ALGORITMOS PARA O PROBLEMA DE *NILCATENATION* COM APLICAÇÃO NA DETECÇÃO DE LAVAGEM DE DINHEIRO EM CRIPTOMOEDAS

CLYNTON AUGUSTO TOMACHESKI AMARAL

ALGORITMOS PARA O PROBLEMA DE *NILCATENATION* COM APLICAÇÃO NA DETECÇÃO DE LAVAGEM DE DINHEIRO EM CRIPTOMOEDAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Sebastián Alberto Urrutia
Coorientadora: Anolan Yamilé Milanés Barrientos

Belo Horizonte

Dezembro de 2020

Amaral, Clynton Augusto Tomacheski

A485a Algoritmos para o Problema de *Nilcatenation* com Aplicação na Detecção de Lavagem de Dinheiro em Criptomoedas [manuscrito] / Clynton Augusto Tomacheski Amaral. — 2020
xii, 58 f. il.

Orientador: Sebastián Alberto Urrutia.
Coorientadora: Anolan Yamilé Milanés Barrientos.
Dissertação (mestrado) — Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 56-58

1. Computação – Teses. 2. Algoritmos em Grafos – Teses. 3. Programação Inteira – Teses. 4. Lavagem de Dinheiro – Teses. 5. Bitcoin – Teses. I. Urrutia, Sebastián Alberto. II. Barrientos, Anolan Yamilé Milanés. III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV. Título.

CDU 519.6*62(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa – CRB 6.ª Região n.º 1510

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Algoritmos para o Problema de Nilcatenation com Aplicação na Detecção de Lavagem de Dinheiro em Criptomoedas

CLYNTON AUGUSTO TOMACHESKI AMARAL

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Sebastian Alberto Urrutia - Orientador
Departamento de Ciência da Computação - UFMG

Profa. Anolan Yamilé Milanés Barrientos - Coorientadora
Departamento de Computação - CEFET-MG

Prof. Thiago Ferreira de Noronha
Departamento de Ciência da Computação - UFMG

Prof. Jeroen Antonius Maria van de Graaf
Departamento de Ciência da Computação - UFMG

Prof. João Fernando Machry Sarubbi
Departamento de Computação - CEFET-MG

Belo Horizonte, 18 de Dezembro de 2020.

Resumo

O saldo de um vértice é definido como a diferença entre a soma dos pesos dos arcos que entram e os arcos que saem do vértice. Desse modo, o Problema de *Nilcatenation* (NCP) consiste em encontrar um subconjunto de arcos que pode ser removido de forma segura sem alterar o saldo de nenhum vértice.

Neste trabalho, o NCP é analisado teoricamente, sendo apresentada uma nova prova de NP-completude e uma primeira formulação de programação linear inteira. Além disso, são introduzidos um conjunto de grafos de teste e mecanismos para o pré-processamento de instâncias do problema. Ademais, é apresentada uma comparação experimental do método de *branch and bound* e do algoritmo de *local branching*, ambos utilizando a formulação proposta. Os resultados experimentais mostram que o algoritmo de *local branching* tem desempenho melhor ou igual na maioria das instâncias de teste.

Seguindo a premissa que o histórico de transações de uma criptomoeda pode ser modelado como um grafo, este trabalho também propõe a aplicação do NCP como um mecanismo para identificar padrões de lavagem de dinheiro em um grafo gerado a partir das transações de uma criptomoeda.

Considerando a crescente importância das criptomoedas no mundo financeiro e que uma de suas principais criticas é sua exploração em atividades ilícitas, este trabalho apresenta um estudo experimental dos algoritmos propostos aplicados à transações extraídas do histórico de transações do Bitcoin, demonstrando a viabilidade da aplicação do NCP neste contexto.

Palavras-chave: Algoritmos em Grafos, Programação Inteira, *Local Branching*, Lavagem de Dinheiro, Criptomoedas.

Abstract

The balance of a vertex is the difference between the sum of weights of arcs entering and arcs leaving the vertex. Thus, the Nilcatenation Problem (NCP) consists of finding a subset of arcs that can be safely removed without modifying the balance of any vertex.

This work analyses theoretically the NCP, presenting a new NP-completeness proof, and a first linear integer programming formulation. Moreover, it introduces a set of benchmark instances and mechanisms for NCP instances preprocessing. Furthermore, the study presents an experimental comparison of the branch and bound method and the local branching algorithm, both of them using the proposed formulation. Experimental results show that the local branching algorithm performs equally or better on the majority of the instances.

Following the premise that it is possible to model a cryptocurrency transaction history as a graph, this work also expands the NCP usage as a method to detect possible money laundering patterns on graphs generated by cryptocurrency transactions.

Considering the cryptocurrencies growing importance in the financial world and one of its main critiques is that cryptocurrencies can be exploited as a tool for criminal activity, this work presents an experimental study of the proposed algorithms applied to transactions extracted from Bitcoin's blockchain, demonstrating the viability of the NCP employment in such context.

Keywords: Graph Algorithms, Linear Programming, Local Branching, Money Laundering, Cryptocurrencies.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Acrônimos

NCP Problema de *Nilcatenation*

PSS Problema de Soma de Subconjuntos

MPSS Problema de Soma de Subconjuntos Multidimensional

PLI Programação Linear Inteira

LLL Algoritmo de Redução de Bases de Lenstra–Lenstra–Lovász

Sumário

Capítulo 1

Introdução

O Problema de *Nilcatenation* (NCP) é um problema em multigrafos direcionados e foi introduzido por Géraud et al. [2017] como um mecanismo para compressão de grafos transacionais. Dado que o saldo de um vértice é a diferença entre os pesos dos arcos de saída e dos arcos de entrada desse vértice, então o NCP consiste em encontrar um conjunto de arcos que pode ser retirado do grafo original sem alterar o saldo de nenhum vértice.

O objetivo principal desta dissertação é inicialmente analisar o NCP sob uma perspectiva de otimização combinatória, introduzindo problemas relacionados, uma formulação de programação linear inteira e um algoritmo de pré-processamento para instâncias do NCP. Para em seguida, discutir a aplicação e comparar o desempenho de algoritmos para resolver o NCP, como o algoritmo baseado em redução de bases de Lenstra–Lenstra–Lovász (LLL), proposto por Géraud et al. [2017], e o algoritmo de *local branching*, cuja a utilização é sugerida por este trabalho.

Ademais, com o intuito de ampliar a aplicação do NCP, também é proposta a sua utilização como uma ferramenta para identificar padrões compatíveis com a lavagem de dinheiro em grafo de transações de criptomoedas. Especificamente para o Bitcoin, o histórico de transações entre usuários é publicamente acessível e existem técnicas para sua modelagem como um grafo [Reid & Harrigan, 2011]. Seguindo essa metodologia, alguns padrões estruturais podem emergir no grafo de transações em possíveis casos de lavagem de dinheiro, como por exemplo, um ciclo formado por uma série de transações entre contas com o mesmo valor.

Neste contexto, considerando um grafo que represente o histórico de sucessivas transações entre usuários, o NCP tem como objetivo encontrar um subgrafo do histórico das transações que pode ser removido sem alterar o saldo final de cada usuário. Desse modo, devido ao impacto negativo da lavagem de dinheiro para a populariza-

ção do uso de criptomoedas como uma alternativa viável aos métodos tradicionais de transações financeiras, este trabalho também analisa o grafo de transações do Bitcoin, apresentando resultados da aplicação do NCP em transações simuladas e na rede de transações reais.

1.1 Justificativa

Existem poucos estudos na literatura sobre experimentação computacional de algoritmos e formulações para o NCP. Apenas o trabalho de Géraud et al. [2017] apresenta um algoritmo baseado no algoritmo de Lenstra–Lenstra–Lovász para resolver o NCP, porém não realiza nenhum tipo de avaliação de desempenho em grafos gerados aleatoriamente ou baseado em redes reais. Sendo assim, é importante expandir a análise teórica sobre o problema e apresentar novos algoritmos para resolvê-lo.

Criptomoedas fornecem uma plataforma para realização de transações seguras e sem limitações de fronteiras. No entanto, existem indícios de que cerca de um quarto dos usuários e metade das transações do Bitcoin estão relacionadas à atividades ilícitas [Foley et al., 2019], consequentemente causando um descorajamento em sua ampla utilização por partes legítimas. Ou seja, é importante que existam mecanismos para mitigar atividades criminosas no âmbito de criptomedas, a fim de manter o interesse legítimo sobre tal tecnologia.

Por fim, são recorrentes na literatura análises sobre os distintos *blockchains* de criptomoedas, que em linhas gerais, são cadeias distribuídas de blocos em uma rede que armazenam a história pública de transações entre usuários. Existem vários estudos onde são aplicadas análises estatísticas e topológicas sobre as transações em criptomoedas [Reid & Harrigan, 2011; Ron & Shamir, 2012; Meiklejohn et al., 2013; Fleder et al., 2015; Maesa et al., 2017]. Sendo assim, é significativo analisar os subproblemas oriundos da história pública de transações, empregando metodologias ainda não totalmente exploradas nesse contexto, como por exemplo técnicas de otimização combinatória.

1.2 Contribuições

Nesta dissertação é introduzida uma formulação de programação linear inteira para o Problema de *Nilcatenation* e uma prova alternativa de NP-completude. Além disso, é avaliada a aplicação proposta por Géraud et al. [2017] do algoritmo Lenstra–Lenstra–Lovász. O trabalho também propõe o uso do algoritmo de *local branching* com o intuito de encontrar melhores soluções.

Como não existem instâncias de teste na literatura para o problema, o trabalho também define um conjunto de grafos de teste com características diversas, permitindo a análise do desempenho dos algoritmos propostos neste trabalho e de futuros algoritmos. Ademais, é apresentado uma avaliação do desempenho de um algoritmo de *brach and bound* utilizando a formulação proposta e do algoritmo de *local branching*, definindo assim um *baseline* comparativo para futuros algoritmos.

Por fim, é proposta uma ampliação da aplicação do NCP para servir de método de detecção de lavagem de dinheiro em criptomoedas, incluindo os resultados da aplicação em transações simuladas e na rede de transações reais do Bitcoin.

1.3 Organização

Neste capítulo foram apresentadas uma breve introdução ao problema, justificativa para seu estudo e o escopo das contribuições deste trabalho. O restante desta dissertação está organizado da seguinte forma:

Capítulo 2 - Problema de *Nilcatenation* em Grafos. São apresentadas as definições necessárias para descrever formalmente o NCP e sua prova de NP-completude. Além disso, é traçado um paralelo com outros problemas encontrados na literatura.

Capítulo 3 - Formulação e Pré-Processamento para o NCP. Nesse capítulo é introduzida a formulação do NCP como um problema de programação linear inteira e é proposto um método para o pré-processamento de instâncias do NCP.

Capítulo 4 - Algoritmo de *Local Branching* para o NCP. A partir da formulação do NCP como um problema de programação linear inteira, é proposto e justificado o uso do algoritmo de *local branching* para solucionar o NCP.

Capítulo 5 - Experimentos Computacionais. Neste capítulo é definido o conjunto de instâncias de teste e a metodologia utilizada para sua geração. Ademais, é analisado a aplicação do algoritmo de Lenstra–Lenstra–Lovász ao NCP e do algoritmo de pré-processamento para as instâncias de teste. Por fim, são comparados os resultados computacionais do método de *branch and bound* utilizando a formulação proposta e do algoritmo de *local branching*.

Capítulo 6 - Aplicação do NCP para Descoberta de Lavagem de Dinheiro em Criptomoedas. São estabelecidas as definições necessárias relaci-

onadas ao Bitcoin e em seguida é apresentado o experimento conduzido para a detecção de ciclos de lavagem de dinheiro dentro das transações existentes na rede do Bitcoin.

Capítulo 7 - Conclusão. Finalmente, são expostas as conclusões finais sobre o texto e os possíveis trabalhos futuros que podem ser conduzidos sobre o problema.

Capítulo 2

Problema de *Nilcatenation* em Grafos

O Problema de *Nilcatenation* (NCP) envolve multigrafos direcionados, que são grafos que permitem múltiplos arcos entre dois vértices. Neste capítulo o NCP é definido formalmente e são introduzidos dois problemas relacionados: o problema de soma de subconjuntos e o problema de fluxo de soma zero. Por fim, é discutida a relação do NCP com componentes fortemente conexas e é apresentada uma prova alternativa de NP-completude, mais simples do que a presente no trabalho de Géraud et al. [2017].

2.1 Descrição do Problema de *Nilcatenation*

Considere um multigrafo ponderado e direcionado $G = (V, E)$ e uma função $w(e)$ que define os pesos de cada arco. É possível definir o saldo $b(v_i)$ de um vértice v_i como:

$$b(v_i) = \sum_{e_j \in E^+(v_i)} w(e_j) - \sum_{e_k \in E^-(v_i)} w(e_k) \tag{2.1}$$

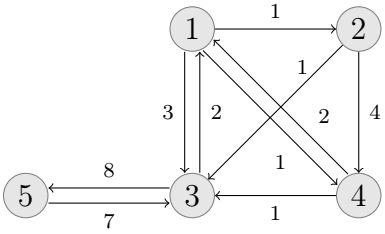
Onde $E^+(v_i)$ representa o conjunto de arcos que chegam no vértice v_i e $E^-(v_i)$ as arcos que saem de v_i. Ou seja, o saldo de cada vértice é definido como a diferença entre o somatório dos pesos dos arcos de entrada e o somatório dos pesos dos arcos de saída. Por sua vez, o saldo completo do grafo G pode ser definido como um vetor $b(G) = \{b(v) \mid v \in V\}$, contendo o saldo de todos os vértices.

Sendo assim, o NCP pode ser definido como um problema de decisão da seguinte forma:

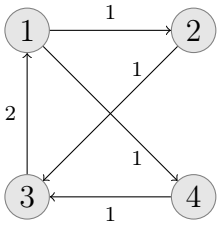
Definição 1 (Versão de Decisão do NCP)**.** *Dado um multigrafo $G = (V, E)$, existe um*

subconjunto de arcos $E' \subseteq E$ de tamanho $k > 0$, tal que $G' = (V, E \backslash E')$ e $b(G) = b(G')$? Com resposta SIM caso exista tal conjunto E' ou NÃO caso contrário.

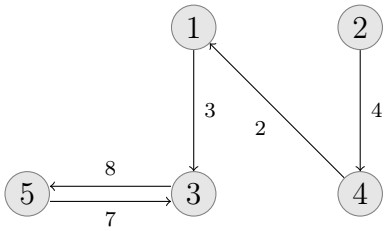
Em outras palavras, se existe um subconjunto de arcos E', denominado de *nilcatenation*, que ao ser removido não altera o valor do saldo de G. A Figura 2.1 apresenta um exemplo do NCP para um grafo direcionado e ponderado.



(a) Grafo G original.



(b) O grafo induzido pelos arcos do *nilcatenation* E' encontrado a partir de G. Cada vértice possui saldo igual a zero.



(c) Grafo G' sem os arcos do *nilcatenation* E'. Os saldos de cada vértice não foram alterados após a retirada do *nilcatenation*.

Figura 2.1: Exemplo do NCP em um grafo direcionado e ponderado.

A versão do NCP como um problema otimização consiste em encontrar o maior inteiro k em que exista um *nilcatenation* para o um grafo $G = (V, E)$. Ou seja, encontrar o conjunto de arcos $E' \subseteq E$ com a maior cardinalidade.

2.2 Problemas Relacionados

Ao estudar um novo problema é importante traçar paralelos e relações com problemas que já foram estudados na literatura. Desse modo, nesta seção são descritos dois problemas que estão correlacionados com o NCP: o problema de fluxo de soma zero e o problema de soma de subconjuntos.

2.2.1 Problema de Fluxo de Soma Zero

Para um grafo direcionado G, um fluxo de soma zero é a atribuição de valores reais não-nulos para os arcos, tal que a soma dos valores dos arcos de entrada é igual a soma dos valores dos arcos de saída. Dessa forma, seja a matriz de incidência $C_{n\times m}$ de G com n vértices e m arcos:

$$C_{n\times m} = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,m} \\ c_{2,1} & c_{2,2} & \dots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \dots & c_{n,m} \end{pmatrix}$$

Onde cada valor $c_{i,j}$ é definido por:

$$c_{i,j} = \begin{cases} 1 & \text{se o arco } j \text{ está entrando no vértice } i \\ -1 & \text{se o arco } j \text{ está saindo do vértice } i \\ 0 & \text{se o arco não é incidente} \end{cases}$$

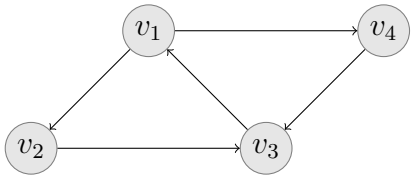
Podemos afirmar que os possíveis fluxos de soma zero para G pertencem ao núcleo da matriz C. Em outras palavras, se o vetor $K = [k_1, k_2, \dots, k_m]^T$ pertence ao núcleo de C, então a atribuição de cada k_j para cada arco j forma um fluxo de soma zero [Akbari et al., 2010]. A Figura 2.2 apresenta um exemplo para um grafo direcionado G.

É possível notar que para um *nilcatenation* E' de G, o conjunto de valores dos pesos de cada arco $e \in E'$ forma um fluxo de soma zero para o grafo induzido pelos arcos em E'. Apesar de possuírem definições similares, esses dois problemas são diferentes, pois o objetivo do problema de fluxo de soma zero é encontrar um conjunto de valores para atribuir aos arcos, enquanto para o NCP esse conjunto de valores já estão definidos e fixos como os pesos de cada arco.

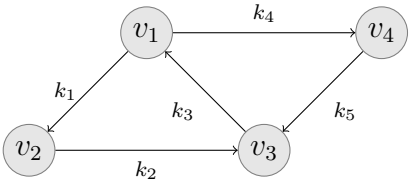
Entretanto, outro paralelo do NCP com o problema de fluxo de soma zero surge ao expandir a definição da matriz de incidência $C'_{n\times m}$ levando em consideração os pesos dos arcos, de forma que cada valor $c'_{i,j}$ dependa do peso w_j correspondente ao arco j:

$$c'_{i,j} = \begin{cases} w_j & \text{se o arco } j \text{ está entrando no vértice } i \\ -w_j & \text{se o arco } j \text{ está saindo do vértice } i \\ 0 & \text{se o arco não é incidente} \end{cases}$$

Portanto, da mesma forma que o núcleo da matriz C é solução para o problema de fluxo de soma zero, os vetores do núcleo da matriz C' serão solução para o NCP se forem formados por coordenadas binárias, representando se um determinado arco será



(a) Grafo G direcionado.



(b) Atribuição dos valores do vetor $K = \{k_1, k_2, k_3, k_4, k_5\}$ aos pesos dos arcos em G.

$$C \cdot K = \begin{pmatrix} -1 & 0 & 1 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{aligned} -k_1 + k_3 - k_4 = 0 &\rightarrow k_3 = k_1 + k_4 \\ k_1 - k_2 = 0 &\rightarrow k_1 = k_2 \\ k_2 - k_3 + k_5 = 0 &\rightarrow k_2 + k_5 = k_3 \\ k_4 - k_5 = 0 &\rightarrow k_4 = k_5 \end{aligned}$$

(c) Se $K \in \ker(C)$, então $C \cdot K = \mathbf{0}$. Dessa forma, para cada vértice, a soma dos valores dos arcos de entrada é igual a soma dos valores dos arcos de saída.

Figura 2.2: Exemplo para o problema de fluxo de soma zero.

parte ou não do *nilcatenation*.

2.2.2 Problema de Soma de Subconjuntos

O Problema de Soma dos Subconjuntos (PSS) é um problema clássico NP-completo de otimização combinatória [Karp, 1972]. Formalmente, o PSS consiste em dado um conjunto de elementos $A \in \mathbb{Z}$, encontrar um subconjunto $S \subseteq A$, tal que a soma de seus elementos seja igual a um inteiro t, ou seja:

$$\sum_{s_i \in S} s_i = t \tag{2.2}$$

Um caso especial desse problema é quando o inteiro t é igual a zero, de forma que $\sum_{s_i \in S} s_i = 0$ e $S \neq \emptyset$, chamado de 0-PSS.

O problema mais geral é o Problema de Soma dos Subconjuntos Multidimensional

(MPSS), onde são considerados n-PSS paralelos, tal que um conjunto solução para um subproblema p permanece como solução para os outros $n-1$ subproblemas. Em outras palavras, dado uma matriz $A \in \mathbb{Z}^{n \times m}$ e um vetor de referencia $t \in \mathbb{Z}^n$, é preciso encontrar um conjunto de vetores S colunas de A, de forma que $\sum_{s_i \in S} s_i = t$, onde $s_i \in \mathbb{Z}^n$.

De maneira similar, é possível definir um caso especial, chamado de 0-MPSS, quando o vetor de referência t é um vetor nulo e $S \neq \emptyset$. A Figura 2.3 contém exemplos simples para ambos os casos especiais 0-PSS e 0-MPSS.

1	4	10	-7	-6	3	2	9

(a) Instância de 0-PSS. Onde $s_1+s_2+s_4+s_8=0$.

3	-2	1	3	5	-8	-3	-11
6	3	1	-6	-8	3	5	2
-5	1	4	6	2	-2	-3	-8

(b) Instância de 0-MPSS. Onde $s_2+s_5+s_5=(0,0,0)^T$.

Figura 2.3: Exemplos para o 0-PSS e o 0-MPSS. Os elementos destacados representam os subconjuntos escolhidos como solução.

Novamente por meio da matriz de incidência $C_{n \times m}$ de um grafo G, é possível destacar a equivalência do NCP com uma instância do MPSS. De fato, o NCP é uma variação notavelmente mais esparsa do MPSS, já que existem apenas dois valores não-nulos por coluna na matriz de incidência do grafo.

2.3 Definições sobre o NCP

2.3.1 Relação com Componentes Fortemente Conexas

Uma componente fortemente conexa de um grafo direcionado G é um conjunto maximal de vértices C, onde para cada par de vértices u e v existem caminhos que conectam os dois vértices em ambas direções, ou seja, todos os vértices em C são alcançáveis por qualquer outro vértice na mesma componente fortemente conexa. A Figura 2.4 mostra um exemplo de um grafo direcionado e seu conjunto de componentes fortemente conexas.

Dado um algoritmo para solucionar o NCP, é possível dividir o grafo original em suas componentes fortemente conexas e executar o algoritmo em cada uma das componentes.

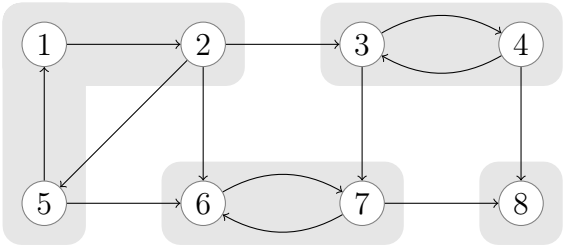


Figura 2.4: Cada região sombreada representa uma componente fortemente conexa do grafo direcionado G. Adaptado de Cormen et al. [2009]

Proposição 1. *A união das soluções de cada componente fortemente conexa será a solução do NCP para o grafo original [Géraud et al., 2017].*

Demonstração. Considere que $v_i, v_j \in V$ pertencem a diferentes componentes conexas de G. Então não é possível coexistirem em G os caminhos p_{ij} (entre v_i e v_j) e p_{ji} (entre v_j e v_i) já que por definição se ambos caminhos p_{ij} e p_{ji} existirem, então os vértices v_i e v_j pertencem à mesma componente fortemente conexa. Dessa forma, o saldo que sai de v_i até v_j não pode retornar até v_i, ou seja, os arcos de um *nilcatenation* que são incidentes a v_i são necessariamente parte da componente fortemente conexa que inclui v_i. Portanto é possível processar cada componente fortemente conexa em separado a fim de encontrar o *nilcatenation* do grafo completo. □

A Proposição 1 tem implicação direta nos algoritmos para solucionar o NCP, pois existem algoritmos lineares para encontrar as componentes fortemente conexas de um grafo, como por exemplo os algoritmos de Tarjan [Tarjan, 1972] ou de Kosaraju-Sharir [Sharir, 1981].

2.3.2 Prova de NP-Completude

A prova a seguir é baseada em Géraud et al. [2017], porém utiliza apenas uma redução polinomial 0-PSS $\leq_P$ NCP, considerando que o 0-PSS é um problema NP-completo [Géraud et al., 2017].

Proposição 2. *NCP pertence a NP.*

Demonstração. Como definido na Seção 2.1, o NCP é um problema de decisão. Dado um grafo $G = (V, E)$, é possível verificar de modo polinomial um certificado E' usando as seguintes condições:

- $E' \neq \emptyset$ e $E' \subseteq E$.

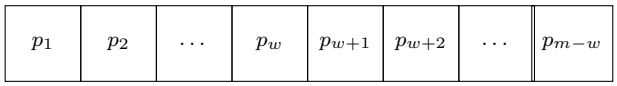
- Considerando um conjunto dos saldos dos vértices $B = \{b_1, b_2, \ldots, b_{|V|}\}$, inicialmente com $B = \{0, 0, \ldots, 0\}$. Então para cada arco $e_k \in E'$, com peso w_k saindo do vértice v_i até o vértice v_j, é possível definir $b_i = b_i - w_k$ e $b_j = b_j + w_k$. Desse modo, se $\forall b \in B$, $b = 0$, então E' é solução.

Claramente, a complexidade assintótica do algoritmo de verificação é $\mathcal{O}(|E|)$ e o tamanho do certificado também é polinomial, já que $|E'| \leq |E|$. Desse modo, o NCP pertence a NP. □

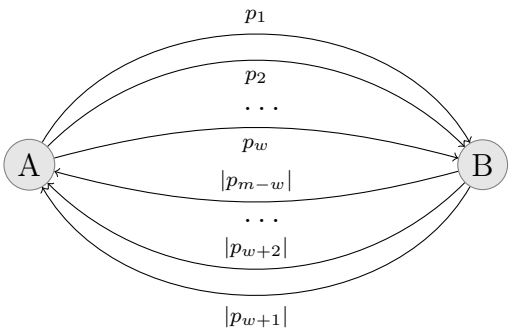
Uma instância genérica do 0-PSS pode ser definida como sendo um conjunto P de m elementos de números inteiros. Dessa forma, é possível gerar uma instância do NCP correspondente a partir da seguinte transformação polinomial:

1. Criar um multigrafo direcionado G com dois vértices A e B;
2. Para cada elemento $p_i \in P$, se $p_i \geq 0$, criar um arco e_i de A até B com o peso $w(e_i) = p_i$, caso $p_i < 0$, criar um arco e_i de B até A com o peso $w(e_i) = |p_i|$.

A Figura 2.5 apresenta a construção de uma instância do NCP a partir de uma instância genérica do 0-PPS.



(a) Instância genérica do 0-PPS com w elementos positivos e $m - w$ negativos.



(b) A mesma instância de (a) após a transformação polinomial para uma instância do NCP.

Figura 2.5: Redução em tempo polinomial de uma instância genérica de 0-PSS para uma instância de NCP.

Proposição 3. *A solução para a instância original do 0-PSS é* ***sim****, se e somente se, a solução para a instância gerada de NCP também é* ***sim****.*

Demonstração. Se existe um *nilcatenation* E' para um grafo construído G e algum inteiro $k = (0, m]$, de modo que $|E'| = k$, é possível afirmar que cada arco de

$\{e'_1, e'_2, \ldots, e'_k\} \in E'$ corresponde a um elemento do conjunto P, definindo assim um conjunto $P' \subseteq P$, onde $P' = \{p'_1, p'_2, \ldots, p'_k\}$. Por definição o saldo do vértice A no grafo induzido pelos arcos de $|E'|$ é igual a zero, e todos os arcos pertencentes a E' são incidentes a A, portanto o saldo $b(A) = \sum_{p'_i \in P'} p'_i = 0$. Ou seja, o conjunto P' é solução para a instância do 0-PSS.

O mesmo raciocínio pode ser aplicado inversamente, se $P' = \{p'_1, p'_2, \ldots, p'_k\}$ é solução para a instância de 0-PSS, então existe um conjunto de arcos correspondente E' que é obrigatoriamente solução para a instância de NCP, já que mostrado anteriormente $b(A) = 0$ e por definição $b(B) = -\sum_{p'_i \in P'} p'_i = 0$.

□

Capítulo 3

Formulação e Pré-Processamento para o NCP

Modelos de programação linear inteira (PLI) são úteis para definir a estrutura de um problema de otimização combinatória e são aplicados em uma variada gama de problemas. Existem algoritmos consolidados que usam modelos programação inteira para encontrar soluções exatas com garantias de otimalidade, como por exemplo o método de *branch and bound*.

Neste capítulo é apresentada uma primeira formulação matemática para o NCP baseada em um modelo de programação inteira com o objetivo de maximizar a cardinalidade do conjunto de arcos do *nilcatenation* encontrado. O modelo final apresentado contém apenas variáveis de decisão binárias, em outras palavras, o NCP foi modelado como um problema de otimização inteira binária.

Por fim, é proposto um método de pré-processamento de instâncias a fim de retirar vértices e arcos que nunca serão parte de um *nilcatenation*, consequentemente simplificando o modelo gerado e diminuindo o tamanho da entrada.

3.1 Modelo de Programação Linear Inteira

A fim de definir o modelo matemático, podemos definir a matriz de incidência $C_{n \times m}$ de um grafo $G = (V, E)$, onde $n = |V|$, $m = |E|$ e cada valor $c_{i,j}$ depende do peso w_j correspondente ao arco j, da seguinte forma:

$$c_{i,j} = \begin{cases} w_j & \text{se o arco } j \text{ está entrando no vértice } i \\ -w_j & \text{se o arco } j \text{ está saindo do vértice } i \\ 0 & \text{se o arco não é incidente} \end{cases}$$

Para controlar quais arcos farão parte da solução final E', podemos definir um vetor coluna X com m variáveis de decisão binárias, onde se $x_j = 1 \rightarrow e_j \in E'$:

$$X^T = \begin{pmatrix} x_1 & x_2 & \dots & x_m \end{pmatrix}$$

Dessa forma, a formulação de programação linear inteira do NCP é dada por:

$$\max \sum_{j=1}^{m} x_j, \tag{3.1}$$

sujeito a:

$$\sum_{j=1}^{m} x_j \cdot c_{i,j} = 0, \text{ para } i = 1, 2, \dots, n \tag{3.2}$$

$$x_j \in \{0, 1\} \tag{3.3}$$

A função objetivo 3.1 maximiza o número de arcos do *nilcatenation*. A restrição 3.2 garante o equilíbrio do saldo em cada vértice (definido pela equação 3.4) é respeitado, ou seja, dado um vértice v_i o somatório dos pesos dos arcos (pertencentes à solução) de entrada e saída são iguais. Por fim, a restrição 3.3 define que todas as variáveis de decisão são binárias.

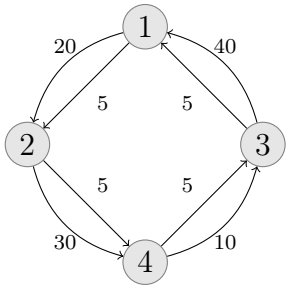
$$\sum_{j \in E'^{+}(v_i)} w_j = \sum_{k \in E'^{-}(v_i)} w_k \tag{3.4}$$

Uma pequena melhoria na formulação é impor que a solução vazia $\{x = 0 \mid x \in X\}$ não seja viável. Considerando que o menor *nilcatenation* em um grafo sem laços é um subgrafo com dois arcos, com pesos iguais e em direções opostas, entre dois vértices distintos, é possível utilizar a restrição adicional:

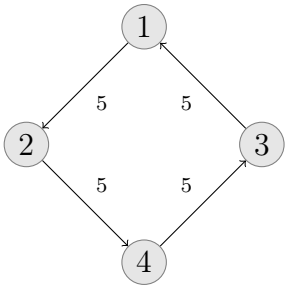
$$\sum_{j=1}^{m} x_j \geq 2 \tag{3.5}$$

A Figura 3.1 apresenta um exemplo simples contendo o grafo G original, seu respectivo *nilcatenation* e a visualização dos arcos escolhidos como solução na matriz

de incidência do grafo.



(a) Grafo G original.



(b) O grafo induzido pelos arcos do *nilcatenation*.

$$C = \begin{pmatrix} & x_2 & & x_4 & & x_6 & & x_8 \\ -20 & -5 & 0 & 0 & 0 & 0 & 40 & 5 \\ 20 & 5 & -30 & -5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 5 & -40 & -5 \\ 0 & 0 & 30 & 5 & -10 & -5 & 0 & 0 \end{pmatrix}$$

(c) A matriz C e as variáveis x_2, x_4, x_6 e x_8 como solução do problema.

Figura 3.1: A partir da matriz de incidência do grafo G é possível encontrar o conjunto de arcos que formam o *nilcatenation*. Note que a solução $X^T = [0, 1, 0, 1, 0, 1, 0, 1]$, que representa os arcos, satisfaz a restrição $C \cdot X = [0, 0, 0, 0]^T$.

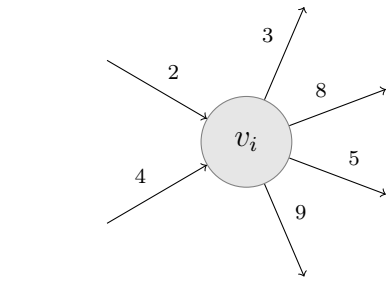
3.2 Pré-Processamento de Instâncias para o NCP

De acordo com a Proposição 1, é possível dividir um grafo em suas componentes fortemente conexas e executar qualquer algoritmo para o NCP em cada componente separadamente. No entanto, para grafos que possuem apenas uma componente fortemente conexa, esse método não pode ser aplicado. Sendo assim, esta seção apresenta um método para o pré-processamento de instâncias do NCP, com o objetivo de retirar vértices e/ou arcos que nunca serão parte da solução, simplificando o modelo gerado.

Dado um grafo $G = (V, E)$, podemos definir duas estratégias de pré-processamento:

vértices Para um vértice v ser parte do grafo induzido pelos arcos de um *nilcatenation*, o conjunto de pesos dos arcos incidentes a v devem respeitar a restrição 3.4 de

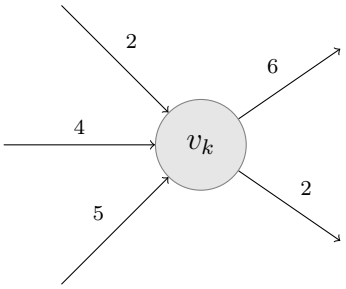
conservação de saldo. Sendo assim, é possível criar uma instância de 0-PSS, onde para cada arco j é criado um elemento com valor w_j caso o arco esteja chegando em v ou $-w_j$ caso o arco esteja saindo de v. Dessa forma, se não existir solução para o 0-PSS associado, o vértice v e consequentemente seus arcos incidentes podem ser removidos de maneira segura do grafo. A Figura 3.2 apresenta exemplos para ambos os casos.



4	2	-3	-8	-5	-9

$t = 0$

(a) O vértice v_i pode ser removido, pois não existe solução para o 0-PSS associado.



5	4	2	-6	-2

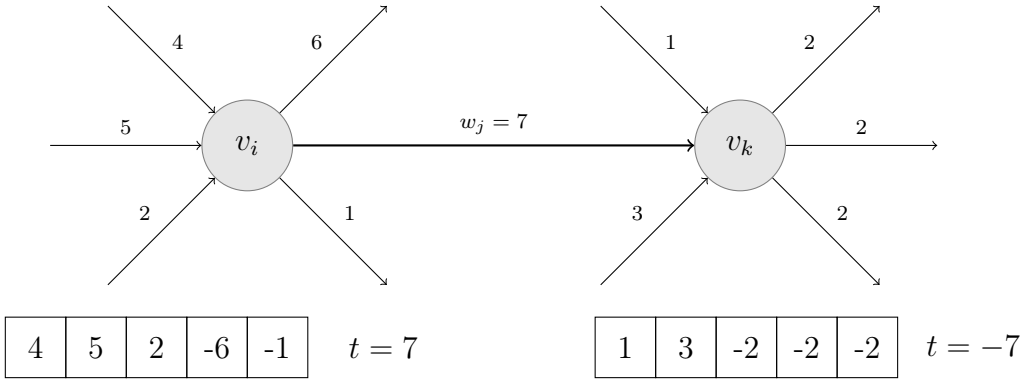
$t = 0$

(b) O vértice v_k não pode ser removido, pois existem duas soluções $\{4, 2, -6\}$ e $\{2, -2\}$ para o 0-PSS associado.

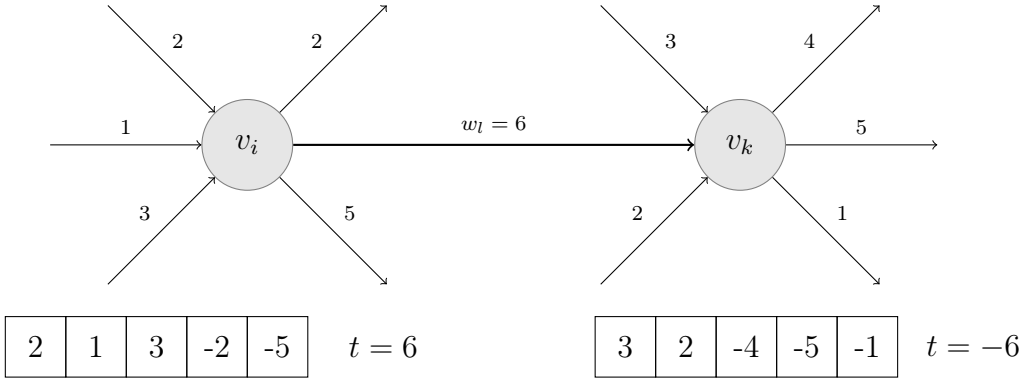
Figura 3.2: Exemplo do pré-processamento de vértices.

arcos Um arco e_j de peso w_j que conecta o vértice v_i ao vértice v_k só será parte de um *nilcatenation* se: (i) é possível encontrar dois subconjuntos $S_i^+ \subseteq E^+(v_i)$ e $S_i^- \subseteq E^-(v_i) \setminus \{e_j\}$ tal que a diferença do somatório dos pesos de S_i^+ e S_i^- seja igual a w_j; (ii) é possível encontrar dois subconjuntos $S_k^+ \subseteq E^+(v_k) \setminus \{e_j\}$ e $S_k^- \subseteq E^-(v_k)$ tal que a diferença do somatório dos pesos de S_k^+ e S_k^- seja igual a $-w_j$. Ou seja, existem duas instâncias de PSS associadas. Caso não existam soluções para ambas instâncias, o arco e_j pode ser removido de maneira segura do grafo. A Figura 3.3 apresenta exemplos para ambos os casos.

O método proposto utiliza ambas estratégias de forma iterativa a fim de remover vértices e arcos de um grafo. Como base do algoritmo de pré-processamento é utilizado um algoritmo de programação dinâmica para resolver cada instância do PSS. Dado um conjunto de pesos $S = \{s_1, s_2, \ldots, s_n\}$, a complexidade assintótica do algoritmo é pseudo-polinomial e limitada em $\mathcal{O}(\sum_{i=1}^{n} |s_i| \cdot n)$, ou seja, além do tamanho da entrada é limitada pela soma dos valores absolutos dos pesos.



(a) O arco w_j pode ser removido, pois não existe solução para o PSS com $A = \{1, 3, -2, -2, -2\}$ e $t = -7$.



(b) O arco w_l não pode ser removido, pois ambos os PSS associados possuem solução.

Figura 3.3: Exemplo do pré-processamento de arcos.

O Algoritmo 1 apresenta o pseudo código para um algoritmo de programação dinâmica para o PSS. Na linha 4, a sub-rotina `progDinamicaPSS` define uma matriz `memo` de *memoization* de duas dimensões para armazenar os índices dos elementos e as possíveis somas. A matriz `memo` é inicializada com valores boleanos falsos, significando que não existe solução para determinado índice e soma.

Na linha 5 é executado um laço definindo para cada valor de s_i que $\texttt{memo}[i][s_i]$ possui solução, já que sempre existirá uma solução para uma soma s_i, escolhendo apenas o item de índice i. Por fim, o laço na linha 12 define a relação de recorrência geral para um índice i e uma soma t como sendo $\texttt{memo}[i][t] \leftarrow \texttt{memo}[i-1][t] \vee \texttt{memo}[i-1][t-s_{i-1}]$.

Algoritmo 1: Algoritmo de Programação Dinâmica para o PSS

```
1  progDinamicaPSS (S = {s_1, s_2, ..., s_n}, t)
2      limInferior ← 0;
3      limSuperior ← 0;
4      memo ← matriz de memoization;
5      para cada s_i ∈ S faça
6          memo[i][s_i] ← Verdadeiro;
7          limInferior ← min(limInferior, limInferior + s_i);
8          limSuperior ← max(limSuperior, limSuperior + s_i);
9      fim
10     para cada s_i ∈ S faça
11         para soma ← limInferior até limSuperior faça
12             memo[i][soma] ← memo[i − 1][soma] ∨ memo[i − 1][soma − s_{i−1}];
13         fim
14     fim
15     retorna memo[n][t];
```

Finalmente, é possível definir o algoritmo de pré-processamento para um grafo de entrada $G = (V, E)$. O método consiste em utilizar ambas estratégias de remoção de vértices e arcos da seguinte maneira iterativa:

1. Para cada vértice $v_i \in V$, criar uma instância de 0-PSS associada. Caso não exista solução para o 0-PSS, remover o vértice v_i e os arcos incidentes a v_i do grafo.

2. Executar o Passo 1 até mais nenhum vértice poder ser removido do grafo.

3. Para cada arco $e_j \in E$, criar duas instâncias de PSS associadas. Se não existirem soluções para ambas as instâncias, remover o arco e retornar ao Passo 1.

4. Retornar o novo grafo.

O Algoritmo 2 apresenta o pseudo código do método de pré-processamento. Note que na linha 8 é executado a estratégia de remoção de vértices e na linha 19 a estratégia de remoção de arcos.

Com o intuito de definir a complexidade assintótica do algoritmo de pré-processamento do NPC, é importante ressaltar que a cada iteração do algoritmo ao menos um arco é retirado do grafo. Portanto, a sub-rotina `progDinamicaPSS` é executada no máximo $|E| \cdot (|V| + 2|E|)$ vezes. O pior caso para uma instância de PSS associada ao grafo é dado por $\mathcal{O}(w_{\max} \cdot \delta_{\max})$, onde $w_{\max}$ é o maior somatório dos

Algoritmo 2: Algoritmo de Pré-Processamento para o NCP

preProcessamentoNCP $(G = (V, E))$
  $continuarProcessamento \leftarrow$ Verdadeiro;
  enquanto $continuarProcessamento$ **faça**
    $continuarProcessamento \leftarrow$ Falso;
    $processarVertices \leftarrow$ Verdadeiro;
    enquanto $processarVertices$ **faça**
      $processarVertices \leftarrow$ Falso;
      para cada $v_i \in V$ **faça**
        $A_{\text{pos}} \leftarrow \{w_j \mid e_j \in E^+(v_i)\}$;
        $A_{\text{neg}} \leftarrow \{-w_j \mid e_j \in E^-(v_i)\}$;
        $possuiSol \leftarrow$ `progDinamicaPSS(`$A_{pos} \cup A_{neg}$`, 0)`;
        se $\neg possuiSol$ **então**
          $processarVertices \leftarrow$ Verdadeiro;
          $V \leftarrow V \setminus \{v_i\}$;
          $E \leftarrow E \setminus E(v_i)$;
        fim
      fim
    fim
    para cada $e_j \in E$ **faça**
      $v_i \leftarrow$ `verticeEntrada(`e_j`)`;
      $A_i^{\text{pos}} \leftarrow \{w_i \mid e_i \in E^+(v_i)\}$;
      $A_i^{\text{neg}} \leftarrow \{-w_i \mid e_i \in E^-(v_i) \setminus \{e_j\}\}$;
      $possuiSolEntrada \leftarrow$ `progDinamicaPSS(`$A_i^{pos} \cup A_i^{neg}$`, `w_j`)`;
      $v_k \leftarrow$ `verticeEntrada(`e_j`)`;
      $A_k^{\text{pos}} \leftarrow \{w_k \mid e_k \in E^+(v_k) \setminus \{e_j\}\}$;
      $A_k^{\text{neg}} \leftarrow \{-w_k \mid e_k \in E^-(v_k)\}$;
      $possuiSolSaida \leftarrow$ `progDinamicaPSS(`$A_k^{pos} \cup A_k^{neg}$`, `$-w_j$`)`;
      se $\neg(possuiSolEntrada \wedge possuiSolSaida)$ **então**
        $continuarProcessamento \leftarrow$ Verdadeiro;
        $E \leftarrow E \setminus \{e_j\}$;
        terminar laço;
      fim
    fim
  fim
  retorna G;

pesos dos arcos incidentes a um vértice $v \in V$ e $\delta_{\max}$ é o maior grau de um vértice em G. Portanto, é possível afirmar que a complexidade assintótica do algoritmo de pré-processamento é limitada por $\mathcal{O}(w_{\max} \cdot \delta_{\max} \cdot (|E| \cdot |V| + |E|^2))$, ou seja, o algoritmo de pré-processamento possui complexidade pseudo-polinomial.

Capítulo 4

Algoritmo de *Local Branching* para o NCP

Este capítulo apresenta o algoritmo de *local branching*, cujo o uso para resolver o NCP é proposto por este trabalho. Além disso, são apresentados os detalhes de implementação do método aplicado ao NCP.

Na literatura, existe apenas o algoritmo proposto por Géraud et al. [2017] para resolver o NCP, que utiliza o algoritmo LLL [Lenstra et al., 1982] como mecanismo para resolver o problema de programação inteira correspondente. No entanto, o algoritmo de LLL possui limitações em uma aplicação prática para o NCP, pois normalmente gera soluções com poucos arcos. Essa hipótese é validada por meio de um experimento conduzido na Seção 5.5, onde o LLL é executado para um conjunto de instâncias do NCP.

Encontrar uma solução viável que forme um *nilcatenation* de qualquer tamanho é um problema NP-completo. Dessa forma, não existem heurísticas construtivas em tempo polinomial para o NCP. De fato, construir soluções viáveis para um grafo genérico envolve, para cada arco escolhido como parte de uma solução, resolver dois problemas de PSS associados.

Sendo assim, este trabalho propõe o uso do algoritmo de *local branching* como alternativa para explorar de modo eficiente o espaço de soluções mantendo a viabilidade das soluções encontradas. Como o algoritmo de *local branching* se baseia em uma busca em árvore e existe um limite de tempo para sua execução, o método tem comportamento heurístico. Com efeito, para um tempo de execução limite infinito, o algoritmo de *local branching* é exato.

4.1 Algoritmo de *Local Branching*

Proposto por Fischetti & Lodi [2003], o algoritmo de *local branching* permite o uso de resolvedores para problemas de programação linear inteira mista como um mecanismo caixa-preta para explorar, de forma controlada, subespaços de solução. Por exemplo, para o NCP é possível empregar qualquer resolvedor PLI que utilize a formulação apresentada no Capítulo 3.

Originalmente o algoritmo *local branching* é genérico e pode ser aplicado em problemas com variáveis binárias, inteiras e contínuas. No entanto, todas as variáveis de decisão para o NCP são binárias. Desse modo, com o intuito de simplificar as definições, o algoritmo será descrito a seguir considerando apenas variáveis binárias, representadas pelo conjunto $\mathcal{B}$.

Dado um valor inteiro k e uma solução de referência $\bar{x}$, seja $S = \{\bar{x}_j = 1 \mid j \in \mathcal{B}\}$ o conjunto de variáveis de decisão que pertencem a solução, é possível definir uma vizinhança k-OPT de $\bar{x}$ que satisfaça a seguinte restrição local:

$$\Delta(x, \bar{x}) = \sum_{j \in S}(1 - x_j) + \sum_{j \in \mathcal{B} \setminus S} x_j \leq k \tag{4.1}$$

O lado esquerdo da restrição 4.1 conta o número de variáveis que trocaram de valor entre 0 e 1 em relação à solução referência $\bar{x}$, definindo assim o tamanho da vizinhança em um processo de busca local. Essa restrição pode ser utilizada como um esquema de ramificação, particionando o espaço de solução com duas restrições:

$$\Delta(x, \bar{x}) \leq k \text{ (ramificação à esquerda) } \textbf{ou } \Delta(x, \bar{x}) \geq k + 1 \text{ (ramificação à direita)} \tag{4.2}$$

De modo geral, o algoritmo consiste em aplicar o esquema de ramificação 4.2 para explorar o espaço de soluções do problema original. O algoritmo tem como parâmetros o limite de tempo total, o limite de tempo de exploração de um único nó e o valor inicial para k.

A partir de uma solução inicial de referência $\bar{x}$, que pode ser obtida por um resolvedor PLI (que retorna a primeira solução viável encontrada), é criada uma árvore de busca contendo apenas o nó raiz com uma restrição inicial $\Delta(x, \bar{x}) \leq \infty$, ou seja, o nó raiz não possui nenhuma restrição local adicional. Note que é possível que a solução inicial seja ótima ou o modelo seja inviável, dessa forma a solução é retornada e não é expandida a árvore de busca.

Em seguida de forma iterativa, é adicionada a restrição de ramificação à esquerda

$\Delta(x, \bar{x}) \leq k$ e o problema é resolvido com essa nova restrição por meio de um resolvedor PLI buscando soluções melhores que o limite inferior inicialmente dado por $c^T\bar{x}$ (o valor da função objetivo da solução $\bar{x}$). Na próxima iteração, a árvore de busca será explorada de acordo com a solução x' encontrada pelo resolvedor PLI:

Solução ótima Foi encontrada uma solução ótima x' para a vizinhança atual. Dessa forma, a última restrição de ramificação à esquerda é substituída pela restrição de ramificação à direita $\Delta(x, \bar{x}) \geq k + 1$ e a solução de referência $\bar{x}$ é atualizada por x' e também é atualizado o valor de limite inferior.

Modelo inviável É garantido que a vizinhança atual não possui uma solução viável que tenha valor de função objetivo melhor que o limite inferior. Dessa forma, a última restrição de ramificação à esquerda é substituída pela restrição de ramificação à direita $\Delta(x, \bar{x}) \geq k+1$ e a vizinhança é ampliada em $\lceil k/2 \rceil$. Se esse caso também ocorreu na iteração anterior, então é realizado uma diversificação completa, onde a próxima execução do resolvedor PLI busca a primeira solução independente do limite inferior, com um tempo de execução limitado apenas pelo tempo total restante.

Solução viável Uma solução foi encontrada para a vizinhança atual, atualizando a solução de referência $\bar{x}$ por x' e se for melhorante, também atualizando o limite inferior. Porém a solução não é garantidamente ótima dentro do tempo limite. Nesse caso, não é possível substituir a restrição de ramificação à esquerda pela restrição de ramificação à direita $\Delta(x, \bar{x}) \geq k + 1$, porque é possível que a solução ótima para o problema inicial ainda pertença a vizinhança atual. Sendo assim, é apenas excluída a última restrição à esquerda $\Delta(x, \bar{x}) \leq k$ e se esse caso também ocorreu na iteração anterior, é adicionada a restrição tabu $\Delta(x, \bar{x}) \geq 1$. Note que para um problema de programação inteira com variáveis inteiras e/ou contínuas é necessário realizar o refinamento da solução antes de adicionar a restrição tabu, ou seja, fixar as variáveis binárias a fim de obter um certificado de otimalidade. Contudo, como o NCP possui apenas variáveis binárias, esse passo não é necessário.

Não encontrou solução viável Não foi encontrada solução com custo melhor que o limite inferior dentro do limite de tempo especificado para a exploração de um nó. Portanto, é executado somente um dos passos a seguir:

- Se na iteração anterior uma diversificação já tenha ocorrido ou se também não foi encontrada uma solução viável, a última restrição local de rami-

ficação à esquerda é excluída e em seguida é realizado o processo de diversificação completo, ampliando a vizinhança em $\lceil k/2 \rceil$. Dessa forma, a próxima iteração o resolvedor PLI busca a primeira solução independente do limite inferior, com um tempo de execução limitado apenas pelo tempo total restante.

- Ou caso contrário, é adicionada a restrição tabu $\Delta(x, \bar{x}) \geq 1$ e a vizinhança é diminuída em $\lceil k/2 \rceil$.

O processo é iterado até que seja encontrado uma solução ótima ou o certificado de inviabilidade para o problema original. O algoritmo é limitado pelo parâmetro de tempo de execução e o número máximo de diversificações que pode ser definido como um parâmetro adicional. Ao final das iterações, caso o tempo limite não tenha sido esgotado, é executado uma última vez o resolvedor PLI para tentar encontrar um certificado de otimalidade para o problema, explorando assim o último nó à direita da árvore de busca criado. O Algoritmo 3 descreve o pseudo código do algoritmo de *local branching*, detalhando atualização dos valores de melhor solução incumbente e limite inferior, além das variáveis necessárias para a execução do processo de diversificação.

4.2 Aplicação do Algoritmo de *Local Branching* ao NCP

Para aplicar o algoritmo de *local branching* para o NCP é necessário apenas utilizar a formulação apresentada no Capítulo 3 em um resolvedor PLI caixa-preta. Especificamente no contexto deste trabalho, foi escolhido utilizar o método de *branch and bound*, que realiza a enumeração das soluções candidatas por meio de uma árvore de busca, de forma que uma solução candidata é ramificada ou descartada de acordo com valores de limite inferior e superior para a função objetivo. À nível de implementação foi utilizado o algoritmo de *branch and bound* do pacote de otimização `ILOG CPLEX`.

Dado um grafo de entrada $G = (V, E)$, é necessário definir os seguintes parâmetros para o algoritmo de *local branching*:

$t_{\texttt{limite}}$ Tempo limite total (em segundos) de execução do algoritmo.

$t_{\texttt{nó}}$ Tempo limite (em segundos) de exploração de um nó da árvore de busca, implementado da seguinte forma:

$$t_{\texttt{nó}} \leftarrow \texttt{min}(\tfrac{|V|}{|E|} \cdot t_{\texttt{limite}},\ 60s)$$

Algoritmo 3: Algoritmo de *Local Branching* [Fischetti & Lodi, 2003]

LocalBranching $(k, tempoTotalLim, tempoNoLim, maxDiver)$
  $rhs \leftarrow melhorLB \leftarrow LB \leftarrow tempoLim \leftarrow \infty; x^* \leftarrow otm \leftarrow indefinido$;
  $primeiraSol \leftarrow Verdadeiro$; $diversificar \leftarrow Falso$; $contDiver \leftarrow tempoExec \leftarrow 0$;
  repita
    se $rhs < \infty$ **então**
      Adicionar restrição local à esquerda $\Delta(x, \bar{x}) \leq rhs$;
    fim
    $tempoLim \leftarrow$ `min(`*tempoLim, tempoTotalLim - tempoExec*`)`;
    $x', otm \leftarrow$ `PLI(`$tempoLim, LB, primeiraSol$`)`;
    $tempoLim \leftarrow tempoNoLim$;
    se *otm = ótimo encontrado* **então**
      se $c^T x' > melhorLB$ **então**
        $melhorLB \leftarrow c^T x'; x^* \leftarrow x'$;
      fim
      se $rhs \geq \infty$ **então**
        retorna x^* *e otm*;
      fim
      Excluir última restrição local para $\Delta(x, \bar{x}) \geq rhs + 1$;
      $primeiraSol \leftarrow diversificar = Falso$;
      $\bar{x} \leftarrow x'$; $LB \leftarrow c^T x'$; $rhs \leftarrow k$;
    fim
    se $otm \leftarrow$ *inviável* **então**
      se $rhs \geq \infty$ **então**
        retorna x^* *e otm*;
      fim
      Trocar última restrição local para $\Delta(x, \bar{x}) \geq rhs + 1$;
      se $diversificar$ **então**
        $tempoLim \leftarrow \infty; LB = 0; contDiver \leftarrow contDiver + 1$;
        $primeiraSol \leftarrow Verdadeiro$;
      fim
      $rhs \leftarrow rhs + \lceil k/2 \rceil$; $diversificar \leftarrow Verdadeiro$;
    fim
    se $otm \leftarrow$ *encontrou solução viável* **então**
      se $rhs < \infty$ **então**
        se $retornarPrimeiraSol$ **então**
          Excluir última restrição local $\Delta(x, \bar{x}) \leq rhs$;
        fim
        senão
          Trocar última restrição local $\Delta(x, \bar{x}) \leq rhs$ por $\Delta(x, \bar{x}) \geq 1$;
        fim
      fim
      se $c^T x' > melhorLB$ **então**
        $melhorLB \leftarrow c^T x'; x^* \leftarrow x'$;
      fim
      $primeiraSol \leftarrow diversificar = Falso$;
      $\bar{x} \leftarrow x'$; $LB \leftarrow c^T x'$; $rhs \leftarrow k$;
    fim
    se $otm \leftarrow$ *não encontrou solução viável* **então**
      se $diversificar$ **então**
        Excluir última restrição local $\Delta(x, \bar{x}) \leq rhs$;
        $rhs \leftarrow rhs + \lceil k/2 \rceil$; $tempoLim \leftarrow \infty; LB = 0$;
        $contDiver \leftarrow contDiver + 1$; $primeiraSol \leftarrow Verdadeiro$;
      fim
      senão
        Trocar última restrição local $\Delta(x, \bar{x}) \leq rhs$ por $\Delta(x, \bar{x}) \geq 1$;
        $rhs \leftarrow rhs - \lceil k/2 \rceil$;
      fim
      $diversificar \leftarrow Verdadeiro$;
    fim
  até $tempoExec > tempoTotalLim$ *ou* $contDiver > maxDiver$;
  $x^*, otm \leftarrow$ `PLI(`$tempoTotalLim - tempoExec, melhorLB, Falso$`)`;
  retorna x^* *e otm*;

k Valor inicial para definir a vizinhança k-OPT. Neste trabalho foi utilizado $k = 20$, valor considerado eficiente para a maioria dos problemas de programação inteira [Fischetti & Lodi, 2003].

$d_{\texttt{max}}$ Número máximo de diversificações permitidas. Definido como: $d_{\texttt{max}} = |V|$.

Capítulo 5

Experimentos Computacionais

Neste capítulo são apresentados os experimentos computacionais relacionados ao estudo do NCP sobre uma ótica de otimização combinatória. Para este propósito, foram avaliados os seguintes algoritmos:

1. **B&B:** Método de *branch and bound* utilizando a formulação de programação linear inteira apresentada na Seção 3.1. Para sua implementação foi utilizada a API de C++ do pacote de otimização `ILOG CPLEX`.

2. **B&B + LocB:** A mesma implementação do método de *branch and bound*, porém com a opção da heurística de *local branching*[1] ativada no `ILOG CPLEX`.

3. **LocB:** Algoritmo de *local branching* descrito na Seção 4.1, implementado em C++ pelo autor.

4. **NCP-LLL:** Algoritmo de Lenstra–Lenstra–Lovász aplicado ao NCP, proposto por Géraud et al. [2017], implementado em C++ pelo autor, utilizando a biblioteca `fplll` [FPLLL development team, 2016].

Na Seção 5.1 é descrito o processo de geração de instâncias de teste para a execução dos algoritmos e é definido um conjunto de grafos de teste. Por fim, com o objetivo de analisar o comportamento dos algoritmos apresentados neste trabalho, foram conduzidos os seguintes experimentos:

- A execução do algoritmo de pré-processamento para cada uma das instâncias de teste a fim de conhecer seu comportamento, cujos os resultados são analisados na Seção 5.2.

[1] `https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.7.1/ilog.odms.cplex.help/CPLEX/Parameters/topics/LBHeur.html`

- Na Seção 5.3 é apresentada a comparação do desempenho da execução dos algoritmos `B&B`, `B&B + LocB` e `LocB` para as instâncias de teste.
- Por fim, na Seção 5.5 é realizada uma análise da aplicação do `NCP-LLL` em comparação ao `B&B`, com o intuito de validar a hipótese que normalmente o `NCP-LLL` gera soluções com poucos arcos.

Os experimentos foram executados em uma máquina equipada com um processador Intel Core i7-9750H (6 núcleos e 12 *threads*), operando com um *clock* de 2.6GHz (4.5GHz em turbo) e 16GB de memória RAM. Todos os códigos foram escritos em C++ e compilados com o compilador `clang` versão 12.0.0, por fim, o resolvedor PLI utilizado foi o `ILOG CPLEX` versão 12.10. Todas as instâncias de teste e os códigos fonte dos algoritmos implementados estão disponíveis em um repositório público[2].

5.1 Geração de Instâncias

Não existem instâncias na literatura para avaliar o desempenho de algoritmos para o NCP. Sendo assim, foi gerado de forma aleatória um conjunto de grafos de teste com o intuito de comparar a formulação e algoritmos propostos neste trabalho. Devido à Proposição 1 apresentada na Seção 2.1, todos os grafos aleatórios possuem apenas uma única componente fortemente conexa, pois em caso contrário, seria mais eficiente dividir o grafo em suas componentes antes de executar qualquer algoritmo. Para garantir que a instância possui apenas uma componente fortemente conexa foi utilizado o algoritmo de Kosaraju-Sharir. Os parâmetros utilizados para a geração dos grafos foram:

- O número de vértices $|V|$ do grafo.
- A probabilidade p_e de existir um arco entre dois vértices quaisquer, tal que $|E| \approx p_e \cdot |V| \cdot |V - 1|$.
- O tamanho do conjunto de pesos $|W|$ que serão atribuídos a cada arco.
- O valor limite para o peso de um arco $w_{\texttt{lim}}$, tal que para todo $w_j \in W$, $1 \leq w_j \leq w_{\texttt{lim}}$.

Como o problema de NCP trabalha com multigrafos foi escolhida uma probabilidade fixa de 0.05 de existir múltiplos (dois ou três) arcos entre dois vértices. Na Figura 5.1 é exemplificada a relação dos parâmetros com o nome dado à uma instância.

[2] `https://github.com/ctomacheski/ufmg-ncp`

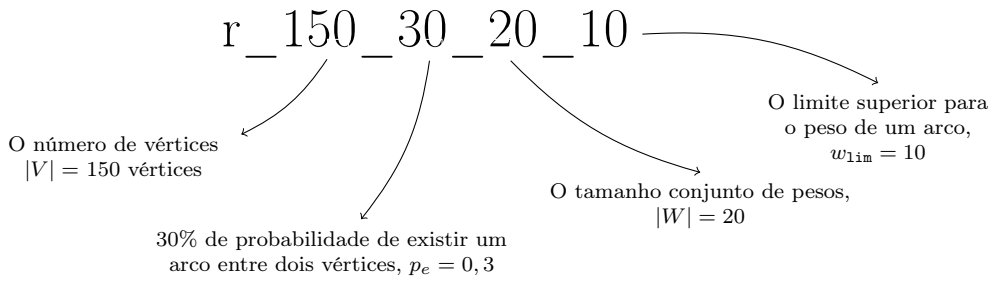


Figura 5.1: Exemplo dos parâmetros para a geração da instância `r_150_30_20_10`.

Considerando a equivalência do NCP com o MPSS, é possível definir uma métrica de densidade d para as instâncias aleatórias, derivada da mesma métrica utilizada para problemas de MPSS [Pan & Zhang, 2016]:

$$d = \frac{m}{n \cdot \log_2(\max(s_i))} \rightarrow \frac{|E|}{|V| \cdot \log_2(\max(w))} \tag{5.1}$$

Note que a métrica de densidade também é válida para instâncias de PSS, onde $n = 1$. Ou seja, a densidade associada ao PSS de um vértice no grafo pode ser calculada utilizando os arcos incidentes. Para o PSS e MPSS, a métrica de densidade está fortemente correlacionada com a dificuldade de resolver uma instância, onde uma densidade baixa é um indicativo de que é possível encontrar uma solução para a instância com menor esforço computacional.

Para analisar o desempenho dos algoritmos, foi gerado um conjunto de oito grafos com o objetivo de produzir instâncias heterogêneas divididas em três classes: **pequenas**, **médias** e **grandes**, com relação ao número de vértices e arcos. Cada classe foi subdivida em dois tipos diferentes de instâncias: **densas** e **esparsas**, em relação à métrica 5.1 de densidade.

A Tabela 5.1 apresenta as instâncias, onde a coluna $\delta_{\texttt{médio}}$ é referente ao grau médio dos vértices, a coluna $w_{\texttt{max}}$ define o maior peso de um arco, $w_{\texttt{médio}}$ o valor médio do peso dos arcos, a coluna d é referente à densidade de cada grafo, a coluna $d_{\texttt{médio}}$ é o valor médio de densidade para as instâncias de PSS associadas a cada vértice e, por fim a coluna $d_{\texttt{max}}$ corresponde à maior densidade de um PSS para um vértice no grafo.

5.2 Pré-Processamento das Instâncias

O objetivo do algoritmo de pré-processamento descrito na Seção 3.2 é eliminar vértices e arcos que nunca serão parte de um *nilcatenation*, com a expectativa de diminuir a complexidade de uma instância. Dessa forma, com o objetivo de validar sua aplicação,

Tabela 5.1: Instâncias Aleatórias de Teste

Classe	Tipo	Instância	$\|V\|$	$\|E\|$	$\delta_{\texttt{médio}}$	$w_{\texttt{max}}$	$w_{\texttt{médio}}$	d	$d_{\texttt{médio}}$	$d_{\texttt{max}}$
pequenas	densas	r_30_40_30_20	30	391	13,03	30	15,60	2,66	5,36	7,95
		r_40_40_30_20	40	633	15,82	26	14,57	3,37	6,74	9,36
	esparsas	r_60_10_50_5	60	383	6,38	49	34,39	1,13	2,27	4,09
		r_80_8_60_8	80	514	6,42	43	31,80	1,18	2,38	4,42
médias	densas	r_50_50_90_45	50	1286	25,72	89	47,92	3,97	7,94	9,42
		r_75_35_100_50	75	2080	27,73	99	50,71	4,18	8,38	10,56
	esparsas	r_100_8_60_10	100	865	8,65	60	35,4	1,46	2,93	4,40
		r_150_6_80_10	150	1447	9,65	78	34,7	1,53	3,08	4,93
grandes	densas	r_200_20_80_40	200	8280	41,40	80	42,96	6,55	13,09	15,66
		r_300_15_100_50	300	14101	47	100	50,08	7,07	14,16	18,51
	esparsas	r_400_2_40_15	400	3394	8,49	39	20,46	1,60	3,23	6,62
		r_600_1_50_10	600	3862	6,44	35	24,51	1,25	2,53	4,87

o algoritmo foi implementado em C++ e executado para cada uma das instâncias de teste.

A Tabela 5.2 apresenta os resultados de execução, as colunas $|V|$, $|E|$, d são referentes ao número de vértices, o número de arcos, e a densidade do grafo antes do pré-processamento. De forma similar, as colunas $|V_{\texttt{novo}}|$, $|E_{\texttt{novo}}|$, $d_{\texttt{novo}}$ são referentes a essas mesmas características após a execução do pré-processamento. Por fim, a coluna t apresenta o tempo de execução do algoritmo em segundos.

Ao analisar os resultados da Tabela 5.2, está claro que o algoritmo de pré-processamento apenas eliminou vértices e arcos do grafo nas instâncias do tipo **esparsas**. Esse comportamento é explicado pelas características de cada tipo de grafo, especificamente o grau médio de um vértice pertencente ao grafo. Uma condição simples para um vértice v não ser excluído do grafo, é que algum peso de um arco que entre em v seja igual à um peso de uma arco que saia de v:

$$A = \{w(e_j) \mid e_j \in E^-(v)\}$$
$$B = \{w(e_l) \mid e_l \in E^+(v)\}$$
$$A \cap B \neq \emptyset$$

Essa condição garante que, caso verdadeira, v não pode ser retirado do grafo. No entanto, é mais fraca que a restrição oriunda de um problema de PSS associado ao vértice, já que não considera as combinações de pesos entre os arcos. Mesmo assim,

Tabela 5.2: Resultados para o pré-processamento de instâncias.

Classe	Tipo	Instância	Reduziu o grafo?	$\|V\|$	$\|E\|$	d	$\|V_{\text{novo}}\|$	$\|E_{\text{novo}}\|$	d_{novo}	t (s)
pequenas	densas	r_30_40_30_20	Não	30	391	2,66	-	-	-	0,01
		r_40_40_30_20	Não	40	633	3,37	-	-	-	0,01
	esparsas	r_60_10_50_5	Sim	60	383	1,13	58	289	0,88	0,79
		r_80_8_60_8	Sim	80	514	1,18	77	295	0,70	2,4
médias	densas	r_50_50_90_45	Não	50	1286	3,97	-	-	-	0,66
		r_75_35_100_50	Não	75	2080	4,18	-	-	-	1,33
	esparsas	r_100_8_60_10	Sim	100	865	1,46	100	802	1,35	2,66
		r_150_6_80_10	Sim	150	1447	1,53	150	1370	1,45	8,92
grandes	densas	r_200_20_80_40	Não	200	8280	6,55	-	-	-	12,19
		r_300_15_100_50	Não	300	14101	7,07	-	-	-	34,29
	esparsas	r_400_2_40_15	Sim	400	3394	1,60	400	3260	1,54	61,36
		r_600_1_50_10	Sim	600	3862	1,25	536	2436	0,88	434,97

ela é útil para entender a dificuldade de retirar vértices de um grafo que seja denso. Dessa forma, considere o conjunto W de pesos que gera os elementos dos conjuntos A e B, dado um peso $w \in A$, a probabilidade de $w \notin B$ é calculada por:

$$\mathsf{P}(w \in A \mid w \notin B) = \left(\frac{|W| - 1}{|W|}\right)^{|B|}$$

Portanto a probabilidade de $\forall w \in A, w \notin B$, ou seja, que $A \cap B = \emptyset$ é:

$$\mathsf{P}(A \cap B = \emptyset) = \mathsf{P}(w \in A \mid w \notin B)^{|A|}$$

Assim podemos definir que $\mathsf{P}(A \cap B \neq \emptyset) = 1 - \mathsf{P}(A \cap B = \emptyset)$. Como $|A| = |E^-(v)|$ e $|B| = |E^+(v)|$, é possível notar que com o crescimento do grau de v, a probabilidade de existir algum arco que entra em v com peso igual a um outro arco que saia de v cresce rapidamente. Por outro lado, ao aumentar o tamanho do conjunto de pesos, a probabilidade tende a diminuir. De modo similar, um arco e_j com peso w_j que conecta o vértice v_i ao vértice v_k não pode ser excluído caso exista algum outro arco com peso igual a w_j em $E^+(v_i)$ ou em $E^-(v_k)$.

Portanto, para as instâncias **densas** onde os valores de grau médio de um vértice são maiores, o algoritmo de pré-processamento não é aplicável. Contudo, o algoritmo pode ser empregado para as instâncias **esparsas**, nas quais $\delta_{\texttt{médio}} \leq 10$, conseguindo eliminar vértices e arcos, consequentemente diminuindo o valor da métrica de densidade das instâncias.

5.3 Resultados Computacionais sem Pré-Processamento

Com o objetivo de analisar o desempenho e o comportamento dos algoritmos descritos neste trabalho, foram executados para cada instância de teste os algoritmos `LocB`, `B&B` e `B&B + LocB`.

O `GAP` de integralidade é uma métrica importante para analisar o desempenho de algoritmos de PLI, sendo um indicador da qualidade da solução encontrada. Dado que $M_{\texttt{frac}}$ corresponde ao melhor limite superior da relaxação linear, e $M_{\texttt{int}}$ o valor da solução inteira encontrada, o valor do `GAP` de integralidade para um problema de maximização é dado por:

$$\texttt{GAP} = \frac{M_{\texttt{frac}} - M_{\texttt{int}}}{M_{\texttt{int}}}$$

A Tabela 5.3 apresenta os resultados computacionais da execução dos algoritmos sem o algoritmo de pré-processamento, onde foi limitado o tempo de execução para cada algoritmo/instância em 3600 segundos. Cada coluna representa o valor da função objetivo para respectivo algoritmo, ou seja, o número de arcos do *nilcatenation* encontrado. Ademais, são apresentados os valores de `GAP` de integralidade para os algoritmos `LocB`, `B&B` e `B&B + LocB`. Para o `LocB` não é possível definir diretamente um valor para o `GAP`, portanto foi utilizado o valor do limite superior $M_{\texttt{frac}}$ gerado pela relaxação linear do algoritmo `B&B`.

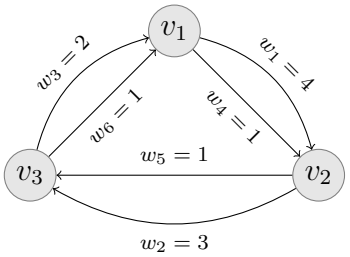
Analisando a Tabela 5.3, é possível perceber que foram encontradas soluções ótimas apenas para as instâncias `r_60_10_50_5` e `r_80_8_60_8`, ambas da classe **pequenas** e **esparsas**. Essas duas instâncias são notavelmente as de menor complexidade, possuindo baixa densidade e número de vértices e arcos reduzidos. Para as outras instâncias os valores de `GAP` são elevados, indicando que a relaxação linear do modelo não produz limites superiores próximos ao valor das soluções inteiras. A Figura 5.2 apresenta um exemplo simples da relaxação linear para um grafo com apenas três vértices e seis arcos. Note que apesar da dimensão reduzida do grafo de exemplo, as características das restrições causam que o valor da relaxação linear não seja próximo do valor da solução inteira.

Desse modo, os valores de `GAP` elevados são explicados pelo fato que se um arco e_j é parte de uma componente fortemente conexa do grafo, ainda que não seja possível que e_j seja parte da solução, durante a relaxação linear é possível definir uma combinação linear do seu peso de forma que e_j pertença ao *nilcatenation* do grafo.

Nenhum dos algoritmos conseguiu encontrar solução dentro do tempo limite para

Tabela 5.3: Resultados computacionais para os algoritmos sem pré-processamento.

Classe	Tipo	Instância	LocB	GAP LocB	B&B	GAP B&B	B&B + LocB	GAP B&B + LocB
pequenas	densas	r_30_40_30_20	**244**	0,29	197	0,61	190	0,67
		r_40_40_30_20	**329**	0,65	302	0,80	299	0.82
	esparsas	r_60_10_50_5	**142**	0,00	**142**	0,00	142	0,00
		r_80_8_60_8	**91**	0,00	**91**	0,00	91	0,00
médias	densas	r_50_50_90_45	**104**	10,29	**104**	10,29	89	12,19
		r_75_35_100_50	**142**	10,29	**142**	12,27	139	12,56
	esparsas	r_100_8_60_10	**68**	6,98	**68**	6,98	62	7.79
		r_150_6_80_10	269	2,50	**306**	2,08	263	2,59
grandes	densas	r_200_20_80_40	**130**	58,22	102	74,48	127	59,63
		r_300_15_100_50	**79**	165,09	73	178,74	72	181,33
	esparsas	r_400_2_40_15	**83**	25,32	79	26,66	69	30,65
		r_600_1_50_10	-	-	-	-	-	-



$$\max \sum_{j=1}^{m} x_j$$

$$\text{s.a. } 4x_1 + x_4 = 3x_2 + x_5$$
$$3x_2 + x_5 = 2x_3 + x_6$$
$$2x_3 + x_6 = 4x_1 + x_4$$

Relaxação linear:	Solução inteira:
$x_j \in [0, 1]$	$x_j \in \{0, 1\}$
max $z = 5,16$	max $z = 3$
$X = \{\frac{1}{2}, \frac{2}{3}, 1, 1, 1, 1\}$	$X = \{0, 0, 0, 1, 1, 1\}$

Figura 5.2: Exemplo da relaxação linear para o NCP. GAP $= \dfrac{5,16 - 3}{3} = 0,72$.

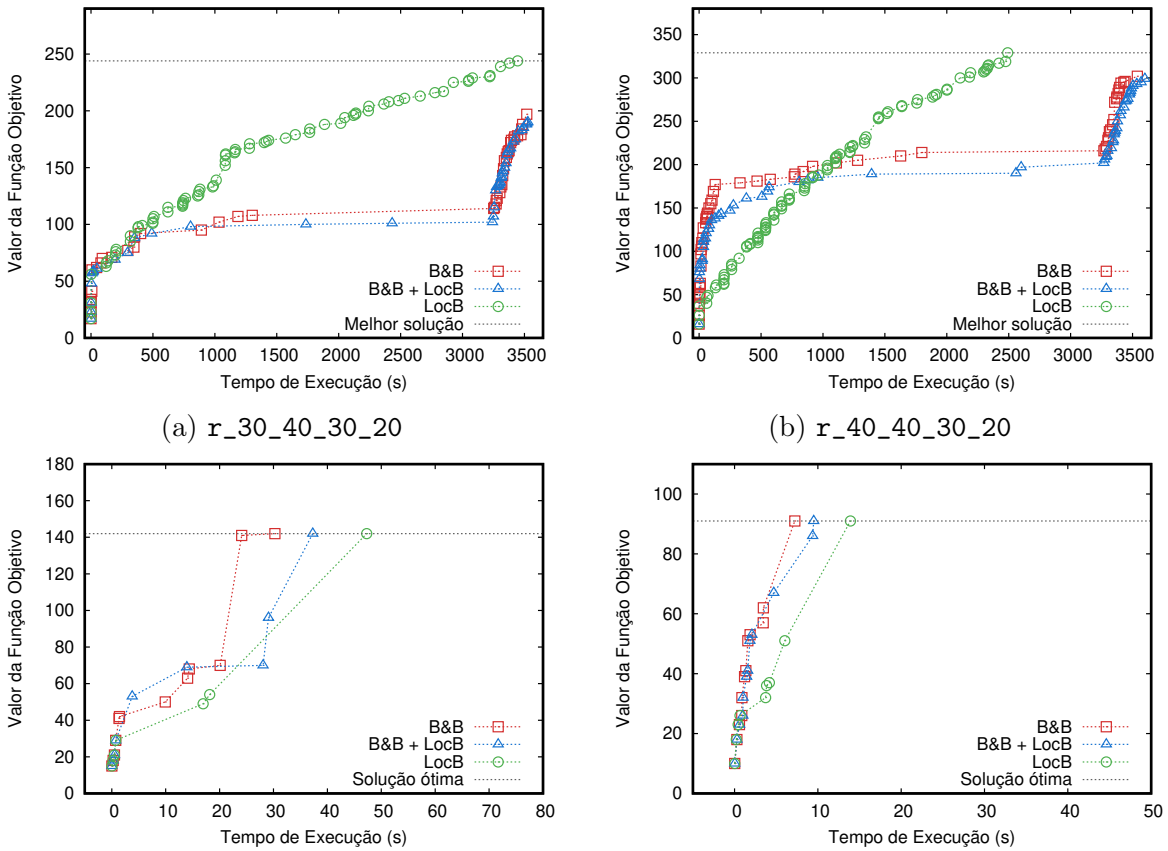
a instância r_600_1_50_10, que é a instância com o maior número de vértices. O modelo apresentado no Capítulo 3 define uma restrição de conservação de saldo para cada vértice no grafo, o que pode explicar a razão a dificuldade de encontrar soluções viáveis para essa instância.

Considerando as onze instâncias onde foram encontradas soluções, o algoritmo de LocB encontrou a melhor solução em dez instâncias, sendo que em cinco de forma

isolada e nas outras seis, o B&B também encontrou a mesmo valor para a função objetivo. Apenas para a instância r_150_6_80_10, o algoritmo B&B teve melhor desempenho isoladamente. Por fim, para nenhuma das instâncias o B&B + LocB encontrou a melhor solução.

5.3.1 Discussão dos resultados para as instâncias da classe pequenas

A Figura 5.3 apresenta a relação das soluções incumbentes encontradas ao longo da execução dos algoritmos B&B, B&B + LocB e LocB para as instâncias da classe **pequenas**. Os gráficos também apresentam a melhor solução encontrada, e no caso das instâncias r_60_10_50_5 e r_80_8_60_8, a solução ótima.



(a) r_30_40_30_20

(b) r_40_40_30_20

(c) r_60_10_50_5

(d) r_80_8_60_8

Figura 5.3: Progresso da execução dos algoritmos para cada instância da classe **pequenas**. Cada gráfico apresenta a relação da melhor solução encontrada pelo tempo de execução em segundos.

Para as instâncias dessa classe, os comportamentos dos algoritmos B&B e B&B + LocB são muito parecidos ao longo do tempo. Nas curvas de execução para as instâncias

r_30_40_30_20 e r_40_40_30_20 é possível notar que ambos algoritmos encontram um conjunto de soluções melhorantes em pouco tempo, porém em seguida é atingido um platô, circunstância que é mantida durante quase todo o tempo de execução. Nesse caso, ambos algoritmos apresentaram uma convergência prematura, alcançando um ótimo local rapidamente. Somente no final da execução, ambos algoritmos conseguiram explorar outra região viável, encontrando assim um novo conjunto de soluções melhorantes.

Por outro lado, o LocB teve um comportamento praticamente linear, encontrando soluções melhorantes sucessivamente, até encontrar a melhor solução em menos da metade do tempo limite de execução. No entanto, não é possível provar a otimalidade dessa solução, em outras palavras, não existe garantia que o LocB não tenha também encontrado apenas um ótimo local.

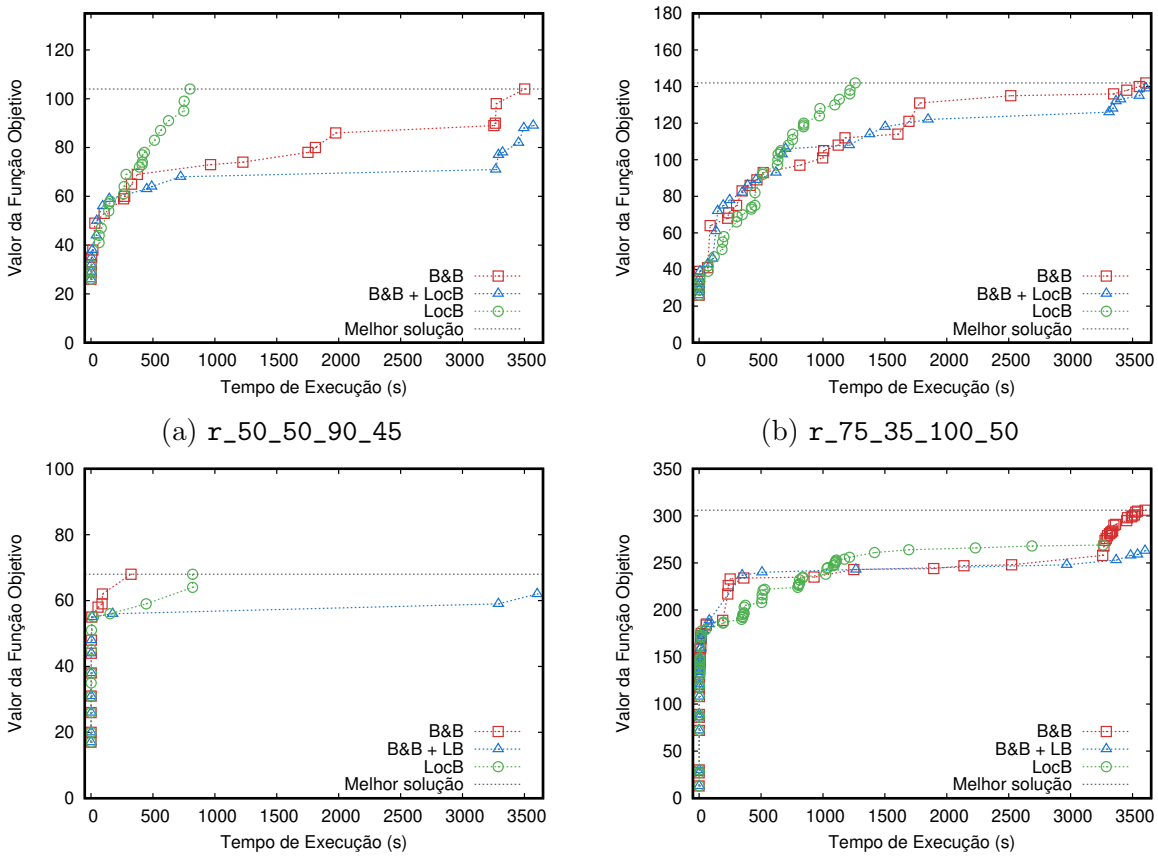
5.3.2 Discussão dos resultados para as instâncias da classe médias

A Figura 5.4 apresenta a relação das soluções incumbentes encontradas ao longo da execução dos algoritmos para as instâncias da classe **médias**. Os gráficos também apresentam a melhor solução encontrada.

Para as instâncias r_50_50_90_45, r_75_35_100_50 e r_100_8_60_10 os algoritmos mantiveram o mesmo comportamento observado para as execuções em instâncias da classe **pequenas**. Porém para a instância r_150_6_80_10 o algoritmo LocB não encontrou a melhor solução, apresentando uma convergência prematura em um ótimo local. Um motivo provável para tal comportamento é o parâmetro de entrada $t_{\texttt{nó}}$, que define o tempo limite de execução do resolvedor PLI para um nó. Se o resolvedor caixa-preta não conseguir melhorar a solução dentro do tempo limite para uma determinada região, definida por uma restrição de ramificação à esquerda $\Delta(x, \bar{x}) \leq k$, eventualmente será executado um processo de diversificação movendo a busca dessa região. No entanto, não existe garantia que o ótimo global não era parte dessa região.

5.3.3 Discussão dos resultados para as instâncias da classe grandes

A Figura 5.5 apresenta a relação das soluções incumbentes encontradas ao longo da execução dos algoritmos para as instâncias da classe **grandes**. Os gráficos também apresentam a melhor solução encontrada. A instância r_600_1_50_10 foi omitida, pois não foram encontradas soluções dentro do tempo limite.



(a) `r_50_50_90_45` (b) `r_75_35_100_50`

(c) `r_100_8_60_10` (d) `r_150_6_80_10`

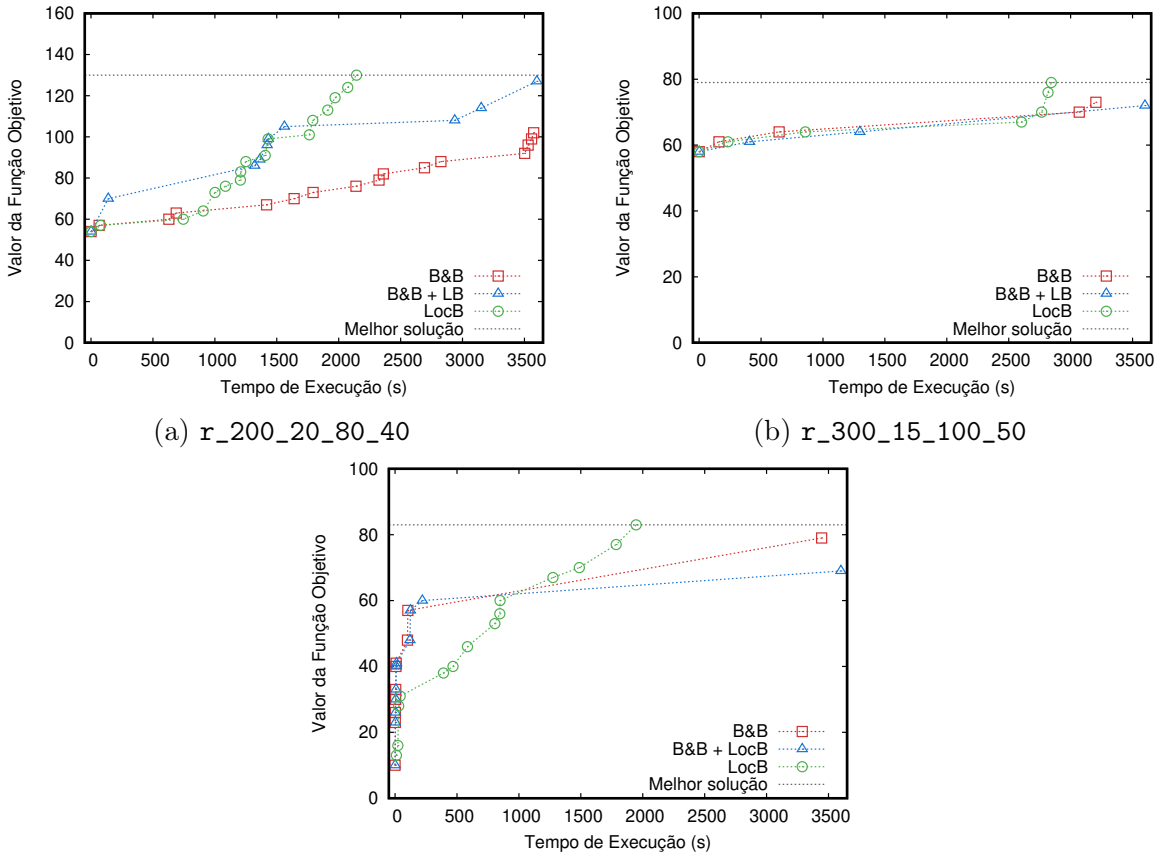
Figura 5.4: Progresso da execução dos algoritmos para cada instância da classe **médias**. Cada gráfico apresenta a relação da melhor solução encontrada pelo tempo de execução em segundos.

O comportamento de execução dos algoritmos para instâncias da classe **grandes** foi similar ao apresentado para a execução em instâncias das classes **pequenas** e **médias**. O algoritmo de `LocB` encontrou soluções melhorantes de forma incremental, enquanto o `B&B` e `B&B + LocB` convergiram prematuramente para um ótimo local.

Como não foi encontrada solução para a instância `r_600_1_50_10` dentro do tempo limite estipulado, existe a probabilidade do modelo gerado pelo grafo ser inviável. No entanto, não foi encontrado um certificado de inviabilidade para a instância.

5.4 Resultados Computacionais com Pré-Processamento

A fim de conhecer o impacto do algoritmo de pré-processamento no desempenho dos algoritmos estudados, os algoritmos `B&B`, `B&B + LocB` e `LocB`. O experimento compu-



(a) `r_200_20_80_40`

(b) `r_300_15_100_50`

(c) `r_400_2_40_15`

Figura 5.5: Progresso da execução dos algoritmos para cada instância da classe **grandes**. Cada gráfico apresenta a relação da melhor solução encontrada pelo tempo de execução em segundos. A instância `r_600_1_50_10` foi omitida, pois não foram encontradas soluções dentro do tempo limite.

tacional considerou apenas as instâncias **esparsas**, já que somente nesse tipo de grafo o pré-processamento conseguiu reduzir as instâncias.

A Tabela 5.4 apresenta os resultados computacionais para a execução limitada em 3600 segundos descontando o tempo t gasto pelo algoritmo de pré-processamento apresentado na Tabela 5.2.

Não foi possível notar nenhuma melhora no desempenho dos algoritmos `B&B`, `B&B + LocB` e `LocB` com a execução do pré-processamento. De fato, os valores dos *nilcatenations* para algumas instâncias foi menor, indicando um possível aumento na convergência prematura.

Tabela 5.4: Resultados computacionais para os algoritmos com pré-processamento.

Classe	Instância	LocB	GAP LocB	B&B	GAP B&B	B&B + LocB	GAP B&B + LocB
pequenas	r_60_10_50_5	142	0,00	142	0,00	142	0,00
	r_80_8_60_8	91	0,00	91	0,00	91	0,00
médias	r_100_8_60_10	68	6,98	67	7,09	60	8,04
	r_150_6_80_10	260	2,62	304	2,10	263	2,59
grandes	r_400_2_40_15	83	25,32	75	28,13	63	33,64
	r_600_1_50_10	-	-	-	-	-	-

5.5 Análise do Algoritmo LLL aplicado ao NCP

Géraud et al. [2017] propõe o uso do algoritmo de LLL como uma ferramenta para encontrar soluções para o NCP. De modo geral, o LLL se baseia em encontrar em um tempo polinomial uma base reduzida B formada por vetores com coordenadas inteiras para uma rede diagonal. Para problemas de PLI, a rede diagonal representa uma instância do problema e o LLL tem como objetivo encontrar o menor vetor de inteiros na base reduzida. Além disso, os vetores em B são quase ortogonais e com a norma L^2 pequena, onde cada vetor é potencialmente uma solução para a instância do problema de PLI.

Como o NCP é um problema NP-completo, o algoritmo LLL pode não encontrar uma solução para uma instância do NCP, apresentando assim um comportamento probabilístico. Ademais, no contexto de aplicações para o NCP, é importante maximizar o número de arcos dos *nilcatenations* encontrados.

Dessa forma, a hipótese sugerida por esse trabalho é que o LLL gera soluções com um número pequeno de arcos, pois como o LLL busca o menor vetor com coordenadas inteiras, consequentemente a cardinalidade das soluções encontradas será pequena. Essa hipótese é validada por meio da execução do algoritmo de LLL implementado em C++ e usando a biblioteca `fplll`. A Tabela 5.5 apresenta os resultados da execução do `NCP-LLL` para as instâncias de teste. A fim de comparação, também foi executado o `B&B` com um limite de tempo de 60 segundos. Foram consideradas apenas as classes **pequenas** e **médias**, pois não foi possível encontrar soluções usando o `NCP-LLL` dentro de um tempo limite de 3600 segundos para instâncias da classe **grandes**.

Apesar de o algoritmo de LLL encontrar soluções em tempo polinomial, onde seis instâncias são resolvidas em menos de 60 segundos, a cardinalidade das soluções encontradas foi muito pequena. De fato, em comparação com a execução do `B&B`, em

Tabela 5.5: Resultados computacionais do algoritmo NCP-LLL

Classe	Tipo	Instância	NCP-LLL	Tempo	B&B	$\frac{\text{NCP-LLL}}{\text{B\&B}}$
pequenas	densas	r_30_40_30_20	7	2s	100	0,07
		r_40_40_30_20	9	12s	164	0,05
	esparsas	r_60_10_50_5	6	2s	142	0,04
		r_80_8_60_8	7	8s	91	0,08
médias	densas	r_50_50_90_45	4	227s	49	0,08
		r_75_35_100_50	3	1860s	42	0,07
	esparsas	r_100_8_60_10	7	56s	55	0,12
		r_150_6_80_10	7	364s	184	0,04
					Média	**0,07**

média o NCP-LLL encontra *nilcatenations* com apenas 7% da quantidade de arcos das soluções geradas pelo B&B. Portanto, para as instâncias analisadas a hipótese de que o LLL gera soluções com poucos arcos é válida.

5.6 Considerações Finais

Os resultados apresentados neste capítulo definem um *baseline* comparativo para futuros algoritmos e formulações para o NCP. É importante destacar que a formulação proposta por este trabalho possui uma relaxação linear não apertada, apresentando melhores resultados de GAP para instâncias **esparsas** quando comparados às instâncias **densas**.

Não foi encontrado um caso dentro das instâncias de teste que justifique o uso do algoritmo B&B + LocB, já que seu desempenho foi consistentemente pior para todas as instâncias. E ainda, tampouco o algoritmo NCP-LLL é aplicável ao NCP, quando o objetivo é maximizar o número de arcos dos *nilcatenations* encontrados.

Finalmente, o algoritmo LocB foi o melhor para a maioria das instâncias, com exceção de apenas a instância r_150_6_80_10. Todavia, é necessário ressaltar que o LocB é um algoritmo parametrizável e não existem garantias que os parâmetros utilizados nos experimentos terão o mesmo comportamento para qualquer instância genérica do NCP.

Capítulo 6

Aplicação do NCP para Detecção de Lavagem de Dinheiro em Criptomoedas

As transações executadas no Bitcoin [Nakamoto, 2008] não são verdadeiramente anônimas. De fato, as transações são atreladas a endereços pseudo-anônimos, existindo assim a possibilidade de agrupar as transações de um usuário. Cada usuário no Bitcoin é identificado por meio de um conjunto de chaves públicas referentes aos seus endereços. Dessa forma, com o objetivo de de-anonimizar as transações, um atacante pode tentar construir um mapeamento entre cada usuário e suas chaves públicas.

Seguindo essa premissa, o histórico de transações de uma criptomoeda pode ser modelado como um grafo, onde os vértices representam os usuários e cada arco uma transação entre duas partes [Reid & Harrigan, 2011]. Para isso, existem heurísticas que agrupam os endereços de cada transação com o intuito de co-relacionar cada transação com um possível usuário [Meiklejohn et al., 2013].

Existem sinais de que uma quantidade considerável de usuários e transações do Bitcoin está relacionada à atividades irregulares, como por exemplo, a lavagem de dinheiro utilizando transações entre duas ou mais partes. Desse modo, considerando a pseudo-anonimidade do Bitcoin e a viabilidade de modelar seu histórico de transações como um grafo, este capítulo aborda a proposta de aplicação do NCP como um mecanismo para descoberta de estruturas em grafos possivelmente relacionadas com esquemas de lavagem de dinheiro. A escolha pelo Bitcoin é devida a sua importância no mundo financeiro, sendo a primeira criptomoeda a alcançar um impacto significativo com ampla utilização.

O presente capítulo está organizado da seguinte forma: na Seção 6.1 são intro-

duzidas algumas definições necessárias sobre Bitcoin. Na Seção 6.2 é introduzida a heurística de geração do grafo de usuários a partir do histórico de transações, que é utilizada por este trabalho. Em seguida, na Seção 6.3 são apresentadas as considerações relacionadas à anonimidade de transações no Bitcoin e os métodos existentes de ofuscamento de transações, que são comumente chamados de *mixers* ou *tumblers*. Por fim, na Seção 6.4 são apresentados os experimentos realizados para validar a hipótese de que o NCP pode ser aplicado para encontrar estruturas relacionadas à lavagem de dinheiro.

6.1 Definições sobre Bitcoin

As definições a seguir são baseadas na documentação do Bitcoin [Bitcoin Project, 2020] e em Antonopoulos [2014].

Bitcoin é uma criptomoeda descentralizada que depende de uma rede par-a-par e um protocolo de prova de trabalho para evitar gasto duplo. Um endereço no Bitcoin é representado por uma chave pública atrelada a um usuário, que detém a chave privada correspondente. Um único usuário pode possuir múltiplos endereços, normalmente gerenciados por *softwares* conhecidos como carteiras (*wallets*). Uma unidade de conta do sistema Bitcoin é chamada de *bitcoin* (BTC) e a menor sub-unidade é o *satoshi* que equivale a 0,00000001 *bitcoin*.

As transações no Bitcoin ocorrem entre endereços e são armazenadas em estruturas chamadas de blocos. Cada um dos blocos possui um *hash* identificador e um apontador para o bloco anterior, formando assim uma cadeia verificável conhecida como *blockchain*. O conceito de *blockchain* vai além do âmbito das criptomoedas, porém quando referido neste trabalho denota apenas o conjunto de blocos que formam o histórico de transações no Bitcoin.

Um aspecto importante do Bitcoin é que as transações só são consideradas válidas após serem processadas por um participante especial da rede chamado de minerador. Esse processo recebe o nome de mineração, em que um minerador resolve o desafio de prova de trabalho para gerar o próximo bloco, agregando e validando um conjunto de transações, que em seguida são adicionadas ao *blockchain*. A Figura 6.1 apresenta, de forma simplificada, a estrutura do *blockchain* do Bitcoin.

Uma única transação consiste de um conjunto de entradas e saídas, o valor transferido e uma taxa a ser paga para quem minerar o bloco que contém a transação. Cada entrada é uma referência à uma das saídas de alguma transação passada que ainda não foi gasta, chamada de *unspent transaction output* (UTXO) e define o endereço de onde

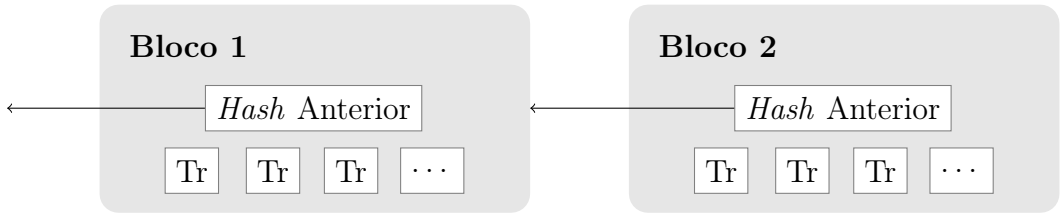


Figura 6.1: A estrutura simplificada do *blockchain* do Bitcoin. Cada bloco pode conter n transações que foram validadas.

será debitado o valor da transação. Consequentemente, as saídas formam um conjunto de UTXOs atrelados aos endereços que receberão o valor da transação.

As saídas de uma transação quando utilizadas como entrada em outra transação, devem ser gastas por completo. Caso o valor da transação seja menor do que o valor correspondente aos endereços de entrada, é necessário criar uma saída adicional onde o valor restante é enviado de volta ao usuário, chamada de endereço de troco. A Figura 6.2 apresenta, de modo simplificado, a estrutura das transações no Bitcoin.

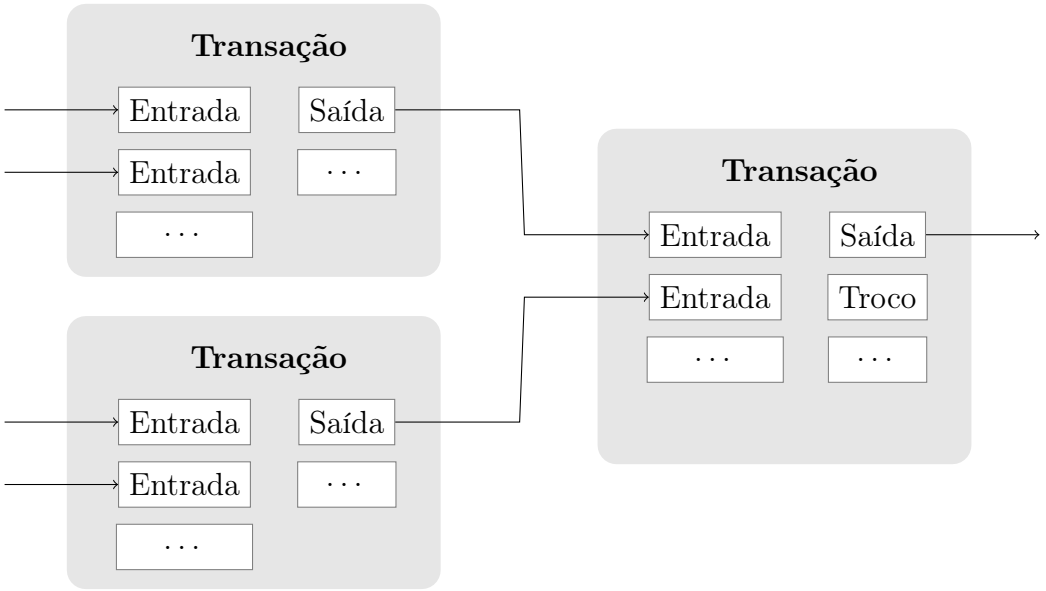


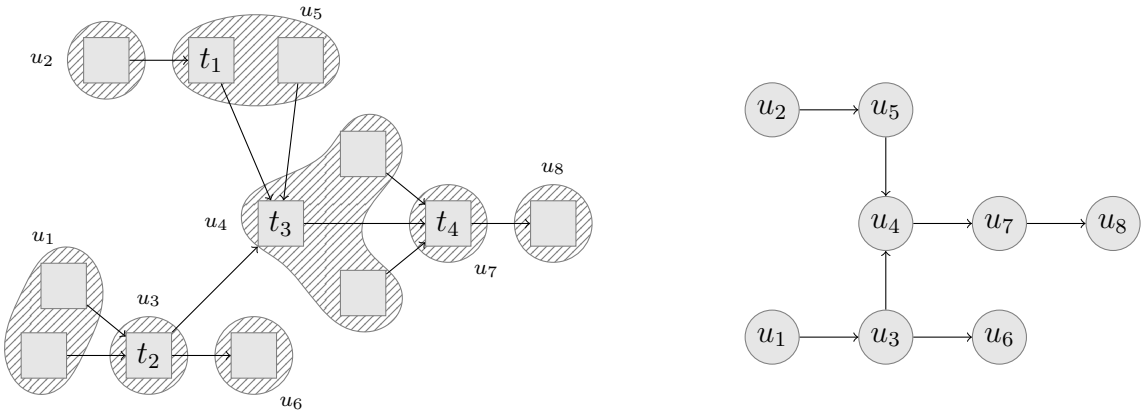
Figura 6.2: Estrutura simplificada das transações no Bitcoin.

6.2 Heurística para Geração do Grafo de Usuários

Reid & Harrigan [2011] apresentam as estrutura topológicas de dois grafos derivados do histórico de transações do Bitcoin: o grafo de transações e o grafo de usuários.

O grafo de transações representa o fluxo de *bitcoins* entre transações ao longo do tempo. Cada vértice é referente a uma transação e cada arco representa a saída,

a entrada e o valor transferido. Por outro lado, o grafo de usuários representa o fluxo de *bitcoins* entre usuários ao longo do tempo, onde cada usuário é representado por um vértice e os arcos representam as transações entre distintos usuários. A Figura 6.3 exemplifica a diferença entre os dois grafos.



(a) O grafo de transações. Cada área sombreada corresponde a um usuário.

(b) Grafo de usuários derivado do grafo de transações.

Figura 6.3: Exemplos de um grafo de transações e um grafo de usuários do Bitcoin.

O grafo de transações pode ser extraído diretamente do *blockchain* do Bitcoin. No entanto, considerando a possibilidade que um único usuário controle múltiplos endereços, é necessário agrupar heuristicamente as transações para construir o grafo de usuários. Meiklejohn et al. [2013] propõe a seguinte heurística para realizar o agrupamento dos endereços presentes em cada transação, a fim de construir o grafo de usuários:

1. Se dois (ou mais) endereços são entradas de uma mesma transação, eles são controlados pelo mesmo usuário.

2. O endereço de troco é controlado pelo mesmo usuário dos endereços de entrada da transação.

Transações com múltiplas entradas normalmente acontecem quando um usuário não possui uma única UXTO com valor suficiente para executar uma transação. Dessa forma, é necessário agrupar as UXTOs a fim de alcançar o valor da transação. Nesse caso, a heurística funciona corretamente já que todos os endereços de entrada pertencem ao mesmo usuário. No entanto, existem protocolos que permitem múltiplos usuários criarem uma transação conjunta. Esse processo é chamado de CoinJoin [Maxwell,

2013], e para transações construídas desse modo, a heurística agrupa de modo incorreto os usuários.

Normalmente, o endereço de troco é um endereço nunca usado anteriormente, ou seja, é criado justamente para uma específica transação. Dessa forma, é considerado como endereço de troco, o endereço de saída que aparece pela primeira vez na história do *blockchain*. Porém se algum endereço de entrada também aparece na saída, então é considerado que não foi criado um novo endereço de troco, pois foi reutilizado um dos endereços do usuário como endereço de troco.

6.3 Anonimidade de Transações no Bitcoin

Transações no Bitcoin não são verdadeiramente anônimas, de fato todo o histórico de transações é público e a movimentação de *bitcoins* pode ser rastreada ao longo do tempo. Dessa forma, se uma chave pública é de alguma forma correlacionada com um indivíduo ou organização, o histórico de transações envolvendo essa chave é público e rastreável. A Figura 6.4 apresenta um exemplo de como é possível relacionar o endereço de um usuário à um outro endereço previamente conhecido, utilizando o grafo de transações do Bitcoin.

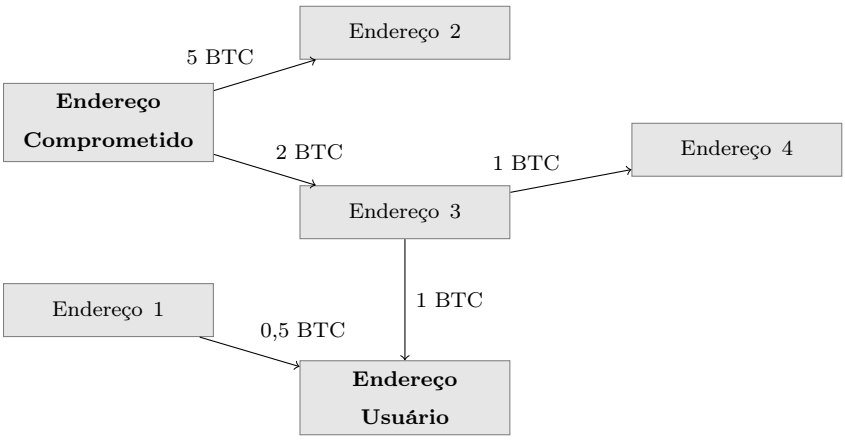


Figura 6.4: Exemplo da rastreabilidade de transações no Bitcoin. Analisando o grafo de transações, é possível facilmente descobrir que parte dos *bitcoins* do usuário é proveniente de um endereço comprometido.

Com o intuito de evitar o rastreamento da origem de *bitcoins*, existem serviços chamados de *mixers* ou *tumblers*, que têm como objetivo ofuscar a relação entre endereços no histórico de transações [Möser et al., 2013]. Para isso, o serviço de *mixing* recebe uma quantia de *bitcoins* de um usuário, e por meio de uma sequência de transações aleatórias, embaralha e combina esse valor utilizando *bitcoins* de outros usuários.

Porém, normalmente é necessário que o usuário pague uma taxa m para utilizar o *mixer*. A Figura 6.5 apresenta um esquema genérico de um serviço de *mixing*, onde um usuário envia uma transação com valor v e recebe de volta $v - m$ após uma sequência desconhecida de transações.

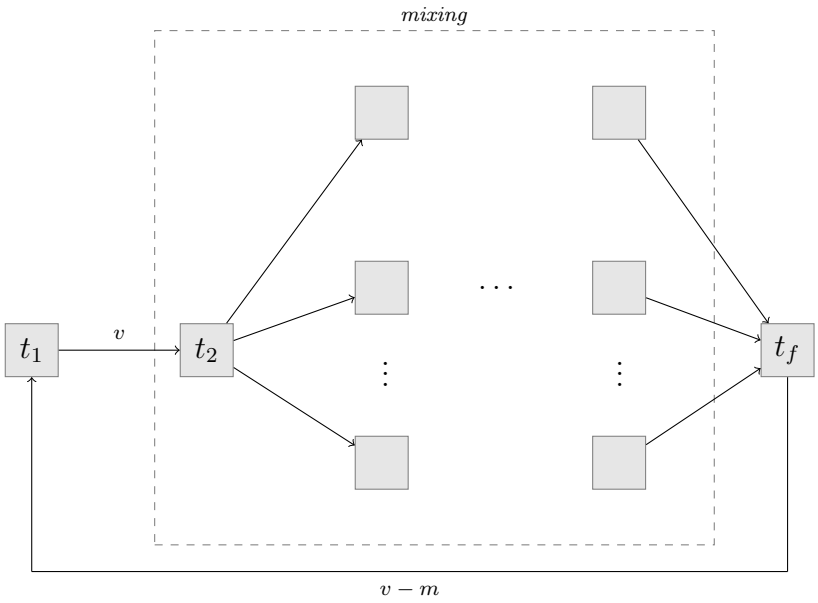


Figura 6.5: Esquema de um serviço de *mixing*. O usuário envia uma transação com valor v e recebe de volta $v - m$, pagando uma taxa m para o serviço.

Um serviço de *mixing* deve garantir que se um usuário envia uma quantia de *bitcoins*, depois de uma janela de tempo, o usuário receberá de volta o valor enviado menos uma taxa correspondente. Essa taxa normalmente deve ser aleatória, a fim de dificultar a identificação do processo por um observador externo. Existem vários protocolos para serviços de *mixing*, como por exemplo o Mixcoin [Bonneau et al., 2014], o CoinSuffle [Ruffing et al., 2014] e o Lockcoin [Bao et al., 2018].

6.4 Aplicação do NCP e Experimentos Computacionais

Uma das estruturas que surge no grafo de usuários e que pode ser considerada um indício de lavagem de dinheiro é um conjunto de transações que partem de um usuário e voltam até esse mesmo usuário sem alterar o saldo dos usuários envolvidos. Sob o ponto de vista do grafo de usuários, essa estrutura pode ser considerada um *nilcatenation*, já que cada usuário é representado por um vértice e o saldo é conservado para cada vértice.

Usuários que desejam realizar transações oriundas de atividades ilícitas, podem e normalmente utilizam serviços de *mixing* com o objetivo de aumentar o nível de anonimidade de suas transações [Möser et al., 2013]. Note que se a taxa do serviço de *mixing* é $m = 0$ e as transações do processo de *mixing* formam uma componente fortemente conexa no grafo de usuários, então a sequência de transações também pode ser modelada como *nilcatenation*.

Sem embargo, ainda é possível expandir a definição de conservação de saldo para um vértice, a fim de também considerar uma folga m, referente ao valor pago para executar o *mixing*. Com o objetivo de generalizar o valor de m, podemos definir a taxa como uma porcentagem p_m do valor transferido v e um valor fixo f_m, de modo que $0 \leq m \leq p_m \cdot v + f_m$. Portanto, a restrição de conservação de saldo pode ser expandida da seguinte forma:

$$\sum_{k \in E'^-(v_i)} w_k - p_m \sum_{k \in E'^-(v_i)} w_k - f_m \leq \sum_{j \in E'^+(v_i)} w_j \leq \sum_{k \in E'^-(v_i)} w_k \tag{6.1}$$

Dessa forma, o valor que entra em um determinado vértice está dentro de um intervalo que considera o valor pago como taxa ao serviço de *mixing*. A Figura 6.6 apresenta um exemplo de processo de *mixing* hipotético com $v = 100$, $p_m = 0,03$ e $f_m = 0,5$. Note que o grafo resultante pode ser modelado como um *nilcatenation* utilizando a restrição 6.1 de conservação de saldo expandida.

Outro fator a ser considerado são as taxas pagas para o nó minerador em cada transação. Uma estratégia simples para contornar esse detalhe, é distribuir o valor da taxa do minerador entre as saídas da transação.

Por último, a fim de validar a hipótese que o NCP pode ser aplicado na descoberta de estruturas relacionadas à lavagem de dinheiro, foram executados dois experimentos distintos:

- Validação da hipótese em um ambiente controlado, simulando transações na rede de testes do Bitcoin e em seguida executando o algoritmo `B&B` (descrito no Capítulo 5) para encontrar um *nilcatenation* no grafo de usuários. O intuito desse experimento é validar que a heurística de geração do grafo de usuários, quando combinada com o NCP, consegue de fato detectar possíveis estruturas relacionadas à lavagem de dinheiro.

- Extração de transações reais do *blockchain* do Bitcoin, levando em consideração o comportamento esperado de serviços de *mixing*. Por fim, a execução dos algoritmos `B&B` e `LocB` (também descrito no Capítulo 5) para um conjunto de grafos de usuários reais e análise dos resultados encontrados.

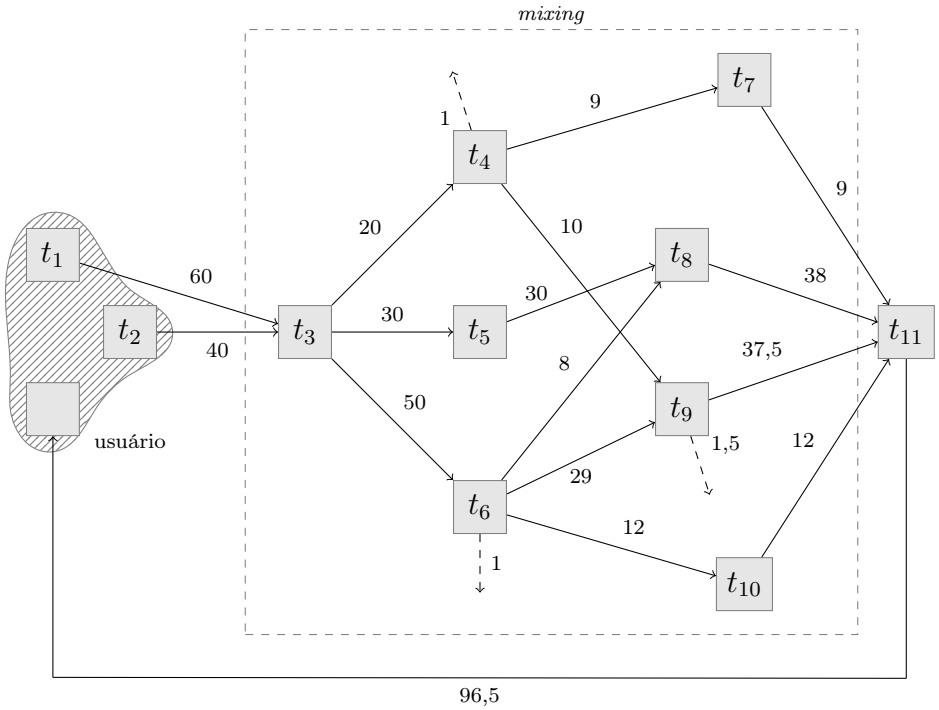


Figura 6.6: Exemplo de uma estrutura possivelmente relacionada à lavagem de dinheiro.

6.4.1 Validação do Algoritmo na Rede de Testes do Bitcoin

O Bitcoin permite testar aplicações com riscos e limitações reduzidas. Para isso existem três diferentes redes onde se pode criar transações entre endereços [Bitcoin Project, 2020]:

mainnet. É a rede oficial do Bitcoin, onde as transações possuem valor monetário e são validadas, transmitidas e armazenadas seguindo um protocolo estrito.

testnet. É uma rede de testes onde não existe valor monetário nas transações. Além disso, possui uma rede par-a-par e algumas relaxações em regras de transações presentes na *mainnet*.

regtest. Permite criar um *blockchain* privado com as mesmas regras da *testnet*, porém tendo controle total sobre a rede ao criar novos blocos. Em outras palavras, não é existe uma rede par-a-par para transmitir as transações.

Com o objetivo de validar a execução da heurística de geração do grafo de usuários e a aplicação do NCP em um ambiente controlado, foi criada uma instância aleatória `r_20_30_15_120`, utilizando o método descrito na Seção 5.1. Em seguida, essa instância foi simulada na rede *testnet*, onde para cada vértice da instância, foi criado uma carteira

com um endereço público e para cada arco foi gerada uma transação correspondente entre os endereços, com o valor do peso do arco. A Figura 6.7 apresenta a instância e o *nilcatenation* encontrado por meio do algoritmo `B&B`. A rede *testnet* foi escolhida para executar esse experimento, porque não há valor financeiro real atribuído à suas transações e possui características semelhantes à rede *mainnet*.

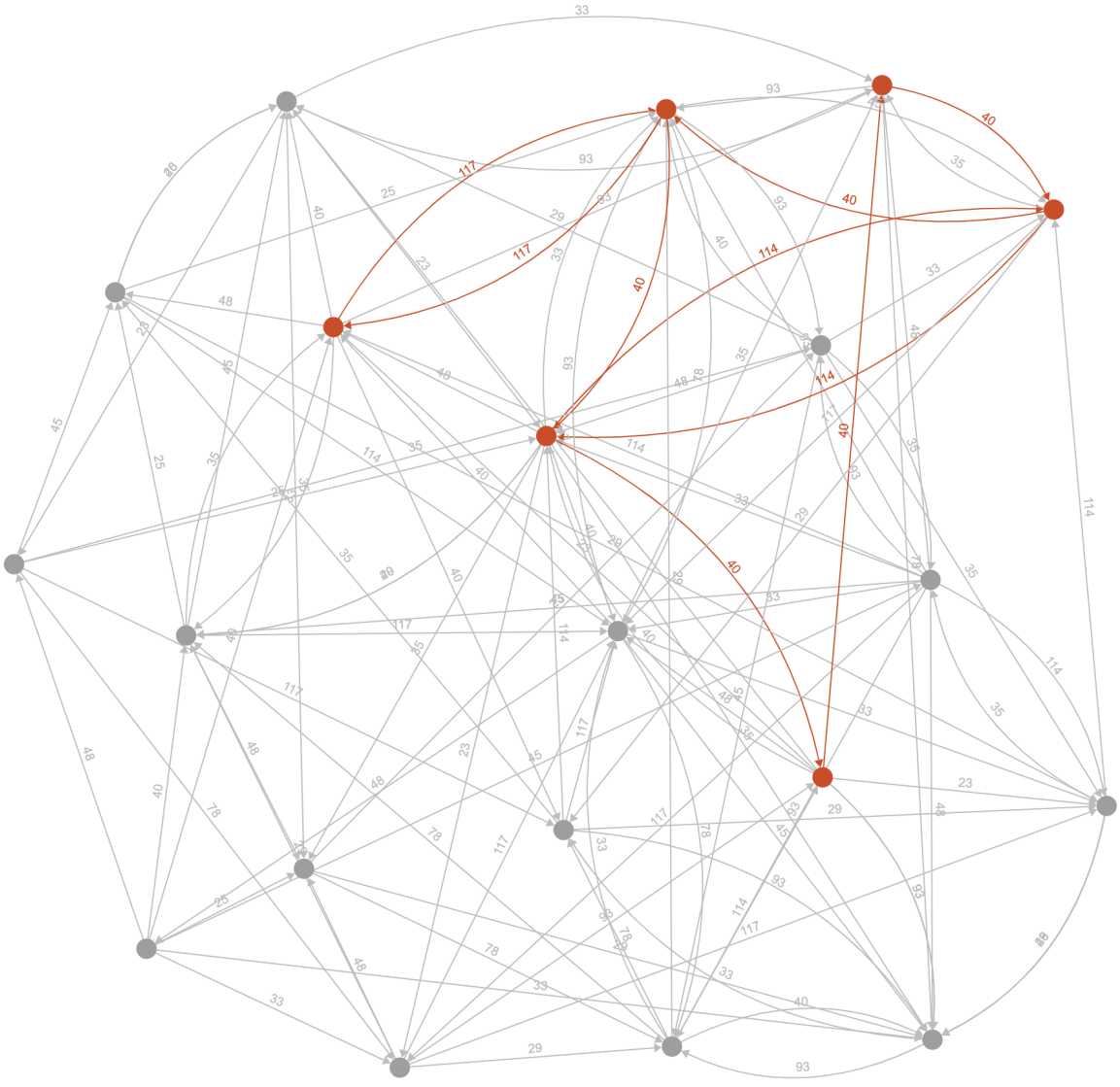


Figura 6.7: Solução para a instância artificial `r_20_30_15_120`. O subgrafo destacado corresponde ao *nilcatenation* encontrado.

O algoritmo para simular a instância `r_20_30_15_120` na rede *testnet* foi implementado em `Node.js` utilizando a biblioteca `bitcoinjs-lib`[1] para a geração dos endereços e transações. Para a distribuição das transações entre os nós da rede, foi

[1] `https://github.com/bitcoinjs/bitcoinjs-lib`

utilizada a `Esplora HTTP API`[2]. Por fim, a extração dos blocos da *testnet* foi realizada por meio da biblioteca `blockchain.info`[3].

A heurística de geração do grafo de usuários foi executada em 25 blocos da rede *testnet* que contêm as transações simuladas. A Tabela 6.1 exibe as características do grafo de usuários gerado pela heurística e a solução encontrada pelo `B&B`. Para esse experimento não foram consideradas as taxas de *mixing* e de mineração.

Tabela 6.1: Grafo de usuários extraído da rede de testes do Bitcoin.

Bloco Inicial	Bloco final	Janela de tempo	Usuários	Transações	`B&B`
1832390	1832415	08/09/2020 23:17:25 09/09/2020 01:00:47	989	1201	21

Após a análise da solução gerada, foi possível concluir que o *nilcatenation* encontrado inclui os arcos referentes às transações simuladas. Portanto, para o ambiente controlado da rede de testes, o algoritmo `B&B` combinado com a heurística de geração do grafo de usuários, conseguiu detectar a estrutura gerada pela instância `r_20_30_15_120`.

6.4.2 Aplicação na Rede de Transações do Bitcoin

Para garantir que a aplicação do NCP leve em consideração as características de diferentes serviços de *mixing*, foram levantados um conjunto com cinco diferentes serviços. A Tabela 6.2 apresenta os detalhes de cada *mixer*: o período de espera para o retorno da transação, a taxa cobrada e o valor mínimo da transação.

Tabela 6.2: Diferentes serviços de *mixing* para o Bitcoin.

Mixer	Período	Taxa (BTC)	Valor Mínimo (BTC)
Blender[4]	0 até 7 dias	0,5% até 2,014%	0,001
Bitcoin Fog[5]	customizável	1% até 3%	0,035
CryptoMixer[6]	0 até 96 horas	0.5% + 0,0005	0,001
Mixtum[7]	0 até 6 horas	4% até 5% + 0,0007	0,005
Bitmix[8]	0 até 24 horas	0,4% até 4% + 0,0003	0,005

[2] `https://github.com/Blockstream/esplora`
[3] `https://github.com/blockchain/api-v1-client-node`
[4] `https://blender.io`

De modo similar ao experimento realizado na rede de testes, foram extraídos da *mainnet* um conjunto de quatro grafos de usuários. Considerando os diferentes serviços de *mixing*, foi definida uma janela de tempo de transações de sete dias, período suficiente para abranger a maioria dos *mixers* avaliados. Como o sistema do Bitcoin adiciona em média um novo bloco a cada 10 minutos, foram extraídos 1000 blocos para cada grafo de usuários, com o intuito de alcançar a janela de tempo requerida.

A Tabela 6.3 apresenta as características das instâncias criadas pela heurística de geração do grafo de usuários. Note que para cada instância, existe uma grande quantidade de vértices isolados. No contexto da aplicação do NCP, esses vértices podem ser desconsiderados, já que o menor *nilcatenation* possível tem ao menos dois vértices.

Tabela 6.3: Grafos de usuários extraídos do Bitcoin.

Instância	Bloco Inicial	Bloco final	Janela de tempo	Usuários	Transações	Vértices isolados
B1	641633	642632	31/07/2020 13:14 07/08/2020 10:16	2927801	5622091	2242085
B2	642633	643632	07/08/2020 10:23 14/08/2020 02:21	2848205	5485983	2199952
B3	643633	644632	14/08/2020 02:40 20/08/2020 23:10	3049193	5674356	2364935
B4	644633	645632	20/08/2020 23:22 28/08/2020 00:07	3001185	5656050	2325491

Com o objetivo de diminuir a complexidade dos grafos gerados, cada instância foi dividida em suas componentes fortemente conexas utilizando o algoritmo de Kosaraju-Sharir. Na Tabela 6.4 são exibidos as características para as componentes fortemente conexas encontradas. Foram consideradas apenas as componentes formadas por mais que um vértice. A coluna $|V|_{\texttt{médio}}$ apresenta o número médio de vértices de cada componente, $|V|_{\texttt{max}}$ o valor máximo de vértices e $\sigma(|V|)$ o desvio padrão. De maneira similar, $|E|_{\texttt{médio}}$, $|E|_{\texttt{max}}$ e $\sigma(|E|)$ apresentam as mesmas métricas para o número de arestas. Notavelmente, cada uma das instâncias possui uma componente fortemente conexa com mais de 600000 vértices, agrupando a maioria das transações.

Por fim, para cada uma das instâncias extraídas foi executado os algoritmos `B&B` e `LocB`. Com o objetivo de considerar o uso de serviços de *mixing*, o modelo utiliza a restrição de conservação de saldo expandida. Os parâmetros utilizados foram uma

[5] `https://bitcoinfog.info`
[6] `https://cryptomixer.io`
[7] `https://mixtum.io`
[8] `https://bitmix.biz`

Tabela 6.4: Características das componentes fortemente conexas.

Instância	Componentes	$\lvert V\rvert_{\texttt{médio}}$	$\lvert V\rvert_{\texttt{max}}$	$\sigma(\lvert V\rvert)$	$\lvert E\rvert_{\texttt{médio}}$	$\lvert E\rvert_{\texttt{max}}$	$\sigma(\lvert E\rvert)$
B1	2191	312,97	678369	452,71	930,58	2029061	1354,12
B2	2015	321,71	641309	432,06	939,90	1884677	1269,76
B3	2084	328,33	676882	439,85	948,25	1966529	1277,89
B4	2117	319,17	667201	437,21	918,24	1930657	1265,16

taxa percentual de $p_m = 0,05$ do valor da transação e uma taxa fixa de $f_m = 0,0007$ BTC:

$$0,95 \sum_{k \in E'^{-}(v_i)} w_k - 0,0007 \leq \sum_{j \in E'^{+}(v_i)} w_j \leq \sum_{k \in E'^{-}(v_i)} w_k \tag{6.2}$$

Desse modo, a restrição 6.2 resultante engloba os valores de taxa para todos os *mixers* analisados, considerando os valores de taxa da Tabela 6.2. De modo geral, a restrição define que o somatório dos valores que entram em um vértice v_i deve respeitar um limite inferior (correspondente ao lado esquerdo) que considera os descontos das taxas de *mixing*, limitado superiormente pelo caso em que ambos os somatórios de entrada e saída são iguais.

A Tabela 6.5 apresenta os resultados das execuções dos algoritmos `B&B` e `LocB` para cada instância. Além do tamanho do *nilcatenation* encontrado $\lvert E'\rvert_{\texttt{soma}}$, a tabela exibe o valor médio da solução $\lvert E'\rvert_{\texttt{médio}}$ para cada componente, $\lvert E'\rvert_{\texttt{max}}$ a maior solução para uma única componente, o desvio padrão $\sigma(\lvert E'\rvert)$ e o tempo total de execução $t_{\texttt{total}}$ em segundos. A execução dos algoritmos para cada componente fortemente conexa foi limitada em 300 segundos.

Por meio dos resultados encontrados, é possível perceber que ambos os algoritmos encontraram *nilcatenations* para todas as instâncias extraídas das transações reais do Bitcoin. Notalvemente, o algoritmo `LocB` encontra *nilcatenations* com maior dimensão em número de transações entre usuários quando comparado ao algoritmo `B&B`. Com efeito, o NCP é aplicável para encontrar estruturas possivelmente relacionadas à lavagem de dinheiro. Ainda assim, a presença de *nilcatenations* no grafo de usuários é apenas um indicativo de atividades ilícitas. Desse modo, é necessário ressaltar que tais estruturas devem ser analisadas mais profundamente e de preferência correlacionados com outras técnicas de análise do histórico de transações.

Tabela 6.5: Resultados da execução dos algoritmos `B&B` e `LocB` para as instâncias do Bitcoin.

Algoritmo `B&B`					
Instância	$\|E'\|_{\texttt{soma}}$	$\|E'\|_{\texttt{médio}}$	$\|E'\|_{\texttt{max}}$	$\sigma(\|E'\|)$	$t_{\texttt{total}}$(s)
B1	890	148,33	753	297,34	1038
B2	**3877**	553,85	3706	1390,63	2112
B3	1740	580	1728	994,20	1210
B4	1052	175,33	543	249,86	1022
Algoritmo `LocB`					
Instância	$\|E'\|_{\texttt{soma}}$	$\|E'\|_{\texttt{médio}}$	$\|E'\|_{\texttt{max}}$	$\sigma(\|E'\|)$	$t_{\texttt{total}}$(s)
B1	**896**	112	753	260,15	2321
B2	3876	553,71	3706	1390,70	2855
B3	**1754**	438,5	1738	866,33	1982
B4	**3455**	493,57	2939	1090,49	1933

Capítulo 7

Conclusão

Este trabalho abordou o NCP como um problema de otimização, relacionando problemas da literatura e introduzindo uma primeira formulação de programação linear inteira. Também foi introduzido um algoritmo para o pré-processamento de instâncias e foi proposta a utilização do algoritmo de *local branching* para resolver o NCP. Por fim, foram executados experimentos computacionais para analisar o comportamento dos algoritmos propostos.

Em paralelo, também foi sugerida a aplicação do NCP como uma ferramenta para encontrar possíveis estruturas relacionadas à lavagem de dinheiro em criptomoedas. Sob essa perspectiva, o trabalho apresentou experimentos computacionais em transações extraídas da rede de testes e de transações reais do Bitcoin, confirmando a viabilidade da aplicação do NCP nesse contexto.

A seguir, são discutidas as contribuições da presente dissertação e são indicados possíveis trabalhos futuros que podem contribuir para o estudo do NCP, seja sob a ótica de otimização combinatória ou de suas aplicações.

7.1 Contribuições

Sob uma perspectiva teórica, o NCP foi apresentado como um problema de decisão e foi introduzida uma nova prova de NP-completude, que utiliza apenas uma redução do problema de soma de subconjuntos ao NCP, simplificando a prova de Géraud et al. [2017] encontrada na literatura. Ademais, foi introduzida uma primeira formulação do NCP como um problema de programação linear inteira. Consequentemente, gerando implicações práticas para o projeto de algoritmos para resolver o NCP, já que existem técnicas consolidadas que podem ser utilizadas para resolver problemas de programação linear inteira.

O trabalho também propôs a utilização do algoritmo de *local branching* para resolver o NCP. De acordo com os resultados computacionais para a maioria das instâncias de teste, o algoritmo de *local branching* encontrou melhores soluções, quando comparado ao o método de *branch and bound* utilizando a formulação proposta. Portanto, foram encontrados resultados que justificam a sua utilização para resolver o NCP.

Para a execução dos experimentos computacionais foi necessária a criação de instâncias de teste, pois não existiam instâncias na literatura para avaliar o desempenho de algoritmos para o NCP. Desse modo, foi gerado um conjunto de grafos aleatórios, com características heterogêneas, definindo um *baseline* comparativo para futuros algoritmos e formulações.

Anteriormente na literatura, somente Géraud et al. [2017] haviam proposto um algoritmo para resolver o NCP, o qual é baseado na utilização do algoritmo de LLL. Dessa forma, o presente trabalho também analisou a aplicação do algoritmo proposto por Géraud et al. [2017]. Por meio de experimentos computacionais, foi confirmada a hipótese que o LLL não é aplicável para resolver o NCP, quando o objetivo é maximizar a cardinalidade do *nilcatenation* encontrado.

Por fim, foi proposta a ampliação da aplicação do NCP como uma ferramenta para encontrar possíveis estruturas relacionadas à lavagem de dinheiro no Bitcoin. A formulação de programação inteira foi expandida, a fim de considerar a utilização de serviços de *mixing* para dificultar a rastreabilidade das transações. Nesse sentido, foram executados experimentos computacionais na rede de testes e rede real do Bitcoin, indicando de fato que o NCP pode ser aplicado no contexto de detecção de estruturas relacionadas à lavagem de dinheiro.

7.2 Trabalhos Futuros

O conjunto de instâncias de testes introduzido por esse trabalho é pequeno e pode ser expandido, com o intuito de considerar outros tipos de instâncias. Preferencialmente, instâncias que simulem características dos grafos de usuários da rede real do Bitcoin, como por exemplo a presença de poucas mas grandes componentes fortemente conexas e de usuários com poucas transações.

Outro aspecto que pode ser melhorado é a relaxação linear do modelo de programação linear inteira proposto. Uma direção de estudo é definir restrições adicionais, com o objetivo de tornar a formulação mais apertada, gerando melhores valores para o `GAP` de integralidade. Também é importante a análise de outros métodos para resolver problemas de programação linear inteira, como por exemplo, programação por

restrições ou heurísticas de busca local que trabalhem com soluções inviáveis.

Em relação à aplicação do NCP para a detecção de lavagem de dinheiro em criptomoedas, um possível trabalho futuro é expandir as análises conduzidas neste trabalho para outras criptomoedas, para as quais seja viável modelar o histórico de transações como um grafo de usuários. E ainda, é notavel que a utilização heurística de geração de grafos é prejudicada por serviços de *mixing* que utilizam o protocolos baseados no CoinJoin ou que realizam a ofuscação de transações por meio de canais alternativos, que não realizam as transações dentro do Bitcoin. Portanto, é aconselhável a ampliação do presente estudo, aplicando heurísticas mais complexas e considerando de forma mais aprofundada as taxas de mineração e *mixing*.

Referências Bibliográficas

Akbari, S.; Daemi, A.; Hatami, O.; Javanmard, A. & Mehrabian, A. (2010). Zero-sum flows in regular graphs. *Graphs and Combinatorics*, 26:603–615.

Antonopoulos, A. M. (2014). *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st edição. ISBN 1449374042.

Bao, Z.; Wang, B.; Zhang, Y.; Wang, Q. & Shi, W. (2018). Lockcoin: a secure and privacy-preserving mix service for bitcoin anonymity. *CoRR*, abs/1811.04349.

Bitcoin Project (2020). Bitcoin developer guide. https://developer.bitcoin.org.

Bonneau, J.; Narayanan, A.; Miller, A.; Clark, J.; Kroll, J. A. & Felten, E. W. (2014). Mixcoin: Anonymity for bitcoin with accountable mixes. In Christin, N. & Safavi-Naini, R., editores, *Financial Cryptography and Data Security*, pp. 486--504, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. & Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edição. ISBN 0262033844, 9780262033848.

Fischetti, M. & Lodi, A. (2003). Local branching. *Mathematical Programming*, 98:23–47.

Fleder, M.; Kester, M. S. & Pillai, S. (2015). Bitcoin transaction graph analysis. *CoRR*, abs/1502.01657.

Foley, S.; Karlsen, J. & Putnins, T. (2019). Sex, drugs, and bitcoin: How much illegal activity is financed through cryptocurrencies? *Review of Financial Studies*, 32:1798–1853.

FPLLL development team (2016). fplll, a lattice reduction library.

Géraud, R.; Naccache, D. & Roşie, R. (2017). Twisting Lattice and Graph Techniques to Compress Transactional Ledgers. In *SecureComm 2017 - 13th EAI International Conference on Security and Privacy in Communication Networks*, pp. 1–20, Niagara Falls, Canada.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pp. 85--103. Springer US, Boston, MA.

Lenstra, A.; Lenstra, H. & Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261.

Maesa, D.; Marino, A. & Ricci, L. (2017). Data-driven analysis of bitcoin properties: exploiting the users graph. *International Journal of Data Science and Analytics*.

Maxwell, G. (2013). Coinjoin: Bitcoin privacy for the real world. https://bitcointalk.org/index.php?topic=279249.

Meiklejohn, S.; Pomarole, M.; Jordan, G.; Levchenko, K.; McCoy, D.; Voelker, G. M. & Savage, S. (2013). A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pp. 127--140, New York, NY, USA. ACM.

Möser, M.; Böhme, R. & Breuker, D. (2013). An inquiry into money laundering tools in the bitcoin ecosystem. In *2013 APWG eCrime Researchers Summit*, pp. 1–14.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.

Pan, Y. & Zhang, F. (2016). Solving low-density multiple subset sum problems with svp oracle. *Journal of Systems Science and Complexity*, 29:228–242.

Reid, F. & Harrigan, M. (2011). An analysis of anonymity in the bitcoin system. *Security and Privacy in Social Networks*, 3.

Ron, D. & Shamir, A. (2012). Quantitative analysis of the full bitcoin transaction graph.

Ruffing, T.; Moreno-Sanchez, P. & Kate, A. (2014). Coinshuffle: Practical decentralized coin mixing for bitcoin. In Kutyłowski, M. & Vaidya, J., editores, *Computer Security - ESORICS 2014*, pp. 345--364, Cham. Springer International Publishing.

Sharir, M. (1981). A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67 – 72. ISSN 0898-1221.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160.