figura A.4: *A árvore (c) é um exemplo de DFS para o grafo (a), seguindo a ordem de exploração das arestas (b).*

O algoritmo permite que o conjunto de arestas seja classificado em:

1. uma aresta (u, v) é **aresta de árvore** (A) se v foi descoberto durante a sua exploração;
2. as **arestas de retorno** (R) conectam um vértice u com um antecessor v na árvore de busca;

3. **arestas diretas** (D) são aquelas que não são de árvore e que conectam um vértice v a um descendente u na árvore de busca; e
4. **arestas cruzadas** (C) são todas as outras.

Observe que em um grafo não orientado os conceitos de arestas de retorno e de arestas diretas são os mesmos. Em um grafo orientado, as arestas de retorno formam um ciclo orientado, enquanto que as diretas formam um ciclo não orientado.

Algoritmo A.1: DFS(G)

Entrada: Grafo G.

Saída: Uma árvore de busca primeiro em profundidade.

```
1  para cada u ∈ V(G) faça
2      cor(u) ← BRANCA
3      π(u) ← NIL
4  tempo ← 0
5  para cada u ∈ V(G) faça
6      se (cor(u) = BRANCA) então
7          DFS-VISIT (u)
```

Algoritmo A.2: DFS-VISIT(u)

Entrada: Um vértice $u \in V(G)$.

```
1  cor(u) ← CINZA    /* Branco, o vértice u acabou de ser descoberto */
2  tempo ← tempo + 1
3  d(u) ← tempo
4  para cada v ∈ Adj(u) faça                    /* Explora a aresta (u,v) */
5      se (cor(v) = BRANCA) então
6          π(v) ← u
7          DFS-VISIT (v)
8  cor(u) ← PRETA          /* Colore u de preto; ele é finalizado */
9  f(u) ← tempo
10 tempo ← tempo + 1
```

Para mais informações sobre DFS veja [45] e para mais informações sobre algoritmos para grafos veja [18].

A.1.2 Busca em largura (BFS)

O algoritmo de busca em largura (*Breadth-First Search* – BFS) é um algoritmo que encontra os caminhos mais curtos a partir de um vértice inicial. Essa técnica é

utilizada em vários algoritmos mais complexos, como por exemplo o algoritmo de *Dijkstra* (para determinar o menor caminho entre vértices para grafos ponderados) e o de árvore geradora mínima.

Dado um grafo G e um vértice de origem s, a busca em largura explora sistematicamente as arestas de G até "descobrir" cada vértice acessível a partir de s. O algoritmo calcula a distância (menor número de arestas) desde s até todos os vértices acessíveis. Ele encontra o caminho mais curto de s a qualquer outro vértice.

Para controlar o andamento durante a busca em largura, cada vértice é colorido de BRANCO, CINZA ou PRETO. No início, todos os vértices são brancos e mais tarde eles podem se tornar acinzentados e depois pretos.

Algoritmo A.3: BFS(G,s)

Entrada: Um grafo G e um vértice $s \in V(G)$.

Saída: Uma árvore de busca primeiro na largura.

```
1  para cada v ∈ V(G) − {s} faça
2      cor(u) ← BRANCA
3      d(u) ← ∞
4      π(u) ← NIL
5  cor(s) ← CINZA
6  d(s) ← 0
7  π(s) ← NIL
8  Q ← 0
9  ENQUEUE(Q,s)
10 enquanto (Q ≠ ∅) faça
11     u ← DEQUEUE(Q)
12     para cada v ∈ Adj(u) faça
13         se (cor(v) = BRANCa) então
14             cor(v) ← CINZA
15             d(v) ← d(v) + 1
16             π(u) ← u
17             ENQUEUE(Q,v)
18     cor(u) ← PRETA
```

A busca em largura constrói uma árvore primeiro na extensão, contendo inicialmente apenas sua raiz s. Sempre que um vértice branco v é descoberto na verificação da lista de adjacências de um vértice u já descoberto, o vértice v e a aresta (u,v) são adicionados à árvore. Dizemos que u é predecessor ou pai de v na árvore. Se u está em um caminho na árvore a partir da raiz s até o vértice v, então u é um ancestral de v e v é um descendente de u.

As funções $cor(v)$, $\pi(v)$ e $d(u)$ têm a mesma conotação do algoritmo DFS explicado anteriormente. Para mais informações sobre BFS veja [18].

A.2 Noções sobre Álgebra

A **álgebra** é a arte de manipular somas, produtos e o poder dos números [31]. O termo álgebra tem suas origens em meados do século IX, no livro *Hisab al-jabr wál-muqabala* do matemático Persa *Abu Ja'far Muhammad ibn Musa Al-khwarizmi*, considerado o fundador da álgebra. Nesse trabalho, *al-jabr* foi utilizado para designar procedimentos para resolução de equações, tais como: a soma da mesma quantidade positiva a ambos os membros de uma equação, para eliminar quantidades negativas; e o produto da mesma quantidade positiva por ambos os membros de uma equação, para eliminar frações. Com o tempo, *al-jabr* passou a ser usado para representar conhecimentos gerais sobre operações e equações numéricas.

No início, a álgebra foi basicamente aplicada a generalizações do conceito de números naturais, racionais positivos e negativos, completos e irracionais. Em seguida, reconheceu-se que muitas das idéias ditas "algébricas" se aplicavam igualmente a objetos que não são números, como por exemplo vetores, matrizes e transformações.

Uma importante evolução da álgebra está relacionada ao estudo de propriedades de operações algébricas sobre objetos gerais, sem a especificação da natureza da atuação destas operações, nem mesmo a descrição de como os resultados das operações devem ser calculados. Este estudo faz-se com a identificação de estruturas algébricas abstratas, tomando como hipótese um determinado conjunto de regras básicas que a operação é suposta a verificar, como por exemplo a comutatividade, associatividade e distributividade.

A.2.1 A Álgebra Abstrata

A **álgebra abstrata** é uma parte da matemática que estuda as estruturas algébricas. Vamos descrever as principais estruturas algébricas, bem como suas características e composições, com o objetivo de identificar estruturas básicas para a contextualização da teoria da representação através de *quivers*.

Uma **estrutura algébrica** (ou estrutura algébrica abstrata) é formada por um conjunto não vazio C, considerado o suporte da estrutura, e uma operação binária sobre C, que é uma função $f : C \times C \to C$ sobre o conjunto, que satisfazem certos axiomas.

Neste caso, algumas convenções são comumente utilizadas. Se $f : C \times C \to C$ for uma operação binária em C, é comum escolher um símbolo, geralmente "$+$" ou "$*$", para representar a aplicação da operação sobre dois elementos $a, b \in C$. Por exemplo, escrevendo "$a + b$" ou "$a * b$" em vez de $f(a, b)$.

Um **elemento neutro** n (também chamado de **identidade**) para uma operação binária "$*$" sobre um conjunto C é um elemento tal que, $e \in C$ e $e * a = a * e = a$ para todo $a \in C$ e se $a * b = b * a = e$, para $a, b \in C$, a é considerado **invertível** e b é o **inverso** de a.

Existem conjuntos na qual os elementos podem ser trabalhados algebricamente, isto é, a combinação de dois elementos de um conjunto, possivelmente de diferentes maneiras, para obter um terceiro elemento deste conjunto. Neste caso, também existe a propriedade de **fechamento** no conjunto. Sobretudo, tais operações algébricas podem estar sujeitas a certas regras, chamadas axiomas.

A estrutura algébrica mais simples é o **monóide**. Esta estrutura é composta pelo par $(C, *)$ e deve satisfazer as seguintes propriedades:

1. a operação "$*$" tem identidade e em C;
2. a operação "$*$" é Associativa, ou seja, $(a * b) * c = a * (b * c)$ para quaisquer $a, b, c \in C$.

Vários conjuntos podem ser caracterizados como monóides, tais como: o conjunto dos números inteiros $\mathbb{Z}$ com a operação binária multiplicativa "*" e identidade $\{1\}$; ou o conjunto $\mathbb{Z}$ com a operação binária aditiva "+" e identidade $\{0\}$.

Uma característica importante da álgebra abstrata é a que define os **grupos**. Estes são simplesmente monóides em que todos seus elementos são invertíveis. O conjunto $\mathbb{Z}$ com a operação binária aditiva "+", identidade $\{0\}$ e a operação subtração "-" como inverso é um grupo. Um grupo/monóide $(C, *)$ é comutativo, também chamado de **abeliano**, caso a operação binária seja comutativa. Ou seja, se $a * b = b * a$, para quaisquer $a, b \in C$.

Algumas aplicações práticas na matemática, e até mesmo em outras ciências como física e geometria, precisam de estruturas mais complexas, por exemplo a soma e multiplicação de matrizes quadradas de mesma dimensão. Tal complexidade está basicamente relacionada à quantidade ou a forma de aplicação destas operações, bem como aos axiomas que as acompanham. Nesta seção, vamos tratar ainda de anéis, corpos e espaços vetoriais, módulos, álgebras e módulos sobre uma álgebra.

Um **anel** é composto por um conjunto A e duas operações binárias, chamadas adição "$+$" e produto "$*$", sobre este conjunto. O terno ordenado $(A, +, *)$ ainda deve possuir as seguintes propriedades:

- Propriedades para adição
  1. $\forall a, b, c \in A, (a + b) + c = a + (b + c)$ (associatividade);
  2. $\forall a, b \in A, a + b = b + a$ (comutatividade);
  3. $\forall a \in A$, $\exists e \in A \mid a + e = a$ (identidade);
  4. $\forall a \in A$, $\exists b \in A \mid a + b = e$ (simetria).
- Propriedades para o produto

1. $\forall a,b,c \in A$, $(a*b)*c = a*(b*c)$ (associatividade);
2. $\forall a,b,c \in A$, $a*(b+c) = a*b+a*c$, e $(b+c)*a = b*a+c*a$ (distributividade).

De forma geral, um anel pode ser pensado como uma generalização do conjunto $\mathbb{Z}$. Isto porque pode-se somar e multiplicar elementos de um anel mantendo todas as propriedades para ambas as operações. Este anel pode ser um **anel de divisão**, caso exista uma forma de dividir todo elemento do anel por um valor não nulo (ou seja, $A = A\backslash\{0\}$), ou um **corpo**, caso seja um anel de divisão comutativo. O conjunto $\mathbb{Q}$ dos números racionais e $\mathbb{C}$ dos números complexos são corpos. Note-se que o conjunto dos números naturais $\mathbb{N}$ não é um anel, desde que não atende a lei de simetria na adição.

Uma das estruturas mais interessantes da álgebra é o espaço vetorial. Este tem suas origens em tópicos da geometria e física, para descrever força e velocidade, e se diferem de anéis e corpos porque uma de suas operações é realizada com elementos que estão fora do conjunto. Mais formalmente, um **espaço vetorial** (ou **K-espaço vetorial**) é composto por um conjunto não vazio X, um corpo K e duas operações: adição entre os elementos de X e uma multiplicação entre um elemento de X e um elemento do corpo K. Quando tais operações são feitas sobre um anel, ao invés de sobre um corpo, temos a caracterização de um **módulo**. A Tabela A.1 descreve todos os axiomas que um espaço vetorial deve satisfazer com relação a suas operações.

Axiomas sobre a operação em X	**Axiomas sobre a operação entre X e K**
$\forall A, B \in X; A+B \in X$	$\forall A \in X; c \in K; c*A \in X$
$\forall A, B \in X; A+B = B+A$	$\forall A \in X; c,d \in K; c*(d*A) = (c*d)*A$
$\forall A, B, C \in X; (A+B)+C = A+(B+C)$	$\exists e \in K \mid A*e = A \forall A \in X$
$\forall A \in X \exists e \in X \mid A+e = A$	$\forall A, B \in X; c \in K; c*(A+B) = c*A+c*B$
$\forall A \in X, \exists B \in X \mid A+B = B+A = e$	$\forall A \in X; c,d \in K; (c+d)*A = c*A+d*A$

Tabela A.1: *Axiomas para operações em um espaço vetorial.*

Dado um corpo K, o conjunto das matrizes $M_{n\times m}(K)$ com as operações usuais de adição de matrizes e multiplicação de um elemento de K por uma matriz é um espaço vetorial sobre K. Um outro exemplo é o conjunto $\mathbb{R}^2$, para (x_1, x_2),$(y_1, y_2) \in \mathbb{R}^2$, $\alpha \in \mathbb{R}$ com as seguintes operações:

- $(x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$;
- $\alpha\,(x_1, x_2) = (\alpha x_1, \alpha x_2)$.

Em se tratando de espaços vetoriais, um conceito importante está relacionado à **base** e a **dimensão** de uma espaço vetorial. Seja o conjunto $B = \{\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n}\}$ de vetores de um espaço vetorial E, B é base de E se B gera E (todo elemento de E pode ser escrito através de combinações lineares de elementos em B) e se B é linearmente independente. Um conjunto é **linearmente independente** (L.I.) se nenhum de seus elementos pode ser

formado através de combinações lineares de outros elementos. A **dimensão** é o número de vetores na base do espaço vetorial.

O conjunto {(1,0), (0,1)} é uma base de $\mathbb{R}^2$ desde que todo $(x, y) \in \mathbb{R}^2$ pode ser gerado através da equação $(x, y) = x(1,0) + y(0,1)$ e é L.I. A independência linear é verificada se a combinação linear do conjunto com o vetor nulo do espaço vetorial em questão for igual a zero (por exemplo: $x(1,0) + y(0,1) = (0,0) \Rightarrow x = 0$ e $y = 0$).

Uma **álgebra sobre um corpo K** (ou álgebra sobre K, ou simplesmente álgebra) é usualmente representada como K**-álgebra**. Essa estrutura é composta por um **Anel** $(A, \blacktriangleleft, \blacktriangleright)$, um corpo $(K, +, *)$ e uma operação externa "$\boxdot$" entre elementos de A e K. Ainda mais, desde que K é um corpo, toda K-álgebra é uma caso particular de K-espaço vetorial. A seguir é descrito alguns exemplos de álgebras sobre um corpo:

1. $\mathrm{A} = \left\{ \begin{bmatrix} a & b \\ 0 & c \end{bmatrix} \mid a, c \in \mathbb{Z}; b \in \mathbb{Z}_2; \right\}$, A é uma $\mathbb{Z}$**-álgebra**;
2. $\mathrm{A} = \left\{ \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \mid a, c \in \mathbb{Z}; b \in \mathbb{Q}; \right\}$, A é uma $\mathbb{Z}$**-álgebra**;
3. $\mathrm{A} = \left\{ \begin{bmatrix} a & b \\ 0 & c \end{bmatrix} \mid a \in \mathbb{Q}; b \in \mathbb{R}[x]; c \in \mathbb{Q}[x]; \right\}$, A é uma $\mathbb{Q}$**-álgebra**.

Uma K**-álgebra** Λ ainda pode fazer parte de uma estrutura mais complexa, por exemplo um módulo M sobre a álgebra Λ. Esta estrutura é formada com um grupo abeliano $(M, \text{"}\square\text{"})$ e uma operação externa "$\boxdot$" entre elementos de M e Λ, tal que a operação externa é multiplicativa e associativa. Assim, temos que $M \times \Lambda \to M$ e $(m, a) \mapsto ma$ forma uma Λ**-módulo à direita** e, $\Lambda \times M \to M$ e $(a, m) \mapsto am$ forma um Λ**-módulo à esquerda**. Seja um corpo K, um espaço vetorial V e uma transformação linear $T : V \to V$. Pode-se definir uma estrutura de **K[t] módulo à esquerda** com um elemento de K[t] ($p(t) = a_0 \square a_1 t \square \cdots \square a_d t^d$), um $v \in V$ e a multiplicação externa da seguinte forma: $p \boxdot v = (a_0 I_V \square a_1 T \square \cdots \square a_d T^d) \boxdot (v)$.

A.2.2 Morfismo e Isomorfismo

Um **morfismo** é uma função que têm como domínio e contradomínio conjuntos que dão suporte a estruturas algébricas e que preservam as operações algébricas das estruturas envolvidas [26]. Por exemplo: se $(A, +, *)$ e $(B, \diamond, \bullet)$ são anéis e $\delta : A \to B$ uma função, δ é um morfismo sobre os anéis $(A, +, *)$ e $(B, \diamond, \bullet)$ se:

1. $\delta(a_1 + a_2) = \delta(a_1) \diamond \delta(a_2), \forall a_1, a_2 \in A$;
2. $\delta(a_1 * a_2) = \delta(a_1) \bullet \delta(a_2), \forall a_1, a_2 \in A$.

O morfismo entre os anéis $(A, +, *)$ e $(B, \diamond, \bullet)$ pode ser representado graficamente conforme Figura A.5.

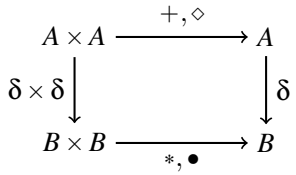


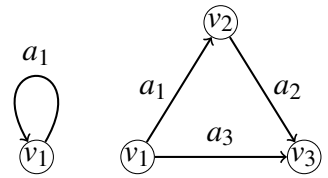
Figura A.5: *Morfismo em Anéis.*

Quando dois conjuntos finitos têm o mesmo número de elementos, então existe uma bijeção entre esses conjuntos. Se um morfismo é bijetivo (ou seja, a função correspondente é uma função injectiva e sobrejetiva e preserva as operações binárias nas estruturas algébricas) tem-se um **isomorfismo** $\delta : A \to B$ (com relação ao exemplo da Figura A.5).

A.2.3 *Quivers*

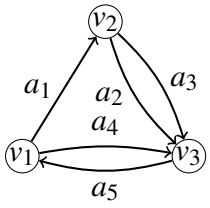
Um **quiver** Q é um grafo direcionado $Q = (V, E, o, d)$ onde V é um conjunto finito de vértices, nomeados aqui através da letra V e E é um conjunto finito de arestas (no caso setas), nomeados através da letra E. Em um *quiver*, as funções $o : E \to V$ e $d : E \to V$ determinam a origem e o destino das arestas em E. Podem-se ter grafos múltiplos e mais de uma aresta entre dois vértices.

Em um *quiver* são permitidos laços, como no exemplo da Figura A.6(a), múltiplas arestas na mesma direção ($o(a_2) = o(a_3)$ e $d(a_2) = d(a_3)$) ou em direções opostas ($o(a_4) = d(a_5)$ e $o(a_5) = d(a_4)$), como no *quiver* da Figura A.6(c).



(a) *Quiver com laços.*

(b) *Quiver com múltiplas arestas na mesma direção.*



(c) *Quiver com arestas em direções opostas.*

Figura A.6: *Exemplos de quivers.*

Os *quivers* são usados para representar estruturas algébricas e suas relações. Este tipo de representação pode ser feita, por exemplo, associando a cada vértice de um *quiver* um espaço vetorial e a cada aresta um mapeamento linear. Formalmente, uma **representação** R de um *quiver* Q sobre um corpo K é um conjunto de espaços vetoriais $\{R_v \mid v \in V\}$ junto com um conjunto de mapeamentos lineares $\{R_a : R_{o_a} \to R_{d_a} \mid a \in E\}$, onde R_{o_a} representa o vértice de origem e R_{d_a} o vértice destino da aresta a no conjunto E. A Figura A.6(a) é uma representação *quiver* com um espaço vetorial V_1 e um endomorfismo $a_1 : V_1 \to V_1$, ou seja, um morfismo de v_1 em si mesmo.

A **álgebra caminho** é a álgebra construída através de caminhos no *quiver*. Ou seja, uma álgebra KQ é o espaço vetorial gerado por todos os caminhos no *quiver* Q. Este tipo de construção gera a base de um espaço vetorial. Para gerar a álgebra caminho é necessário definir uma regra de multiplicação entre os caminhos no *quiver*. Esta regra determina quando concatenar caminhos no *quiver* e em geral é definida da seguinte forma para dois caminhos p e q:

$$p * q = \begin{cases} \text{pq} & \text{se } d(p) = o(q), \\ 0 & \text{caso contrário.} \end{cases}$$

Por exemplo, o *quiver* da Figura A.6(b) tem álgebra caminho de dimensão finita, com base igual a $\{e_{v_1}, e_{v_2}, e_{v_3}, e_{a_1}, e_{a_2}, e_{a_3}, e_{a_1a_2}\}$. Neste caso, quando se multiplica e_{a_1} com e_{a_2} tem-se $e_{a_1a_2}$ e quando se multiplica e_{a_1} com e_{a_3} tem-se 0.

Uma Álgebra Λ é **conexa** (ou **indecomponível**) se não pode ser decomposta como soma direta de duas álgebras. Seja Λ uma k-álgebra de dimensão finita. Denotamos por *mod* Λ a categoria de Λ-módulos a esquerda. Dizemos que Λ é de representação finita se e somente se existe um número finito de Λ-módulos a esquerda indecomponíveis (salvo isomorfismo). A álgebra Λ é **Tame** se para todo número n, todos Λ-módulos de dimensão n é isomórfico a um módulo pertencente a um número finito de famílias 1-parâmetro (veja mais detalhes em [13, 22]).

Sob o ponto de vista de *quivers*, a álgebra KQ é de representação finita se e somente se existe um número finito de caminhos em Q, ou seja, se e somente se Q é um *quiver* finito sem ciclos orientados.

A.2.4 Exemplos de representações de *quivers*

O *quiver* da Figura A.6(a), consistindo de um único vértice e um laço, tem como base da álgebra caminho KQ o conjunto $\{e_1, a, a^2, \ldots, a^K, \ldots\}$ e multiplicação do vetor base dada por:

$$\begin{array}{cc} e_1 a^K = a^k e_1 = a^K & \text{para todo } K \geqslant 0, \text{ e} \\ a^k a^l = a^{K+l} & \text{para todo } K, l \geqslant 0 \text{ e } a^0 = e_1. \end{array}$$

Assim, KQ é isomorfo a álgebra polinomial $K[t]$ na variável t com isomorfismo induzido por K mapeamentos lineares tais como $e_1 \mapsto 1$ e $a \mapsto t$. A Tabela A.2 mostra dois exemplos clássicos da literatura que descrevem como álgebras podem ser representadas graficamente através de *quivers*.

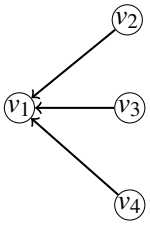
KQ	**K-álgebra**
	$\begin{bmatrix} K & 0 \\ K^2 & 0 \end{bmatrix}$
	$\begin{bmatrix} K & 0 & 0 & 0 \\ K & K & 0 & 0 \\ K & 0 & K & 0 \\ K & 0 & 0 & K \end{bmatrix}$

Tabela A.2: *Exemplos de $KQ \cong K$-álgebra.*

Para maiores informações sobre álgebra veja [22] e [30]. E para mais informações sobre representações *quivers* veja [13].

APÊNDICE B

Apêndice do Capítulo 2

Este apêndice tem como objetivo apresentar os métodos e algoritmos mais utilizados atualmente para se fazer a definitividade de formas quadráticas racionais. Além disto, fornecer uma indicação da análise de complexidade de cada um deles.

B.1 A Definitividade Através dos Menores Principais

Uma das formas de verificar se uma forma quadrática racional é positiva definida é através dos seus menores principais naturalmente ordenados.

Definição B.1 *Seja A uma matriz simétrica de ordem n. Os* **menores principais naturalmente ordenados (MPNOs)** *de A são definidos como determinantes das submatrizes principais líderes (do inglês leading principal submatrix):*

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kk} \end{bmatrix}, \text{ para } k = 1, 2, \dots, n.$$

Assim, A é positiva definida se e somente se seus menores principais naturalmente ordenados são todos estritamente positivos. Este método é implementado no Algoritmo B.1. A complexidade de tempo para se encontrar os menores principais naturalmente ordenados é $O(n^4)$, sendo que o laço na linha 2 tem custo $O(n)$ e calcular o determinante da matriz (linha 3) tem complexidade[1] $O(n^3)$.

[1] Este custo pode variar dependendo do método utilizado, veja mais detalhes em [44].

Algoritmo B.1: DEF_MENORESPRINCIPAIS(A)

Entrada: $A_{n\times n}$ = matriz quadrada de ordem n.

Saída: $res = Verdadeiro$ se A é positiva definida; $res = Falso$ caso contrário.

```
1 res ← Verdadeiro
2 para cada (k ← 1...n) faça
      // Verifica se os MPNOs de A são maiores que zero.
3     se (det(A.matriz_LC(1,k)) ⩽ 0) então
          // A.matriz_LC(1,k) retorna a submatriz com k linhas e k colunas de A.
4         res ← Falso
5     fim
6 fim
7 retorna res
```

Uma solução mais completa, que permite identificar os demais tipos de formas quadráticas racionais, é conhecida como expansão direta. A expansão direta utiliza o determinante de todas as submatrizes principais de ordem k de uma matriz A.

Definição B.2 *Seja A uma matriz simétrica de ordem n. Uma* **submatriz principal** *(do inglês leading submatrix) de ordem k de A é obtida eliminando-se $n-k$ colunas e as correspondentes $n-k$ linhas de A. O determinante de uma submatriz principal de ordem k é chamado de* **menor principal** *de ordem k de A.*

Exemplo B.3 *(Menores Principais) Seja A uma matriz simétrica de ordem 3:*

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Os menores principais de primeira ordem de A são os elementos na diagonal principal de A, (a_{11}, a_{22}, a_{33}). Os menores principais de segunda ordem de A são:

$$det\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix}, \quad det\begin{bmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{bmatrix} \; e \quad det\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

Neste caso, o único menor principal de terceira ordem de A é o determinante da própria matriz:

$$det\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

A definitividade através dos menores principais é diretamente influenciada pelo número de determinantes de várias ordens e este número é exponencial. Ou seja, existem $\begin{pmatrix} n \\ k \end{pmatrix}$ menores principais de ordem k em A, resultando em:

$$\begin{pmatrix} n \\ 1 \end{pmatrix} + \begin{pmatrix} n \\ 2 \end{pmatrix} + \cdots + \begin{pmatrix} n \\ n \end{pmatrix} = 2^n - 1$$

determinantes de várias ordens.

Através dos menores principais, pode-se fazer a definitividade de uma forma quadrática racional $q_{\mathbb{Q}}$, definida através de uma matriz A, da seguinte forma:

- Positiva definida, se e somente se todos os menores principais de A são maiores que zero;
- Positiva semidefinida, se e somente se todos os menores principais de A são não-negativos;
- Negativa definida, se e somente se $-A$ é positiva definida;
- Negativa semidefinida, se e somente se $-A$ é positiva semidefinida; e
- Indefinida, se e somente se não for nenhum dos casos anteriores.

O método de expansão direta é implementado através do Algoritmo B.2. O comando de repetição das linhas 3 a 5 percorre a diagonal principal da matriz A e escreve os menores principais de primeira ordem. A função MENORES_PRINCIPAIS escreve todos os demais menores principais da matriz A, inclusive os repetidos caso existam, e os respectivos determinantes. Desde que existem 2^n menores principais, o algoritmo têm custo exponencial.

Algoritmo B.2: EXPANSÃO_DIRETA(A)

Entrada: $A_{n\times n}$ = matriz quadrada de ordem n.

Saída: Todos os menores principais de A.

```
1  D ← det(A) // calcula o determinante de A.
2  escreva A, escreva D
3  para cada (i ← 1...n) faça
4  |  escreva A(i,i) // mostra os menores principais de primeira ordem.
5  fim
6  MENORES_PRINCIPAIS(A,n) // aciona a função MENORES_PRINCIPAIS
7  função MENORES_PRINCIPAIS(A,n){
8  d ← n
9  se (d > 2) então
10 |  para cada (i ← 1...d) faça
11 |  |  se (i = 1) então
12 |  |  |  B ← A.matriz_LC([1...d−1],[1...d−1])
13 |  |  senão
14 |  |  |  se (i = d) então
15 |  |  |  |  B ← A.matriz_LC([2...d],[2...d])
16 |  |  |  senão
17 |  |  |  |  B
   |  |  |  |  ← A.matriz_LC([1...d−i,d−i+2...d],[1...d−i,d−i+2...d])
18 |  |  |  fim
19 |  |  fim
20 |  |  D ← det(B) // calcula o determinante de B.
21 |  |  escreva B, escreva D
22 |  |  MENORES_PRINCIPAIS(B,d) // aciona a função MENORES_PRINCIPAIS
23 |  fim
24 fim
25 }
```

B.2 Definitividade Através de Limites para Raízes

A definitividade de uma forma quadrática racional $q_{\mathbb{Q}}$ pode ser feita através do polinômio característico da matriz relacionada a $q_{\mathbb{Q}}$. Para isto, é necessário extrair o polinômio característico de $q_{\mathbb{Q}}$ e, através deste, identificar os limites superiores e inferiores para as raízes deste polinômio. Assim, se todas as raízes estiverem em um

intervalo em que o limite inferior é maior que zero, $q_{\mathbb{Q}}$ é positiva definida. De forma análoga, pode-se decidir sobre os demais tipos de formas quadráticas.

A seguir, descrevemos como a definitividade de formas quadráticas racionais pode ser feita através das raízes do polinômio característico. Primeiro, mostramos como extrair o polinômio característico de uma matriz. Em seguida, apresentamos o método de *Cauchy* para determinar limites inferiores e superiores para as raízes do polinômio. A divisão sintética é usada para verificar se zero é uma raiz do polinômio. Por fim, descrevemos um algoritmo que utiliza estes métodos para fazer a definitividade da forma quadrática.

B.2.1 Extraindo o Polinômio Característico

Definição B.4 *[28] O* **polinômio característico** *de uma matriz A é o determinante $P(\lambda) = det(\lambda I_n - A) = \lambda^n + p_1\lambda^{n-1} + \cdots + p_{n-1}\lambda + p_n$ na variável λ de grau n. A equação $det(\lambda I_n - A) = 0$ é dita* **equação característica** *de A.*

Encontrar as raízes de um polinômio pode ser descrito como o problema de álgebra linear de encontrar todos os números λ que satisfazem a equação $Av = \lambda v$. Neste caso, λ são os auto valores (do inglês *eigenvalues*), e v os correspondentes auto vetores (do inglês *eigenvectors*). Todos os auto valores devem satisfazer a equação $det(\lambda I_n - A) = 0$ e, desde que a equação resultante é um polinômio de grau n em λ, A tem exatamente n auto valores reais possivelmente repetidos.

Um dos métodos mais conhecidos para se extrair o polinômio característico de uma matriz é o método de *Leverrier*[2] [27]. Este método permite encontrar o polinômio característico de qualquer matriz quadrada usando o traço da matriz A^k (ou a soma dos elementos na diagonal principal de A^k), onde $k = 1, 2, \ldots, n$. Desta forma, os coeficientes do polinômio característico $P(\lambda) = det(\lambda I_n - A)$ podem ser calculados através da seguinte fórmula:

$$-p_1 = s_1, \tag{B-1a}$$

$$-kp_k = s_k + \sum_{i=1}^{k-1} p_i \cdot s_{k-i} \tag{B-1b}$$

onde $s_k = traço(A^k)$ e $k = 1, 2, \ldots, n$.

[2] Este método foi modificado diversas vezes e recebeu consideráveis melhoramentos, veja outras versões em [23, 27].

O método de *Leverrier* é implementado no Algoritmo B.3. Desde que a multiplicação de matrizes nas linhas 4 e 5 tem custo $O(n^3)$, e deve ser realizada n vezes, o Algoritmo B.3 têm complexidade de tempo $O(n^4)$.

Algoritmo B.3: METODO_LEVERRIER(A)

Entrada: $A_{n\times n}$ = matriz quadrada de ordem n.
Saída: p = polinômio característico da matriz A.

1 $B \leftarrow identidade(A)$ // B recebe a matriz identidade de ordem n.
2 **para cada** $(i \leftarrow 1 \ldots n)$ **faça**
3 $\quad p(i) \leftarrow -\frac{trace(A\cdot B)}{i}$ // *trace* retorna o traço de $(A \cdot B)$.
4 $\quad B \leftarrow (A \cdot B) + (p(i) \cdot eye(n))$
5 **fim**
6 $p = [1, p]$
7 **retorna** p

Encontrar (com precisão) os auto valores de um polinômio em geral é uma tarefa difícil. Isto porque os auto valores são números reais e, possivelmente, com uma grande quantidade de casas decimais. Uma outra opção é, após identificar o polinômio característico, determinar limites inferiores e superiores para as raízes deste polinômio.

B.2.2 Limites para Raízes de Polinômios

Encontrar limites inferiores e superiores para raízes de um polinômio equivale a encontrar um intervalo que contenha todas as raízes deste polinômio. No entanto, é suficiente restringir a busca aos limites superiores. Isto porque o limite inferior *li* para valores de raízes positivas de um polinômio $P(\lambda)$, de grau n, pode ser encontrado através do limite superior *ls* para valores de raízes positivas de $\lambda^n f(\frac{1}{\lambda})$ e então definir $li = \frac{1}{ls}$ [49].

Atualmente, existem métodos de complexidade linear e de complexidade quadrática, com pequenas variações na precisão dos limites, que podem ser usados para encontrar limites superiores para raízes de polinômios. Em geral, os métodos de complexidade linear encontram limites superiores da seguinte forma:

1. Emparelham cada coeficiente negativo do polinômio com um dos coeficientes positivos ainda não emparelhados; e
2. Calculam a maior radiciação destes valores com relação ao índice de cada coeficiente negativo no polinômio.

Uma das opções mais eficientes para se determinar limites superiores para raízes de um polinômio é através do método de *Cauchy* [49]. Seja $P(\lambda)$ um polinômio de grau $n > 0$, com pelo menos um coeficiente $\alpha_{n-k} < 0, 1 \leqslant k \leqslant n$. Se x é o número de coeficientes negativos, então um limite superior sobre os valores de raízes positivas de $P(\lambda)$ é dado por:

$$ls = max_{\{1 \leqslant k \leqslant : \alpha_{n-k} < 0\}} \sqrt[k]{-\frac{x\alpha_{n-k}}{\alpha_0}} \tag{B-2}$$

ou equivalentemente por:

$$ls = max_{\{1 \leqslant k \leqslant : \alpha_{n-k} < 0\}} \sqrt[k]{-\frac{\alpha_{n-k}}{\frac{\alpha_0}{x}}} \tag{B-3}$$

Exemplo B.5 *Para o polinômio* $\lambda^9 + 3\lambda^8 + 2\lambda^7 + \lambda^6 - 4\lambda^4 + \lambda^3 - 4\lambda^2 - 3$ *são feitas as radiciações:* $\left(\sqrt[4]{-\frac{-4}{\frac{1}{3}}}\right) = 1.861, -\left(\sqrt[6]{-\frac{-4}{\frac{1}{3}}}\right) = 1.513, -\left(\sqrt[7]{-\frac{-3}{\frac{1}{3}}}\right) = 1.368$, *e o valor* 1.861 *é um limite superior para raízes positivas deste polinômio.*

De maneira geral, para um polinômio $P(\lambda)$ com x coeficientes negativos, o método de *Cauchy* primeiramente divide seu coeficiente de maior grau α_n em x partes e então emparelha cada parte com um coeficiente negativo ainda não emparelhado. O método de *Cauchy* é implementado no Algoritmo B.4.

Algoritmo B.4: METODO_CAUCHY(P_n)

Entrada: P_n = vetor com os n coeficientes de $P(\lambda)$.

Saída: ls = limite superior sobre os valores de raízes positivas de $P(\lambda)$.

```
1  α ← P(λ)                          // α é um vetor com os coeficientes de P(λ)
2  x ← número de elementos negativos de α
3  ls ← 0
4  se (n + 1 ⩽ 1 ou x = 0) então
5  |  ls ← 0
6  senão
7  |  para (i ← 0...n − 1) faça
   |  |                 // i ← 0 para referenciar corretamente a primeira posição em α
8  |  |  se (α(i) < 0) então
9  |  |  |  tempub ← −( α(i) / (α(0)/x) )^(1/i)
10 |  |  |  se (tempub > ls) então
11 |  |  |  |  ls ← tempub
12 |  |  |  fim
13 |  |  fim
14 |  fim
15 fim
16 retorna ls
```

B.2.3 Divisão Sintética

A **divisão sintética** [2] é uma forma eficiente de dividir um polinômio por binômio do tipo $\lambda - a$, tal que A é uma constante. A Figura B.1 mostra como pode ser feita a divisão sintética do polinômio $P(\lambda) = 2\lambda^6 - \lambda^4 + \lambda^3 + 6$ por $\lambda + 1$. Neste caso, os coeficientes de $P(\lambda)$ são colocados na primeira linha. A segunda linha corresponde aos valores adquiridos a partir da multiplicação dos coeficientes pelo valor de A e a terceira linha armazena o polinômio resultante.

Por exemplo, em b), é feita a multiplicação de 2 (coeficiente da variável de maior grau de $P(\lambda)$) por -1 (valor de A), e resultado é somado com 0 (o próximo coeficiente de $P(\lambda)$). Os demais elementos são calculados da mesma forma. Quando o último coeficiente do polinômio é utilizado no processo, o resto da divisão estará localizado na última coluna da terceira linha. Assim, o polinômio resultante da divisão de $P(\lambda) = 2\lambda^6 - \lambda^4 + \lambda^3 + 6$ por $\lambda + 1$ é formado pelos valores da terceira linha $(2, -2, 1, 0, 0, 0)$, sendo que o grau do polinômio resultante é exatamente uma unidade a menos com relação ao grau de $P(\lambda)$). O Algoritmo B.5 implementa este processo.

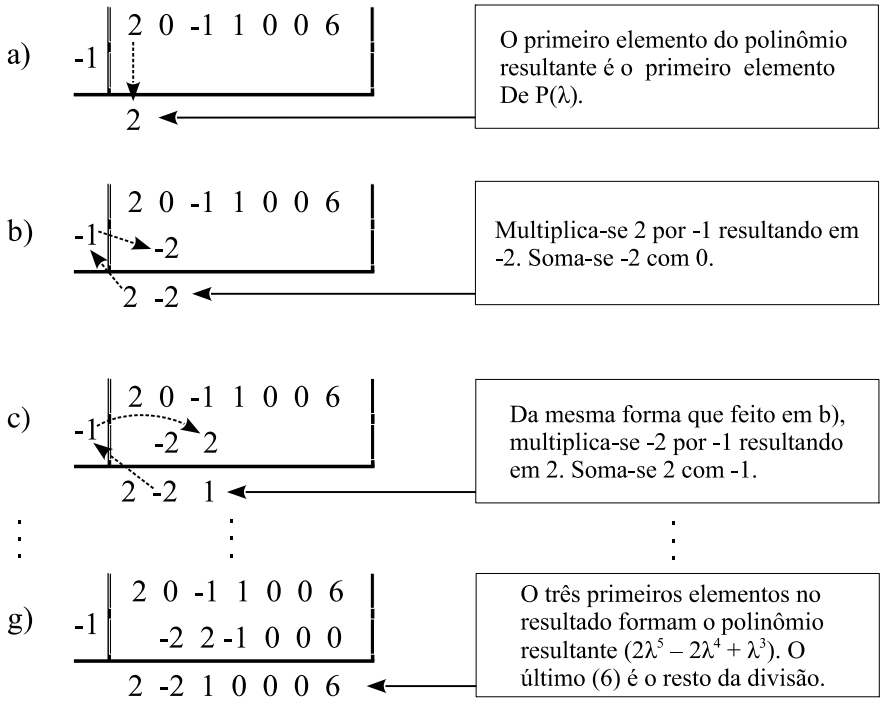


Figura B.1: *Divisão sintética de* $P(\lambda) = 2\lambda^6 - \lambda^4 + \lambda^3 + 6$ *por* $\lambda + 1$. *O polinômio resultante é* $2\lambda^5 - 2\lambda^4 + \lambda^3$ *e a divisão teve resto* 6.

Algoritmo B.5: DIV_SINTETICA(P_n, a)

Entrada: P_n = vetor com os n coeficientes de $P(\lambda)$; A = dividendo.

Saída: *res* = resto da divisão sintética de $P(\lambda)$ por A.

```
1 mult ← 0
2 para cada (i ← 1...n) faça
3 |   res ← P(i) + mult
4 |   mult ← res · a
5 fim
6 retorna res
```

A divisão sintética também permite verificar se A é um limite superior, um limite inferior ou se A é uma raiz do polinômio $P(\lambda)$, da seguinte forma:

- A é um **limite inferior** de $P(\lambda)$ se A é negativo e todos os elementos da terceira linha têm sinais alternados;
- A é um **limite superior** de $P(\lambda)$ se A é positivo e todos os elementos na terceira linha têm o mesmo sinal (positivos ou negativos);
- A é uma **raiz** de $P(\lambda)$ se o resto da divisão é zero.

B.2.4 Definitividade Através de Limites

Esta subseção tem como objetivo apresentar um método para identificar o tipo de formas quadráticas através de limites superiores para raízes de um polinômio. O Algoritmo B.6 implementa este método. Na linha 1, o método de *Leverrier* foi utilizado para extrair o polinômio da matriz A e na linha 2, o método de *Cauchy* foi utilizado para extrair o limite superior lim_sup deste polinômio. Na linha 4, a divisão sintética foi utilizada para verificar se zero é raiz deste polinômio e, com estas informações, decidir o tipo da forma quadrática relacionada a A.

Algoritmo B.6: DEF_LIMITESUPERIOR(A)

Entrada: A = matriz relacionada a forma quadrática $q_{\mathbb{Q}}$.
Saída: definitividade de $q_{\mathbb{Q}}$.

```
1  p ← METODO_LEVERRIER(A)
2  lim_sup ← METODO_CAUCHY(p)
3  lim_inf ← 1/lim_sup
4  r0 ← DIV_SINTETICA(p, 0)
5  se (lim_inf > 0) então
6  |  retorna "q_Q é Positiva Definida"
7  fim
8  se (lim_sup < 0) então
9  |  retorna "q_Q é Negativa Definida"
10 fim
11 se (lim_inf ⩾ 0) então
12 |  se (r0 = 0) então retorna "q_Q é Positiva Semidefinida"
13 fim
14 se (lim_sup ⩽ 0) então
15 |  se (r0 = 0) então retorna "q_Q é Negativa Semidefinida"
16 fim
17 se (lim_inf < 0) e (lim_sup > 0) então
18 |  retorna "q_Q é Indefinida"
19 fim
```

O Algoritmo B.6 tem complexidade de tempo polinomial, sendo que o método de *Leverrier* tem complexidade $O(n^3)$, de *Cauchy* tem complexidade $O(n)$ e a divisão sintética $O(n)$. No entanto, o Algoritmo B.6 não consegue determinar com precisão o tipo da forma quadrática porque o método de *Cauchy* retorna uma aproximação do limite superior.

B.3 Definitividade Através de Manipulações na Matriz

Definição B.6 *[39] Seja A uma matriz simétrica. O* **problema dos auto valores** *pode ser definido como o problema de se encontrar o conjunto de auto valores* λ *e auto vetores x que satisfazem a equação* $Ax = \lambda x$.

Os auto valores de uma matriz simétrica A são usados para decidir o tipo da matriz A e, desde que toda forma quadrática $q_{\mathbb{Q}}$ pode ser representada por uma matriz simétrica, os auto valores podem ser usados para decidir o tipo de $q_{\mathbb{Q}}$.

Definição B.7 *[39] Seja A a matriz simétrica relacionada à forma quadrática* $q_{\mathbb{Q}}$. *Sejam* $\lambda_1, \ldots, \lambda_n$ *os auto valores de A. Então* $q_{\mathbb{Q}}$ *é:*

- *Positiva definida se* $\lambda_i > 0$, *para* $i = 1, \ldots, n$;
- *Negativa definida se* $\lambda_i < 0$, *para* $i = 1, \ldots, n$;
- *Positiva semidefinida se* $\lambda_i \geqslant 0$, *para* $i = 1, \ldots, n$;
- *Negativa semidefinida se* $\lambda_i \leqslant 0$, *para* $i = 1, \ldots, n$ *e*
- *Indefinida se existe algum i tal que* $\lambda_i < 0$, *e se existe algum j tal que* $\lambda_j > 0$, *para* $i, j \in \{1, \ldots, n\}$.

As subseções seguintes apresentam dois métodos para encontrar os auto valores de uma matriz (*QR* e *Jacobi*) e uma forma de encontrar os pivôs de uma matriz (decomposição de *Cholesky*). Após a execução destes métodos, os auto valores estarão em uma matriz diagonal de auto valores (no caso dos métodos *QR* e *Jacobi*) ou em uma matriz diagonal superior (no caso da decomposição de *Cholesky*).

Cada iteração do método *QR* tem custo $O(n^3)$. No entanto, este custo pode cair para $O(n)$ ou $O(n^2)$ se a matriz é tridiagonal *Hermetiana* ou *Hessenberg* superior [36], respectivamente. O algoritmo de *Jacobi* tem complexidade $O(n^3)$ no melhor caso [40] e é bastante utilizado por ser considerado um dos métodos mais confiáveis. Por fim, o pivotamento de matrizes através da fatoração *Cholesky* pode ser feito com $O(n^3)$ operações [47].

B.3.1 O Método *QR*

O método *QR* encontra todos os auto valores (eventualmente os auto vetores) de uma matriz simétrica em dois estágios separados, são eles:

- Decomposição *QR*. A matriz simétrica original é transformada em uma matriz diagonal superior em um número finito de passos;
- Iterações *QR*. Transformações de similaridade são aplicadas de forma que a matriz resultante convirja para uma matriz em que os auto valores possam ser facilmente extraídos.

Seja $A_{m\times n}$ uma matriz em que as colunas $a_1,\ldots,a_n$ são linearmente independentes. Então existe a decomposição $A = QR$, onde $Q_{m\times n}$ é uma matriz na qual as colunas $v_1,\ldots,v_n$ formam um sistema ortonormal, e $R_{n\times n}$ é uma matriz triangular superior.

Definição B.8 *[7] Para $v_1,\ldots,v_n$ formar um sistema ortonormal é necessário que:*

1. *Todos os vetores sejam diferentes de zero e pares ortogonais. Por exemplo, $\langle v_i, v_i\rangle \neq 0\ \forall i$; e $\langle v_i, v_j\rangle = 0\ \forall i \neq j$;*
2. *Todos os vetores sejam normalizados para a unidade. Por exemplo, $\langle v_i, v_i\rangle = 1\ \forall i$.*

O método de *Gram-Schmidt* [7, 35] pode ser usado para calcular as entradas para Q e R. Seja $a_1,\ldots,a_n$ colunas linearmente independentes. A decomposição $A = QR$ é feita através do método de *Gram-Schmidt* em dois passos. Primeiro, define-se o sistema ortogonal de vetores u_i, para $i = 1,\ldots,n$, e então normalizam-se os vetores u_i para encontrar v_i:

$$\begin{aligned}
u_1 &= a_1 & v_1 &= \frac{u_1}{\| u_1 \|}\\
u_2 &= a_2 - \frac{\langle u_1, a_2\rangle}{\langle u_1, u_1\rangle}\cdot u_1 & v_2 &= \frac{u_2}{\| u_2 \|}\\
u_3 &= a_3 - \frac{\langle u_1, a_3\rangle}{\langle u_1, u_1\rangle}\cdot u_1 - \frac{\langle u_2, a_3\rangle}{\langle u_2, u_2\rangle}\cdot u_2 & v_3 &= \frac{u_3}{\| u_3 \|}\\
&\vdots & &\vdots\\
n_n &= a_n - \sum_{i=1}^{n-1}\frac{\langle u_i, a_n\rangle}{\langle u_i, u_i\rangle}\cdot u_i & v_n &= \frac{u_n}{\| u_n \|}
\end{aligned} \tag{B-4}$$

Os vetores $v_1,\ldots,v_n$ são colunas de Q. A matriz R será definida como segue:

$$R = \begin{pmatrix} \| u_1 \| & \langle v_1, a_2\rangle & \ldots & \langle v_1, a_n\rangle \\ 0 & \| u_2 \| & \ldots & \langle v_2, a_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \| u_n \| \end{pmatrix}$$

Exemplo B.9 *A decomposição QR da matriz* $A = \begin{pmatrix} 1 & 3 & 3 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \\ -1 & -3 & -1 \end{pmatrix}$ *é dada como segue:*

- *Primeiro encontramos os vetores normalizados:*

$\langle v_i, a_i\rangle$	a_1	a_2	a_3
v_1	2	2	2
v_2	0	4	2
v_3	0	0	2

- *Finalmente as matrizes Q e R.*

$$Q = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \\ -1 & -1 & 1 \end{pmatrix}, R = \begin{pmatrix} 2 & 2 & 2 \\ 0 & 4 & 2 \\ 0 & 0 & 2 \end{pmatrix}.$$

O Algoritmo B.7 descreve como são realizadas as iterações QR. A matriz resultante da decomposição $A = QR$ é ortogonal e então, $R = Q^t A$. Desta forma, $RQ = Q^t AQ$ é uma transformação de similaridade que conserva os auto valores, e a sequência de transformações em A converge para uma matriz tridiagonal de auto valores.

Algoritmo B.7: ALGO_QR(A)

Entrada: $A_{n\times n}$ = matriz quadrada de ordem n.

Saída: matriz com auto valores na diagonal principal.

```
1  conv ← falso
2  repita
3  |  Q,R ←GRAM_SCHMIDT(A)
4  |  B ← R · Q
5  |  se (A = B) então
6  |  |  conv ← verdadeiro
7  |  senão
8  |  |  A ← B
9  |  fim
10 até (conv)
11 retorna A
```

B.3.2 O Método de *Jacobi*

O método de *Jacobi* [7] executa uma sequência de transformações (rotações no plano) que vão a cada passo eliminando elementos fora da diagonal principal de uma matriz simétrica A e transformando A em uma matriz diagonal de auto valores.

Definição B.10 (Matrizes de Rotação no Plano)*[35]. Matrizes ortogonais na forma:*

$$P_{ij} = \begin{array}{c} \\ \\ \text{linha } i \\ \rightarrow \\ \\ \text{linha } j \\ \rightarrow \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & \text{col } i & & \text{col } j & & \\ & & \downarrow & & \downarrow & & \end{array} \\ \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & & s & & \\ & & & 1 & & & \\ & & & & \ddots & & \\ & & -s & & c & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{pmatrix} \end{array}$$

tal que $c^2 + s^2 = 1$ *são chamados de matrizes de rotação no plano porque executam um rotação no plano-*(i, j) *de* $\mathbb{R}^n$*. As entradas c e s designam uma rotação no ângulo* θ*.*

Tais matrizes de rotações no plano são conhecidas como rotações *Givens*. Aplicar P_{ij} a $0 \neq x \in \mathbb{R}^n$ gira as coordenadas (i, j) de x na forma que:

$$P_{i,j}x = \begin{array}{c} \\ i \\ \rightarrow \\ \\ j \\ \rightarrow \\ \\ \end{array} \begin{array}{c} x_1 \\ \begin{pmatrix} \vdots \\ cx_i + sx_j \\ \vdots \\ -sx_i + cx_j \\ \vdots \\ x_n \end{pmatrix} \end{array}.$$

Portanto, se x_i e x_j não são ambos zero, e se definirmos:

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}} \quad \text{e} \quad s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}}, \tag{B-5}$$

podemos eliminar (zerar) qualquer componente através de uma rotação no plano (i, j) sem afetar qualquer entrada x_i e x_j. Por exemplo, para eliminar o j-ésimo elemento:

$$P_{i,j}x = \begin{array}{c} \\ i \\ \rightarrow \\ \\ j \\ \rightarrow \\ \\ \end{array} \begin{array}{c} x_1 \\ \begin{pmatrix} \vdots \\ \sqrt{x_i^2 + x_j^2} \\ \vdots \\ 0 \\ \vdots \\ x_n \end{pmatrix} \end{array}.$$

O algoritmo de diagonalização de *Jacobi* [35] começa com $A_0 = A$, e produz uma sequência de matrizes $A_k = P_k^T A_{k-1} P_k$, onde o k-ésimo passo P_k é uma rotação no plano construída para eliminar a maior entrada fora da diagonal em A_{k-1}. Portanto, se a_{ij} é a maior entrada em A_{k-1}, então P_k faz a rotação no plano-(i, j) definindo:

$$s = \frac{1}{\sqrt{1+\sigma^2}} \text{ e } c = \frac{\sigma}{\sqrt{1+\sigma^2}} = \sqrt{1-s^2}, \text{ onde } \sigma = \frac{(a_{ii} - a_{jj})}{2a_{ij}}. \tag{B-6}$$

Para $n > 2$, temos que $\| A'_k \|_F^2 \leqslant \left(1 - \frac{2}{n^2-n}\right)^k \| A'_k \|_F^2 \to 0$, da mesma forma que $k \to \infty$. Portanto, se $P^{(k)}$ é uma matriz ortogonal definida por $P^{(k)} = P_1 P_2 \ldots P_k$, então:

$$\lim_{k\to\infty} P^{(K)^T} A P^{(k)} = \lim_{k\to\infty} A_k = D \tag{B-7}$$

é uma matriz diagonal.

B.4 Pivotamento de Matrizes

Definição B.11 (Forma Escalonada)*[41] Uma matriz está na* **forma escalonada** *(do inglês row echelon) se as linhas com todas as entradas iguais a zero estão na parte inferior da matriz e se o primeiro elemento diferente de zero da $(k+1)$-ésima linha está à direita da primeira entrada diferente de zero da k-ésima linha.*

Exemplo B.12 *A matriz A é um exemplo de matriz na forma escalonada. A matriz B possui uma linha com todas as entradas iguais a zero (linha 3) acima de uma linha com uma entrada diferente de zero (linha 4) portanto, não está na forma escalonada.*

$$A = \begin{pmatrix} 1 & 2 & 3 & 5 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 2 & 3 & 5 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Se uma matriz está na forma escalonada, então a primeira entrada diferente de zero a partir da esquerda de cada linha é chamada de **pivô**. O pivotamento de matrizes é considerado uma das maneiras mais eficientes de determinar se uma matriz é positiva definida. Este processo é baseado na Eliminação de *Gauss* [6, 47] e tem custo $O(n^3)$.

Para qualquer matriz simétrica A, os sinais dos pivôs coincidem com os sinais dos auto valores. Sobretudo, a matriz Λ de auto valores e a matriz dos pivôs D possuem o mesmo número de elementos positivos, negativos e iguais a zero.

Uma das propriedades de uma matriz positiva definida é que estas podem ser decompostas na forma $A = R^t R$, para uma matriz triangular superior R com elementos

positivos na diagonal principal (este processo é conhecido como decomposição de *Cholesky*). O leitor pode encontrar mais detalhes e exemplos sobre este processo em [35, 41].

APÊNDICE C

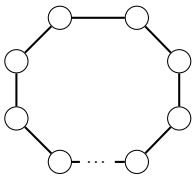
Apêndice do Capítulo 3

C.1 Exemplos de Grafos Críticos e Hipercríticos

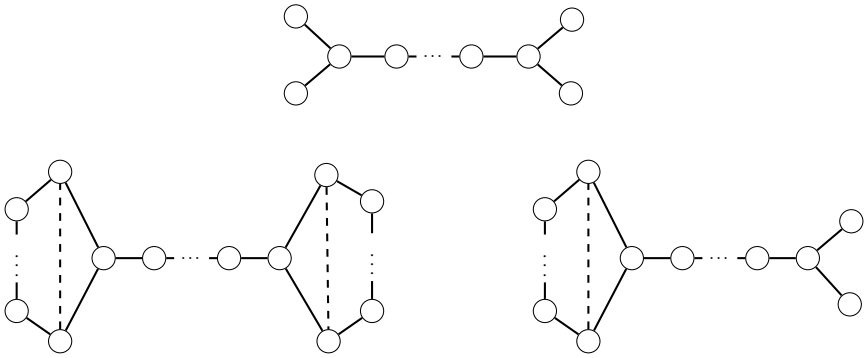
Esta seção descreve alguns exemplos de grafos que podem ser usados para identificar formas inteiras que não são fracamente positivas (restrições críticas) e grafos que podem ser usados para identificar formas inteiras que não são fracamente nãonegativas (restrições hipercríticas). Neste apêndice, as famílias $\widetilde{\mathbb{A}}_n$ e $\widetilde{\mathbb{D}}_n$ são completas. No entanto, existe um conjunto maior de grafos críticos e hipercríticos. Todos os grafos críticos podem ser encotrados em [50] e os hipercríticos em [48].

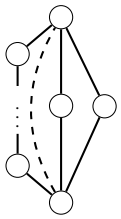
C.1.1 Grafos Críticos

Grafos do tipo $\widetilde{\mathbb{A}}_n$

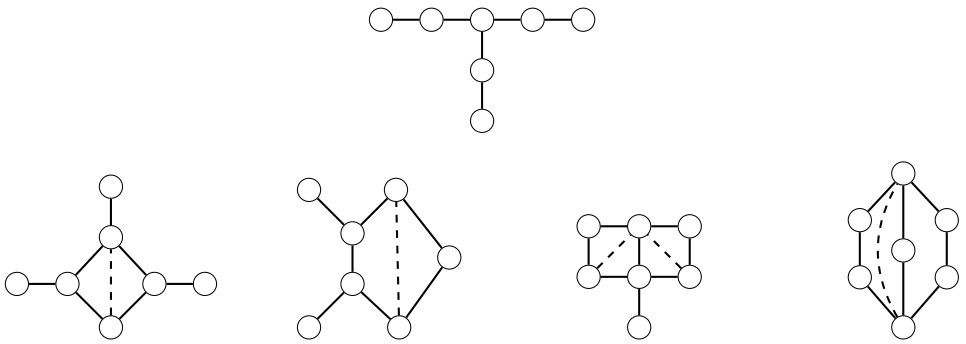


Grafos do tipo $\widetilde{\mathbb{D}}_n$

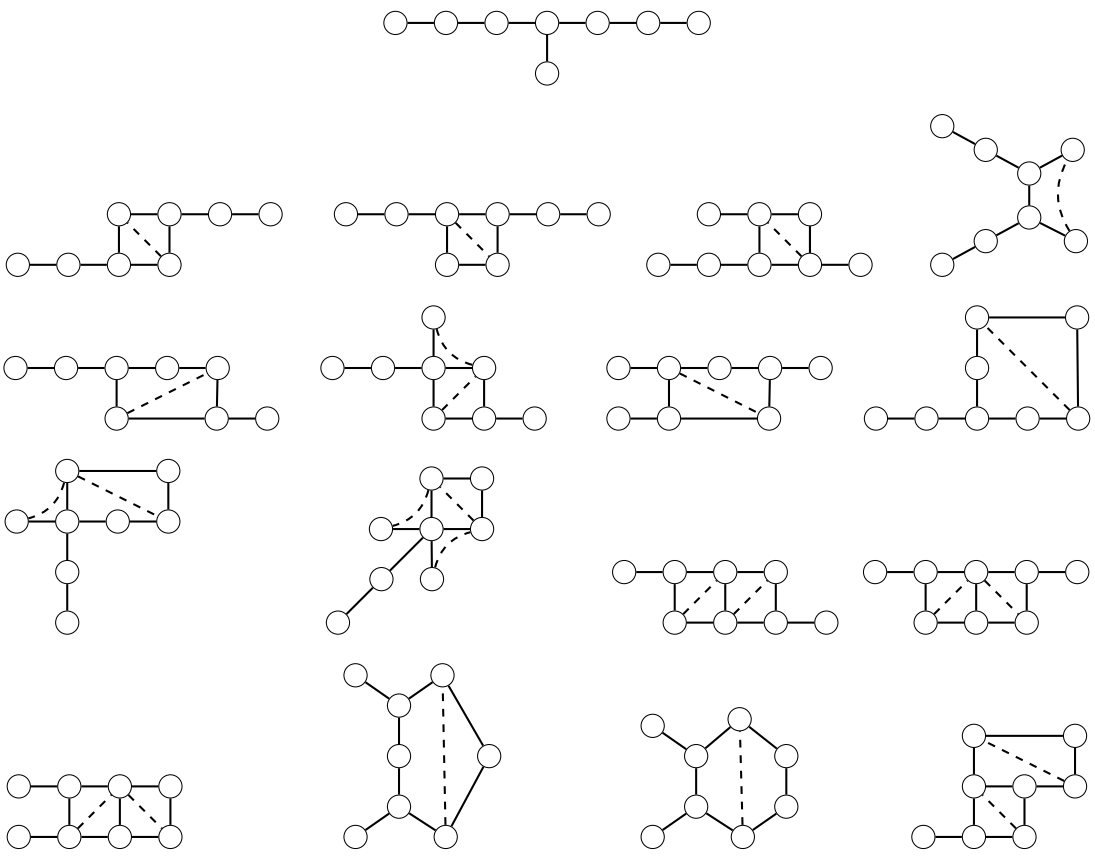


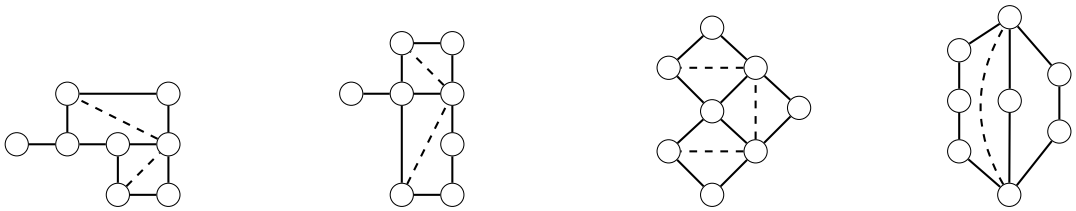


Grafos do tipo $\widetilde{\mathbb{E}}_6$



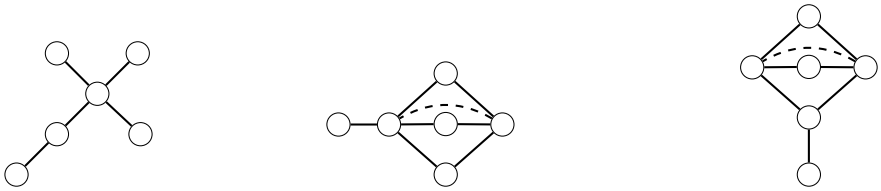
Grafos do tipo $\widetilde{\mathbb{E}}_7$



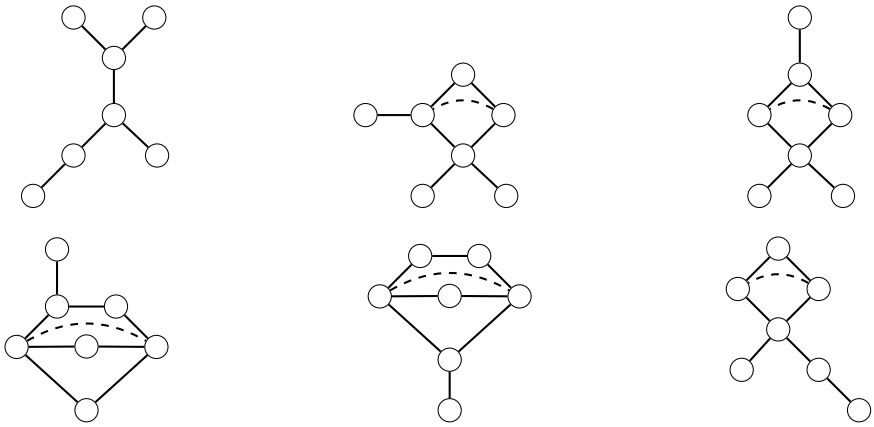


C.1.2 Grafos Hipercríticos

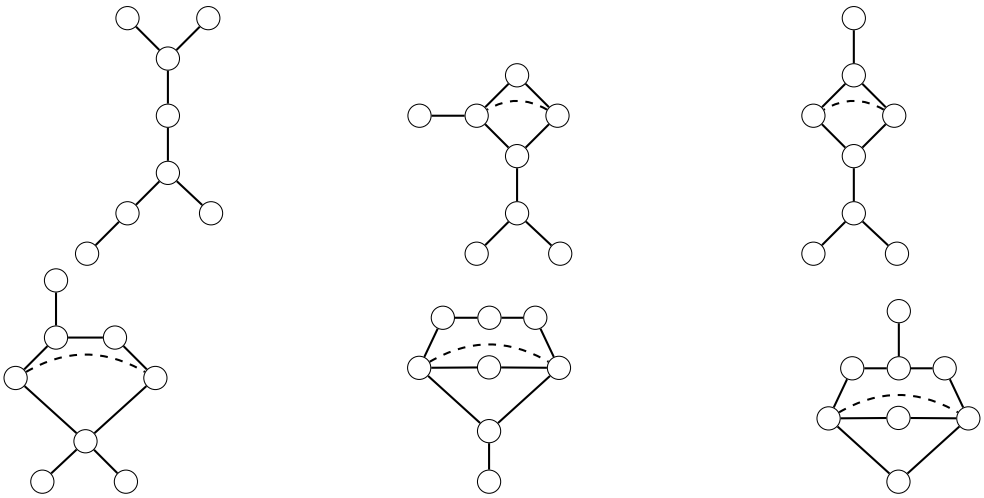
Grafos do tipo $\tilde{\tilde{\mathbb{D}}}_4$

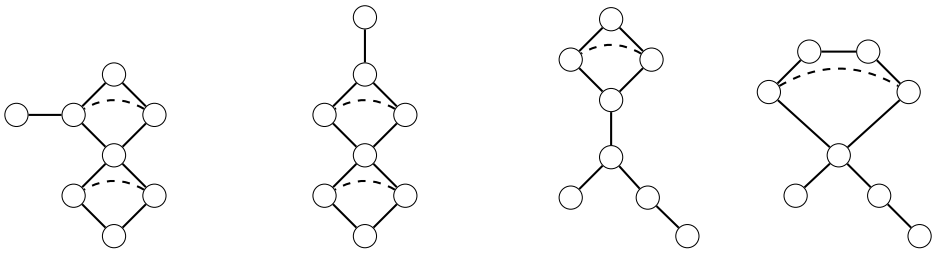


Grafos do tipo $\tilde{\tilde{\mathbb{D}}}_5$

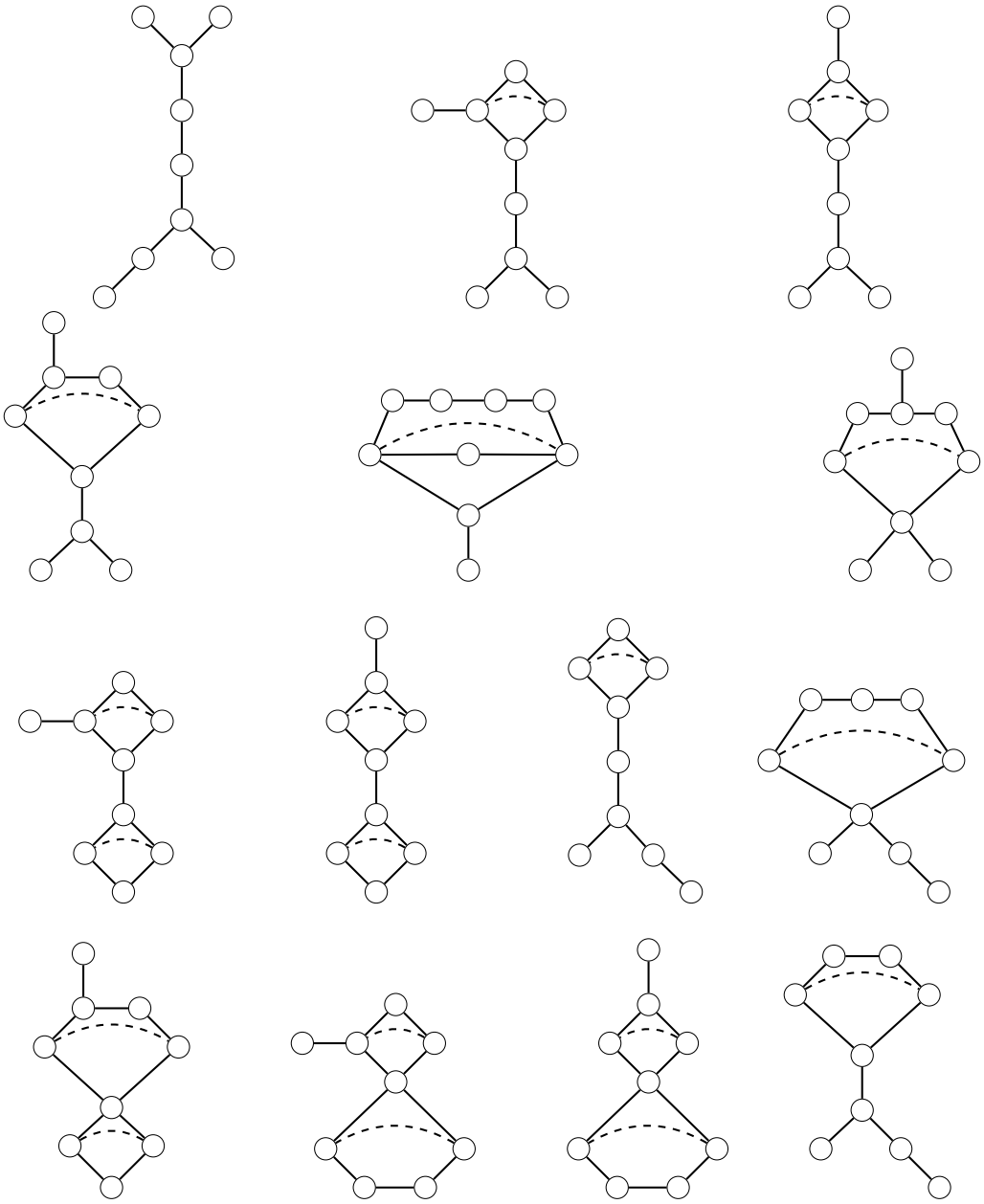


Grafos do tipo $\tilde{\tilde{\mathbb{D}}}_6$





Grafos do tipo $\widetilde{\widetilde{\mathbb{D}}}_7$



C.2 Resultados com Quivers Aleatórios

O experimentos com quivers criados através de uma distribuição aleatória de vértices e arestas foram feitos através da média de 20 testes sobre formas inteiras fracamente positivas com n vértices, da seguinte forma: o primeiro exemplo tem n arestas negativas e $2 \cdot n$ arestas positivas; o segundo $n+1$ e $2 \cdot n$; e o último $n+19$ arestas negativas e $2 \cdot n$ arestas positivas. Para melhorar a confiabilidade da média calculada, o quiver com maior tempo foi desconsiderado. Isto foi necessário porque, para uma quantidade pequena de exemplos, os algoritmos utilizaram um tempo muito maior do que o tempo utilizado com os demais exemplos. Tais experimentos também foram feitos com formas inteiras que não são fracamente positivas e que não são fracamente nãonegativas.

Quivers Aleatórios												
Experimentos para Formas Inteiras Fracamente Positivas												
|V(G)|	**60**		**80**		**100**		**120**		**140**		**160**	
QTD	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**
1	1,42	0,60	8,55	0,40	10,10	1,20	50,89	2,10	92,62	2,10	211,76	2,50
2	1,35	0,70	6,15	0,60	20,70	1,80	39,81	1,30	88,89	2,90	176,53	2,70
3	1,45	0,63	5,77	1,30	29,76	1,99	39,31	2,60	86,65	3,20	176,17	3,40
4	1,43	1,01	7,75	1,96	18,08	2,80	92,30	2,90	85,02	2,99	286,90	3,90
5	1,32	1,50	6,11	4,30	17,76	3,20	156,02	3,50	87,78	3,80	241,31	3,70
6	1,50	3,10	5,94	6,30	17,09	3,45	149,97	3,20	80,32	4,20	556,64	4,89
7	2,28	4,50	11,09	8,20	15,91	3,87	144,90	4,92	78,32	4,60	579,80	4,54
8	2,21	4,20	5,95	8,50	15,15	4,60	128,92	5,80	76,81	5,30	507,18	6,67
9	0,90	6,30	5,81	7,90	55,26	5,10	149,67	7,20	76,50	7,84	488,94	7,20
10	1,91	7,20	5,36	8,86	49,54	4,90	140,92	8,90	144,71	8,73	485,83	7,37
11	1,77	7,40	5,34	9,35	53,93	5,34	184,97	11,80	186,44	10,30	473,20	8,10
12	1,62	8,50	8,45	11,20	50,57	7,10	128,92	11,60	357,25	10,90	431,28	8,50
13	3,39	9,50	8,83	15,90	49,70	7,89	142,93	13,10	346,24	11,80	452,12	9,43
14	8,83	10,10	8,04	17,60	147,09	9,32	311,06	13,79	350,36	11,94	493,50	10,87
15	20,76	13,70	8,31	24,50	135,87	11,67	228,21	15,60	320,82	14,20	482,32	10,90
16	42,22	17,30	33,40	30,50	258,72	14,79	286,16	15,87	611,92	15,89	510,95	12,12
17	40,94	23,10	45,27	23,50	135,40	16,40	157,69	16,40	595,14	23,48	478,56	12,63
18	39,21	22,20	33,40	30,28	142,82	17,65	121,11	16,85	559,50	23,90	995,84	14,31
19	38,80	26,40	32,50	24,40	674,44	20,01	186,01	18,76	817,02	24,84	816,40	14,60
20	42,05	36,20	59,45	35,35	682,49	25,79	137,10	20,50	1689,0	27,26	768,63	16,85
Soma	213,1	168,4	252,0	235,7	1897,8	143,0	2665,8	176,1	5042,3	192,9	8618,0	152,3
Média	11,22	8,87	13,26	12,41	99,89	7,53	140,31	9,27	265,38	10,15	453,58	8,02
Legenda FPT: tempo em segundos para execução de FP_TESTE [10] para formas inteiras fracamente positivas. FPP: tempo em segundos para a execução de FP_POLINOMIAL (veja Algoritmo 3.3).												

Tabela C.1: *Experimentos para Formas Inteiras Fracamente Positivas relacionadas a quivers gerados de forma aleatória.*

Quivers Aleatórios												
Experimentos para Formas Inteiras não Fracamente Positivas												
|V(G)|	**60**		**80**		**100**		**120**		**140**		**160**	
QTD	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**	**FPT**	**FPP**
1	0,33	0,03	0,95	0,04	0,90	0,06	2,35	0,03	3,28	0,04	5,46	0,07
2	0,22	0,03	1,10	0,01	0,69	0,03	2,66	0,01	4,51	0,09	2,47	0,03
3	0,82	0,01	0,96	0,03	0,81	0,02	3,11	0,01	5,14	0,07	6,21	0,08
4	0,72	0,01	1,55	0,03	0,47	0,03	4,26	0,02	6,17	0,04	3,39	0,02
5	1,56	0,01	0,88	0,01	4,20	0,04	1,64	0,02	0,88	0,05	2,86	0,05
6	1,45	0,01	3,12	0,01	2,01	0,05	5,69	0,01	5,29	0,02	3,07	0,02
7	0,68	0,02	1,29	0,01	0,91	0,02	1,45	0,02	8,86	0,07	11,30	0,03
8	0,95	0,01	5,87	0,02	1,85	0,04	1,44	0,01	2,82	0,05	13,20	0,03
9	0,95	0,01	1,05	0,02	1,35	0,01	8,21	0,04	10,80	0,04	5,75	0,02
10	2,79	0,01	1,25	0,02	1,25	0,01	1,01	0,12	3,38	0,02	6,06	0,02
11	0,31	0,02	0,99	0,01	1,85	0,03	2,68	0,01	4,25	0,17	7,85	0,04
12	0,16	0,01	1,13	0,01	1,97	0,01	2,97	0,02	4,42	0,03	4,65	0,02
13	0,18	0,01	1,01	0,01	1,16	0,02	2,63	0,01	1,77	0,03	5,30	0,03
14	0,54	0,01	2,81	0,01	1,64	0,08	2,87	0,02	4,16	0,15	6,00	0,03
15	1,18	0,01	1,02	0,01	1,86	0,02	4,65	0,01	4,21	0,03	8,55	0,05
16	0,33	0,01	2,00	0,01	2,50	0,01	2,79	0,02	5,10	0,07	4,47	0,02
17	0,71	0,01	1,18	0,02	2,62	0,01	3,70	0,02	2,32	0,07	5,11	0,03
18	0,16	0,04	1,57	0,01	2,80	0,03	5,10	0,11	4,36	0,08	7,27	0,02
19	0,40	0,01	3,47	0,02	2,22	0,02	2,54	0,03	5,25	0,05	3,14	0,02
20	0,15	0,01	1,78	0,02	2,83	0,05	4,99	0,03	2,60	0,03	7,60	0,03
Soma	11,80	0,25	29,11	0,30	31,69	0,51	58,53	0,45	78,77	1,03	106,4	0,58
Média	0,62	0,01	1,53	0,02	1,67	0,03	3,08	0,02	4,15	0,05	5,60	0,03
Legenda FPT: tempo em segundos para execução de FP_TESTE [10] para formas inteiras fracamente positivas. FPP: tempo em segundos para a execução de FP_POLINOMIAL (veja Algoritmo 3.3).												

Tabela C.2: *Experimentos para formas inteiras não fracamente positivas relacionadas a quivers gerados de forma aleatória.*

Quivers Aleatórios												
Experimentos para Formas Inteiras não Fracamente Nãonegativas												
|V(G)|	**60**		**80**		**100**		**120**		**140**		**160**	
QTD	**FNT**	**FNP**	**FNT**	**FNP**	**FNT**	**FNP**	**FNT**	**FNP**	**FNT**	**FNP**	**FNT**	**FNP**
1	0,24	0,53	1,40	0,60	1,15	0,45	2,62	0,40	3,73	0,32	5,30	0,12
2	0,71	0,40	1,34	0,60	2,29	0,27	2,47	0,27	4,08	0,28	3,12	0,35
3	0,37	0,19	0,42	0,30	1,97	0,17	3,18	0,35	5,77	0,19	6,68	0,36
4	0,10	0,30	1,57	0,13	2,95	0,12	4,31	0,40	6,19	0,23	7,65	0,14
5	0,17	0,30	1,38	0,49	3,95	0,56	1,75	0,23	5,81	0,22	8,49	0,35
6	0,11	0,30	1,94	0,13	4,20	0,20	6,75	0,20	6,95	0,12	10,07	0,17
7	1,83	0,18	0,74	0,29	1,59	0,37	6,77	0,70	7,57	0,22	9,44	0,87
8	0,13	0,45	0,67	0,72	4,59	1,45	2,15	0,35	4,30	0,44	3,53	0,20
9	0,58	0,82	0,89	0,13	4,87	0,23	2,85	0,30	11,69	0,74	6,85	0,52
10	0,69	0,85	1,05	0,32	2,08	1,56	1,18	0,30	8,06	1,50	2,43	0,10
11	0,10	0,75	0,28	0,82	2,07	2,90	2,97	0,64	5,29	0,10	5,30	0,11
12	0,44	1,20	0,28	0,22	1,77	1,22	3,93	0,70	4,83	0,12	3,97	0,25
13	0,54	0,50	1,05	0,32	3,38	0,38	4,25	0,84	4,52	0,11	7,35	0,12
14	0,17	9,35	0,39	0,37	1,30	1,97	5,32	2,22	3,80	0,23	5,93	0,86
15	0,15	3,19	0,43	1,93	2,37	0,91	3,28	1,85	6,56	0,35	4,61	0,12
16	0,20	8,50	2,86	2,82	2,13	0,83	2,64	3,10	4,13	0,98	3,82	1,68
17	0,16	16,57	2,00	2,86	2,68	0,30	4,57	2,86	5,01	1,19	5,27	1,54
18	0,21	1,95	0,71	0,95	3,42	0,67	5,15	1,35	5,60	2,68	8,67	0,75
19	0,15	3,33	0,52	0,51	4,52	1,10	4,80	3,88	5,63	0,16	8,60	0,16
20	0,05	0,12	0,65	1,85	4,13	0,88	3,30	1,71	5,35	1,02	7,77	0,36
Soma	5,27	33,21	17,71	13,50	52,54	13,64	67,47	18,77	103,1	8,52	114,7	7,45
Média	0,28	1,75	0,93	0,71	2,77	0,72	3,55	0,99	5,43	0,45	6,04	0,39
Legenda FNT: tempo em segundos para execução de FNN_TESTE (veja Algoritmo 3.1) FNP: tempo em segundos para a execução de FNN_POLINOMIAL (veja Algoritmo 3.2)												

Tabela C.3: *Experimentos para formas inteiras não fracamente nãonegativas relacionadas a quivers gerados de forma aleatória.*

APÊNDICE D

Apêndice do Capítulo 4

A função LISTA_VERDES(T) (veja Algoritmo D.1) preenche a lista L_v com vértices verdes da matriz sendo processada T. Assim, se o vértice i sendo analisado recebe uma ou mais arestas de um vértice não congelado j (linha 5), então i é vermelho. Caso contrário i é verde.

Algoritmo D.1: LISTA_VERDES(T)

Entrada: T = matriz de troca.
Saída: L_v = lista de vértices verdes.

```
1  L_v ← ∅
2  para (i ← 1...n) faça
3      verde ← 1                                   // O vértice i é verde
4      para (j ← 1...n) faça
                  // Identifica arestas incidentes de vértices congelados
5          se (T(i, n+j) ⩽ −1) então
6              verde ← 0                           // O vértice i é vermelho
7              sair      // Interrompe o laço (linha 4) e analisa o próximo i
8          fim
9      fim
10     se (verde) então
11         L_v ← i                                 // Inclui i na Lista L_v
12     fim
13 fim
14 retorna L_v
```

A função SELECIONAR_VERTICE(T) (veja Algoritmo D.2) seleciona um vértice verde de $\widetilde{Q}$ (a partir de T) e informa quando a sequência maximal deve ser finalizada (quando $v = 0$). A lista de vértices verdes L_v é preenchida pela função LISTA_VERDES (Algoritmo D.1). A escolha do vértice verde tem o propósito de selecionar o vértice com menor número de arestas incidentes de vértices não-congelados. Esta condição direciona o processo para a construção da menor sequência maximal (veja Teorema 4.1.2).

Algoritmo D.2: SELECIONAR_VERTICE(T)

Entrada: T = matriz de troca.
Saída: v = vértice verde selecionado para mutação.

```
1  v ← 0
2  L_v ← LISTA_VERDES(T)     // Lista com vértices verdes de T (Algoritmo D.1)
3  menorIncidencia ← n
4  enquanto (L_v ≠ ∅) faça
5      i ← vértice verde de L_v
6      incidencia ← 0
7      para (j ← 1...n) faça
8          se (T(i,j) ⩽ −1) então
9              incidencia ← incidencia + 1          // |(v,i)| tal que v ∈ Q_0
10         fim
11     fim
12     se (incidencia < menorIncidencia) então
13         v ← i                                    // Vértice selecionado para mutação
14         menorIncidencia ← incidencia
15     fim
16 fim
17 retorna v
```

A função MUTAÇÃO(T, v) faz a mutação da matriz T em v de acordo com a Definição 4.2. Na linha 5, é calculado no novo t'_{ij} de T quando $i = k$ ou $j = k$. A linha 27 descreve como proceder nos demais casos.

Algoritmo D.3: MUTAÇÃO(T, v)

Entrada: T = matriz de troca, $v \in Q_0$.
Saída: $T_n = \mu_v(T)$.

$T_n \leftarrow T$
para $(i \leftarrow 1 \ldots n)$ **faça**
  para $(j \leftarrow 1 \ldots 2 \cdot n)$ **faça**
    se $(i = v$ ***ou*** $j = v)$ **então**
      $T_n(i,j) \leftarrow -T(i,j)$ // Calculando t'_{ij} (Definição 4.2)
    senão
      se $(T(i,v)) > 0$ **então**
        $[t_{i,v}]_+ \leftarrow T(i,v)$
      senão
        $[t_{i,v}]_+ \leftarrow 0$
      fim
      se $(T(v,j)) > 0$ **então**
        $[t_{v,j}]_+ \leftarrow T(v,j)$
      senão
        $[t_{v,j}]_+ \leftarrow 0$
      fim
      se $(T(i,v)) < 0$ **então**
        $[t_{i,v}]_- \leftarrow T(i,v)$
      senão
        $[t_{i,v}]_- \leftarrow 0$
      fim
      se $(T(v,j)) < 0$ **então**
        $[t_{v,j}]_- \leftarrow T(v,j)$
      senão
        $[t_{v,j}]_- \leftarrow 0$
      fim
      // Calculando t'_{ij} (Definição 4.2)
      $T_n(i,j) \leftarrow T(i,j) + [t_{i,v}]_+ \cdot [t_{v,j}]_+ - [t_{i,v}]_- \cdot [t_{v,j}]_-$
    fim
  fim
fim
retorna T_n

A função ATUALIZA_LISTAS (veja Algoritmo D.4) aciona o Algoritmo D.3 para fazer a mutação da matriz T em direção a μ e atualiza as listas L_{mat}, LA_{mat}, L_{conf} e LA_{conf}. Neste caso, LISTAVERDES(T_n) foi modificado para retornar um vetor de n posições com 1 (verde) ou 0 (vermelho) em cada posição.

A lista LA_{mat} e LA_{conf} são listas auxiliares para a lista L_{mat} e L_{conf}. Estas foram necessárias porque quando o algoritmo faz a terceira mutação μ_j na sequência $\mu_j \cdot \mu_i \cdot \mu_j$ (veja Teorema 4.3.4), a matriz $T(\mu_j(\widetilde{Q}))$ não pode ser colocada em L_{mat} após a matriz $T(\mu_i(\widetilde{Q}))$. Se isto acontecer, a identificação do frame fica comprometida para as próximas matrizes a serem processadas. Portanto, esta matriz é guardada em LA_{mat} e definida uma restrição para μ_j (veja no Algoritmo D.4, linha 8). Quando esta restrição for identificada, a matriz $T(\mu_j(\widetilde{Q}))$ vai para L_{mat} na sequência correta.

Algoritmo D.4: ATUALIZA_LISTAS$(T, \mu, id_T, lista)$

Entrada: T = matriz de troca, μ =vértice para mutação, id_T = identificação de T, $lista$ = lista L_{mat} ou LA_{mat} a ser atualizada.

Saída: T_n = matriz de troca após mutação em μ; id_{Tn} = identificação de T_n; $cores_n$ = cores em T_n; Listas $L_{mat}, L_{conf}, LA_{mat}, LA_{conf}$ e L_{rp} atualizadas.

1 $T_n \leftarrow$ MUTAÇÃO(T, μ) // Algoritmo D.3
2 $cores_n \leftarrow$ LISTA_VERDES(T_n) // Algoritmo D.1
3 $id_{Tn} \leftarrow id_T + 1$ // identificação de T_n
4 **se** $(lista = `L_{mat}')$ **então**
5 $L_{mat}.inserir(T_n)$
6 $L_{conf}.inserir(id_T, id_{Tn}, \mu, cores_n)$
7 **senão**
8 $cores_n(\mu) \leftarrow 1^*$ /* define uma restrição para cor de μ que será usada para colocar T_n em L_{mat} na sequência correta */
9 $LA_{mat}.inserir(T_n)$
10 $LA_{conf}.inserir(id_T, id_{Tn}, \mu, cores_n)$
11 **fim**
12 $L_{rp} \leftarrow$ c-vetores de T_n // inclui os c-vetores positivos de T_n em L_{rp}
13 **retorna** $T_n, id_{Tn}, cores_n, L_{mat}, L_{conf}, LA_{mat}, LA_{conf}, L_{rp}$

A função OBTER_VERTICE (veja Algoritmo D.5) seleciona um vértice verde para mutação. Neste caso, o vértice selecionado não pode gerar um triângulo infinito (veja Teorema 4.3.2) e deve receber uma ou mais arestas de vértices não-congelados (Teorema 4.3.5). A função retornará o vértice para mutação ou, caso não seja selecionado nenhum vértice, retornará 0 (zero).

Algoritmo D.5: OBTER_VERTICE$(T, cores)$

Entrada: T = matriz de troca; $cores$ = cores em T.
Saída: v = vértice de acordo com os Teoremas 4.3.2 e 4.3.5 ou $v = 0$, caso contrário.

```
1  v ← 0
2  para (i ← 1...n) faça
3      se (cores(i) = 1 ou cores(i) = 1*) então
4          aresta ← 0
5          para (j ← 1...n) faça
6              se (T(i, j) ⩽ −2) então
7                  sair /* A mutação em v gera um triângulo infinito
                   (Teorema 4.3.2). Interrompe o laço (linha 4) e analisa o
                   próximo i                                          */
8              fim
9              se (T(i, j) = −1) então
10                 aresta ← 1 /* O vértice i recebe uma ou mais arestas de
                   vértices não-congelados (Teorema 4.3.5)            */
11             fim
12         fim
13         se (aresta) então
14             v ← i
15             retorna v /* Se não sair na linha 6, retorna o vértice i com
                   ou sem a restrição (caso cores(v) = 1 ou cores(v) = 1*)  */
16         fim
17     fim
18 fim
19 retorna v /* Retorna 0. Não existem vértices verdes que estão de acordo
   com os Teoremas 4.3.2 e 4.3.5                                      */
```