Implementações de Algoritmos FPT para o problema do 3-*Hitting Set* utilizando *Clusters* e Grades Computacionais

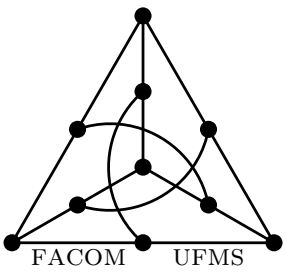
Rodrigo Cesar Sakamoto

Dissertação de Mestrado

Orientação: Prof. Dr. Henrique Mongelli

Área de Concentração: Ciência da Computação

Durante a elaboração desse trabalho o autor recebeu apoio financeiro da CAPES/FUNDECT.



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
Janeiro/2011

Implementações de Algoritmos FPT para o problema do 3-*Hitting Set* utilizando *Clusters* e Grades Computacionais

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por Rodrigo Cesar Sakamoto e aprovada pela comissão julgadora.

Campo Grande, 20 de Janeiro de 2011.

Banca Examinadora:

- Prof. Dr. Henrique Mongelli (orientador) (FACOM-UFMS)
- Prof. Dr. Siang Wun Song (IME-USP)
- Prof. Dr. Edson Norberto Cáceres (FACOM-UFMS)

aos meus pais Egidio e Deise

Agradecimentos

Primeiramente agradeço a Deus, por tornar tudo possível.

A minha família pelo apoio incondicional em todos os momentos da minha vida.

Ao meu orientador, professor Dr. Henrique Mongelli, pela dedicação, paciência, apoio, conselhos, e por ter acreditado em mim.

A todos os professores da Faculdade de Computação da UFMS, por contribuírem diretamente ou indiretamente, no desenvolvimento deste trabalho.

Aos meus amigos, pela amizade e apoio.

Resumo

Muitos problemas práticos são NP-Completos e envolvem um grande volume de dados. A busca por soluções exatas, aproximadas ou ótimas para muitos desses problemas resultaram em diversas técnicas engenhosas, visando, principalmente, a complexidade do problema em termos do tamanho da instância do problema. Uma abordagem alternativa para tentar lidar com a intratabilidade computacional de alguns problemas NP-Completos é a Complexidade Parametrizada. Os algoritmos tratáveis por parâmetro fixo, ou mais conhecidos como algoritmos FPT(*Fixed Parameter Tractability*), exploram a estrutura da instância do problema limitando a aparentemente inevitável explosão combinatorial na solução do problema (a um parâmetro). Neste trabalho será mostrado como combinar o paralelismo e algoritmos FPT, permitindo a utilização de instâncias ainda maiores na solução de problemas FPT. Mais precisamente, será apresentado um algoritmo FPT paralelo no modelo BSP/CGM para o problema do 3-*Hitting Set*, uma adaptação do algoritmo FPT paralelo de Cheetham *et al.*, sendo substituído suas fases, pelo algoritmo de Niedermeier e Rossmanith, e uma implementação de tal algoritmo.

Serão apresentados também dois algoritmos FPT: o algoritmo de Fernau e o algoritmo de Abu-Khzam. Estes algoritmos foram estudados e implementados, porém, seus resultados preliminares não se mostraram satisfatórios devido, principalmente, às estruturas de dados, sendo tais algoritmos descartados neste trabalho.

Palavras-chaves: tratabilidade por parâmetro fixo, algoritmo FPT paralelo, 3-*Hitting Set*.

Conteúdo

Capítulo 1

Introdução

Muitos dos problemas computacionais que aparecem na prática são problemas NP-Completos. Tentar resolvê-los pode implicar em uma explosão combinatorial no espaço de busca. Na tentativa de tentar lidar com a intratabilidade desses problemas, vários métodos têm sido desenvolvidos, como algoritmos de aproximação [5], heurísticas [6] e randomização [7]. Contudo, em aplicações práticas, muitos desses métodos não oferecem garantias quanto ao desempenho ou exatidão dos resultados. Por exemplo, a aproximação sacrifica a exatidão da solução para favorecer a computação eficiente de uma solução aproximada.

Apesar da incontestável importância dos algoritmos aproximados, o surgimento da Complexidade Parametrizada tem aberto um importante meio para tentar lidar com problemas intratáveis, onde, em algumas aplicações, são preferíveis algoritmos tratáveis por parâmetro fixo (FPT), que computam soluções exatas, em correspondência aos algoritmos aproximados. A Complexidade Parametrizada ainda oferece a possibilidade de desenvolvimento de algoritmos que resolvem problemas de complexidade exponencial, de forma eficiente para determinadas instâncias do problema que, em geral, são instâncias reais, na prática, e eram consideradas muito grandes para serem resolvidas com os métodos já existentes. Esse novo ramo da Teoria da Complexidade foi desenvolvida por Downey e Fellows em diversos artigos vistos em [8, 9, 10, 11, 12, 13].

O foco da Complexidade Parametrizada é procurar soluções exatas para problemas difíceis quando for possível fixar um parâmetro relativo ao tamanho da entrada. A idéia é fazer uma análise melhor da instância de entrada, dividindo-a em uma parte principal n (tamanho da "parte principal") e um parâmetro fixo k ("pequena parte" do problema) [8], de forma que a parte principal contribua polinomialmente no total da complexidade do problema, enquanto que a aparentemente inevitável explosão combinatorial fica limitada pelo parâmetro. Explorando tais parâmetros associados ao problema, muitos problemas intratáveis podem ser resolvidos eficientemente, ou seja, a Complexidade Parametrizada não visa mensurar a complexidade somente em termos do tamanho da entrada, mas também em termos do parâmetro. A suposição fundamental é que o parâmetro k seja muito menor que o tamanho da entrada ($k \ll n$) [14].

Diz-se que um problema parametrizável é tratável por parâmetro fixo ou FPT (*Fixed-Parameter Tractable*) quando existe um algoritmo que o resolve em tempo $O(f(k)n^{\alpha})$, em

que α é uma constante e f é uma função arbitrária [11]. Existem duas técnicas comumente aplicadas no desenvolvimento de algoritmos FPT [11]: a redução ao núcleo do problema e a árvore limitada de busca, as quais podem ser combinadas para resolver problemas FPT. A primeira técnica reduz o tamanho do espaço de busca e a segunda explora de forma inteligente, este novo espaço de busca. Além disso, existem novas técnicas [15, 16, 17] que vêm provendo bons resultados em termos de complexidade.

A complexidade exponencial associada ao parâmetro ainda pode resultar em custos proibitivos. A utilização do paralelismo, através da combinação dos modelos BSP (*Bulk Synchronous Parallel*) e CGM (*Coarsed Grained Multicomputer*), favorece o emprego de instâncias ainda maiores na solução de problemas FPT.

O modelo CGM, proposto por Dehne *et al.* [18], consiste em um conjunto de p processadores, cada um com memória local de tamanho $O(n/p)$, conectados por uma rede de interconexão, onde n é o tamanho do problema e $n/p \geq p$. Este modelo é uma simplificação do modelo BSP proposto por Valiant [19], que também é um modelo dito realístico, pois define parâmetros para mapear as principais características de máquinas paralelas reais, ou seja, considerando, dentre outras coisas, o tempo de comunicação entre os processadores.

Um algoritmo BSP/CGM possui rodadas de computação local alternadas com rodadas de comunicação entre os processadores, em que cada processador envia e recebe, em cada rodada, $O(n/p)$ dados. O tempo de execução de um algoritmo BSP/CGM é a soma dos tempos gastos tanto com computação local quanto com comunicações entre os processadores. Nas rodadas de computação local, geralmente utiliza-se o melhor algoritmo sequencial para o processamento.

As implementações em máquinas paralelas reais dos algoritmos projetados no modelo BSP/CGM têm obtido tempos bastante próximos aos previstos no modelo. Além disso, a combinação do paralelismo e de algoritmos FPT tem se mostrado profícua na obtenção de soluções para problemas práticos, como é visto em [20, 21]. Neste trabalho foram estudados os algoritmos FPT que resolvem o problema do 3-*Hitting Set*, propondo algoritmos FPT paralelos para o problema do 3-*Hitting Set*.

No problema do 3-*Hitting Set* deseja-se computar o *hitting set* (cobertura por conjuntos) de tamanho no máximo k, formado por pelo menos um elemento de cada subconjunto de C, de um conjunto finito S, de tamanho no máximo 3. A generalização no tamanho d dos subconjuntos de C é chamado de d-*Hitting Set*. Tais problemas são NP-Completos [3, 11, 22]. Os problemas 3-*Hitting Set* e k-Cobertura por Vértices são importantes membros da classe de problemas Tratáveis por Parâmetro Fixo (FPT).

Existem diversas aplicações práticas para o problema do 3-*Hitting Set* que podem ser vistas em [13, 23, 24, 25, 26]. Uma das aplicações práticas para o problema do 3-*Hitting Set* pode ser encontrada na Área de Biologia Computacional, mais precisamente na combinação de diferentes árvores filogenéticas[13]. Uma solução para resolver os conflitos entre as diferentes árvores é excluir algumas espécies. O conflito entre as diferentes estruturas de árvores pode ser modelado como um conflito triplo e o objetivo é remover o menor número de espécies em uma ordem para evitar todos os conflitos na estrutura da árvore.

O problema do 3-*Hitting Set* possui um algoritmo FPT conhecido para resolvê-lo de forma eficiente, desenvolvido por Niedermeier e Rossmanith [1], descrito na Seção 4.4, cuja complexidade de tempo $O(2.270^k k^3 + n)$, em que n é o tamanho da entrada e k é o tamanho máximo desejado para o *hitting set*. Aplicando a técnica de intercalação [15] das técnicas de redução ao núcleo do problema e árvore limitada de busca têm-se a complexidade de tempo $O(2.270^k + n)$.

O algoritmo FPT paralelo de Cheetham *et al.* [2] no modelo BSP/CGM (Seção 4.5), paraleliza ambas as fases do algoritmo FPT sequencial, redução ao núcleo do problema e árvore limitada de busca. Foram realizadas adaptações no algoritmo de Cheetham *et al.* [2] para resolver o problema do 3-*Hitting Set*, substituindo suas fases, pelo algoritmo de Niedermeier e Rossmanith [1].

Os algoritmos FPT descritos foram implementados na linguagem C/C++ utilizando os ambientes computacionais: Clusters e Grades Computacionais (InteGrade [27]).

O InteGrade é um *middleware* de grade [27] que pode utilizar as bibliotecas de comunicação BSP-Lib e MPI. Este *middleware* é um projeto desenvolvido por cinco instituições: Departamento de Ciência da Computação (IME-USP), Departamento de Informática (PUC-RIO), Departamento de Informática (UFMA), Instituto de Informática (UFG) e Faculdade de Computação (FACOM-UFMS)

No Capítulo 2 é descrito o modelo BSP/CGM ao qual os algoritmos FPT paralelos implementados foram desenvolvidos.

No Capítulo 3 serão vistos alguns conceitos relativos às classes de Complexidade Computacional e Complexidade Parametrizada, necessários para descrever o problema do 3-*Hitting Set*. Na seção sobre Complexidade Computacional serão apresentadas as definições de problemas tratáveis e intratáveis, bem como as classes de problemas P, NP e NP-Completo, e a definição do problema do *Hitting Set*. Na seção sobre Complexidade Parametrizada serão mostrados os conceitos relativos aos problemas tratáveis por parâmetro fixo e a descrição do problema do 3-*Hitting Set*, versão parametrizada.

No Capítulo 4 serão descritos os algoritmos FPT sequenciais de Niedermeier e Rossmanith [1], de Fernau [3] e de Abu-Khzam [4] e um algoritmo FPT paralelo, ambos para o problema do 3-*Hitting Set*, e os conceitos básicos necessários para o desenvolvimento de tais algoritmos.

No Capítulo 5 serão discutidos os detalhes da implementação do algoritmo de Niedermeier e Rossmanith [1] e as estruturas de dados utilizadas pela implementação.

No Capítulo 6 serão apresentados os resultados experimentais obtidos pela implementação e as comparações dos resultados experimentais em diferentes ambientes computacionais.

Por fim, no Capítulo 7 serão apresentadas as conclusões, as dificuldades encontradas e as sugestões para trabalhos futuros.

Capítulo 2

Modelos

2.1 Introdução

Um modelo de computação é uma representação abstrata de um sistema computacional, considerando somente os aspectos lógicos de funcionalidades mais importantes do sistema, direcionando o desenvolvimento de um algoritmo. Em um modelo de computação paralela, deve-se considerar a complexidade inerente do paralelismo, de forma que o modelo abstraia os detalhes dos sistemas paralelos, tornando a análise e projeto de algoritmos mais simples possível e portável para diversas plataformas. Os modelos realísticos são os adequados, pois consideram o ambiente computacional, separando os aspectos da arquitetura e da lógica matemática envolvidas, buscando estabelecer padrões aceitos que reflitam as dificuldades encontradas no desenvolvimento da computação paralela.

Na Seção 2.2 é apresentado o modelo realístico de computação paralela BSP, utilizado para desenvolver as versões paralelas implementadas de alguns algoritmos FPT. Finalmente, na Seção 2.3 é descrito o modelo realístico de computação paralela CGM, utilizado nas versões paralelas de algoritmos FPT que foram implementadas neste modelo.

2.2 Modelo BSP

O modelo BSP (*Bulk Synchronous Parallel Model*) foi proposto por Valiant [19], em 1990, como um modelo de computação paralela realística que não dependesse explicitamente da arquitetura do computador, mas que fosse uma ponte entre as necessidades de *hardware* e *software* na computação paralela. Além de ser um dos primeiros a considerar os custos de comunicação no modelo de computação paralela, o modelo define alguns parâmetros para facilitar a sincronização entre os processadores.

Uma máquina BSP consiste de um conjunto de p processadores, cada um com memória local, comunicando-se através de algum meio de interconexão, gerenciados por um roteador que possibilite a troca de mensagens ponto-a-ponto entre pares de processadores e com facilidade de sincronização global.

Um algoritmo BSP consiste de uma sequência de superpassos separados por barreiras de sincronização, como mostra a Figura 2.1. Em um superpasso, a cada processador é atribuído um conjunto de operações independentes, consistindo de uma combinação de passos de computação, usando dados disponibilizados localmente no início do superpasso, e passos de comunicação, através de instruções de envio e recebimento de mensagens. No final de cada superpasso, todos os processadores sincronizam. Cada processador verifica se sua tarefa obrigatória neste superpasso foi realizada. Os processadores esperam até que todos os outros tenham terminado.

Neste modelo, uma h-relação em um superpasso corresponde ao envio e/ou recebimento de, no máximo, h mensagens em cada processador. Uma mensagem enviada em um superpasso será recebida somente no próximo superpasso e não antes disso. O custo de um algoritmo BSP está atrelado ao número total de sincronizações, ao número total de superpassos e à própria complexidade do algoritmo.

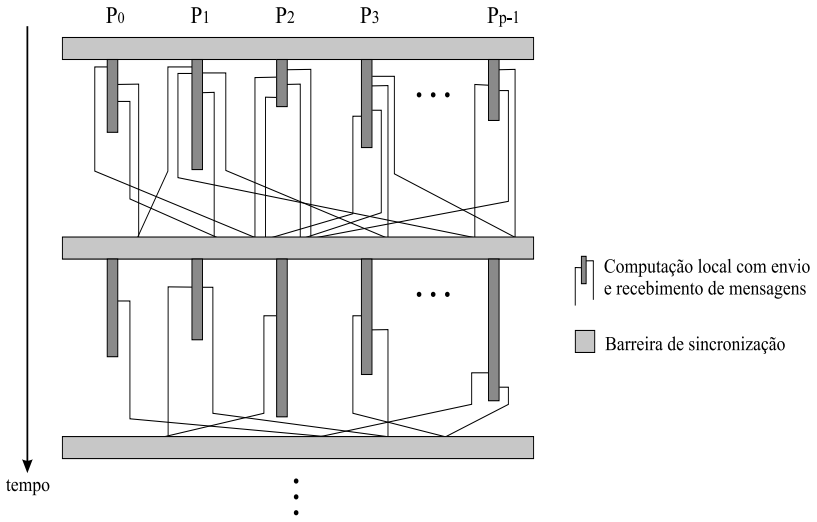


Figura 2.1: O Modelo BSP [28].

2.3 Modelo CGM

O modelo CGM (*Coarse Grained Multicomputer*) foi proposto por Dehne *et al.* [18], em 1993. Este modelo consiste de um conjunto de p processadores, cada um com memória local de tamanho $O(\frac{n}{p})$, onde n é o tamanho do problema e $n/p \geq p$. Os processadores comunicam-se através de uma rede de interconexão que permita a comunicação ponto-a-ponto.

O termo *coarse grained* (granularidade grossa) vem do fato do tamanho do problema ser consideravelmente maior que o número de processadores, ou seja, $\frac{n}{p} \gg p$. Essa restrição de comunicação de granularidade grossa, imposta ao modelo CGM, caracteriza um caso especial do modelo BSP, em que todas as operações de comunicação de um superpasso

são feitas em uma h-relação com $h \leq \frac{n}{p}$.

Em uma rodada de comunicação, uma h-relação (com $h = O(\frac{n}{p})$) é roteada, isto é, cada processador envia $O(\frac{n}{p})$ dados e recebe $O(\frac{n}{p})$ dados.

Um algoritmo CGM consiste uma uma sequência de rodadas (*rounds*), alternando fases bem definidas de computação local e comunicação global, separadas por barreiras de sincronização, como mostra a Figura 2.2. Geralmente, durante uma rodada de computação é utilizado o melhor algoritmo sequencial para o processamento dos dados armazenados localmente. O objetivo é diminuir o número total de rodadas de comunicação. No modelo CGM, o custo de comunicação é modelado pelo número total de rodadas de comunicação. O custo de um algoritmo CGM é a soma dos tempos obtidos em termos do número total de rodadas de computação local e o número total de rodadas de comunicação.

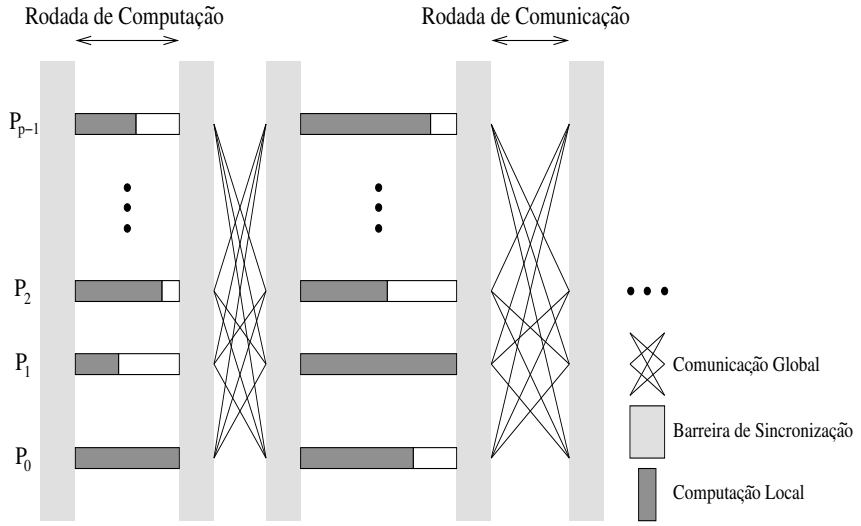


Figura 2.2: O Modelo CGM [29].

Capítulo 3

Intratabilidade, Algoritmos FPT e a Teoria dos Grafos

3.1 Introdução

Neste capítulo serão apresentados os conceitos necessários para o desenvolvimento dos algoritmos FPT, que resolvem o problema do 3-*Hitting Set*, e a sua respectiva definição. Na Seção 3.2 são introduzidos os conceitos referentes à Complexidade Computacional e a descrição do problema do *Hitting Set*. Na Seção 3.3 serão abordados os conceitos relativos aos algoritmos FPT, bem como os conceitos por trás destes algoritmos, e a descrição do problema FPT do 3-*Hitting Set*. Por fim, na Seção 3.4 serão definidos alguns conceitos relativo a hipergrafos.

3.2 Complexidade Computacional

A Teoria da Complexidade Computacional visa estudar o grau de dificuldade inerente aos problemas, estabelecendo os limites inferiores e superiores com relação à eficiência do algoritmo em resolver um problema, mensurada em termos de recursos computacionais necessários para que um algoritmo possa resolver um problema sob o ponto de vista computacional.

De uma maneira didática, com respeito à Teoria da Complexidade Computacional, pode-se agrupar os problemas computacionais em dois grupos [30]: os problemas solucionáveis e os problemas insolucionáveis. Os problemas para os quais não existem algoritmos para resolvê-los são chamados insolucionáveis ou indecidíveis; e os problemas solucionáveis ou decidíveis são aqueles que possuem algoritmos para resolvê-los. Observa-se que, apesar dos grandes avanços tecnológicos da computação nas últimas décadas, alguns problemas, embora possuam solução na teoria, não podem ser resolvidos na prática em tempo de execução aceitável, devido às exigências de tempo de processamento. Para exemplificar, será apresentado o problema do *Hitting Set* (Definição 3.1 e Figura 3.1).

Os problemas insolucionáveis não serão abordados neste trabalho, para maiores detalhes pode-se consultar Lewis e Papadimitriou [30].

Definição 3.1 *(Problema do Hitting Set* [1, 3, 4]*)*
Instância: *Dados uma coleção finita C de subconjuntos de um conjunto finito S e inteiro positivo $k \leq |S|$.*
Questão: *Existe um subconjunto $S' \subseteq S$, $|S'| \leq k$, tal que S' contém pelo menos um elemento de cada subconjunto em C?*

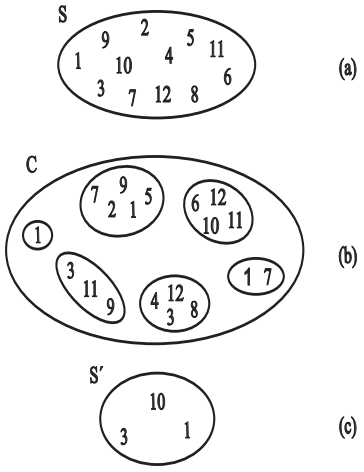


Figura 3.1: (a) Um conjunto S de elementos numerados de 1 a 12. (b) Uma coleção C de subconjuntos de um conjunto finito S. (c) O *hitting set* é o subconjunto $S' \subseteq S$, tal que S' contém pelo menos um elemento de cada subconjunto em C.

O problema do *Hitting Set* [1, 3, 4, 31] pode ser descrito em termos de hipergrafos, interpretando os conjuntos como hiperarestas e os elementos como vértices, e a tarefa consiste em determinar um subconjunto mínimo de vértices que cobre todas as hiperarestas. Este problema pode ser referido como problema de Cobertura por Vértices para hipergrafos. Exemplos de aplicações práticas podem ser vistas em [32, 33, 34].

Um algoritmo de força bruta para o problema do *Hitting Set* tentará testar todas as possíveis coberturas. Suponha-se que existe n conjuntos de tamanho no máximo d. O algoritmo testará todas as d possibilidades para um conjunto com d elementos. Logo, a complexidade de tempo do algoritmo é $O(d^n)$. Digamos que $d = 1000$ e $n = 100000$, poderia haver 1000^{100000} maneiras de cobrir todos os n conjuntos, tornando inviável na prática, pelo menos para as máquinas atuais.

Por isso, identificar a complexidade do problema pode definir meios matemáticos, para verificar se um problema é tratável ou intratável. Um problema que possua um algoritmo exato que resolva o problema de forma eficiente, isto é, em tempo polinomial $O(p(n))$ [35], onde $p(n)$ é um polinômio na variável n, é dito tratável [36]. Define-se algoritmo de tempo polinomial cujo tempo de execução no pior caso é $O(n^k)$ [36]. Além disso, o tempo de execução "aceitável" é uma imposição ao algoritmo de tempo polinomial. Os problemas que exigem tempo computacionalmente inviável, cuja a complexidade inferior

não seja polinomial, são considerados intratáveis [22] e formam a maioria dos problemas existentes.

Quanto à complexidade do problema, pode-se agrupá-los em várias classes de complexidade [36]: P, NP e NP-Completo. Para simplificar, serão considerados problemas de decisão aqueles que podem ser respondidos com "sim" ou "não"[37]. Um problema de decisão X é polinomialmente redutível a Y, se para cada instância I_X de X pode-se construir, em tempo polinomial, uma instância I_Y para Y, tal que I_X é uma instância para X com resposta "sim" se, e somente se, I_Y é uma instância para Y com resposta "sim".

A classe P consiste de problemas tratáveis que são resolvidos por algoritmos de tempo polinomial que resolvem todas as instâncias dos referidos problemas. A classe NP consiste de problemas de decisão para os quais existem algoritmos que verificam uma solução em tempo polinomial. É fácil verificar que os problemas da classe P estão contidas na classe NP, pois os problemas que estão em P possuem algoritmos que resolvem em tempo polinomial seus respectivos problemas, de forma que pode-se utilizar estes, para verificar a solução de um instância com resposta "sim" do problema. A classe NP-Completo pode ser vista como uma subclasse de problemas da classe NP, consistindo de problemas que estão na classe NP tais que qualquer outro problema na classe NP pode ser reduzido polinomialmente a algum problema da classe NP-Completo. Os problemas *Hitting Set* e Cobertura por Vértices são importantes membros da classe NP-Completo [22].

3.3 Complexidade Parametrizada

A Teoria da Complexidade Parametrizada [8, 9, 10, 11, 13, 14, 15, 38, 39, 40] surgiu como uma tentativa promissora para lidar com a intratabilidade de alguns problemas NP-Completos, fornecendo métodos para a solução exata sob certas condições. Para isso, deve-se analisar a estrutura da entrada, isolando alguns aspectos, de forma a fixar parâmetros [8]; confinar a aparentemente inevitável explosão combinatorial a este parâmetro. A parte principal da entrada contribui de forma polinomial para a complexidade total do problema. O objetivo principal é combinar ambas as partes, tanto a que contribui de forma polinomial como a que contribui de forma exponencial para a complexidade total do problema, possibilitando sua resolução; diferentemente da complexidade clássica onde a explosão combinatorial corresponde à toda entrada de um problema intratável. No entanto, é necessário limitar o parâmetro em um "pequeno intervalo" devido à Complexidade Computacional envolvida. Em muitas aplicações, o parâmetro pode ser considerado "bem pequeno" quando comparado ao tamanho da parte principal da entrada [12, 21], sendo que uma pequena faixa de valores de parâmetro é realmente importante na prática.

Antes de apresentar os conceitos relativos à Complexidade Parametrizada, serão introduzidos alguns aspectos da Complexidade Computacional clássica e da Complexidade Parametrizada com relação à especificação da entrada de um problema (instância).

Na Complexidade Computacional clássica, os problemas são especificados pela entrada

do problema (ou instância) e pela questão a ser respondida. Na Complexidade Parametrizada existem três partes para especificar um problema: a entrada do problema (ou instância); os aspectos da entrada do problema que constituem o parâmetro; e a questão a ser respondida.

A base do desenvolvimento da Complexidade Parametrizada está nos problemas NP-Completos como Cobertura por Vértices e Conjunto Independente, ambos para grafos não-dirigidos. Para facilitar a compreensão da Complexidade Parametrizada, o escopo será restrito a problemas de decisão e o parâmetro deverá ser uma função com relação à "entrada básica". Um problema parametrizável pode ser definido como descrito na Definição 3.2.

Definição 3.2 *(Problema Parametrizável* [8, 17]*) Um problema parametrizável é um conjunto* $L \subseteq \sum^* \times \sum^*$ *onde* $\sum$ *é um alfabeto fixo.*

Um melhor entendimento dos conceitos introduzidos anteriormente podem ser exemplificados. A seguir, nas Definições 3.3 e 3.4, são apresentados os problemas NP-Completos parametrizados: Cobertura por Vértices [10] e Conjunto Independente [11]. Tais problemas parametrizados são também conhecidos, respectivamente, como k-Cobertura por Vértices e k-Conjunto Independente. Em ambos problemas, a parte principal é um grafo e o parâmetro é um inteiro positivo.

Definição 3.3 *(Problema do k-Cobertura por Vértices(*[10]*))*
Instância: *Grafo* $G = (V, E)$.
Instância: *Inteiro positivo* k.
Questão: *Existe um conjunto de vértices* $V' \subseteq V$*, de tamanho máximo* k*, tal que para cada aresta* $(u, v) \in E$*, ou* $u \in V'$ *ou* $v \in V'$*?*

Definição 3.4 *(Problema do k-Conjunto Indenpedente(*[11]*))*
Instância: *Grafo* $G = (V, E)$.
Instância: *Inteiro positivo* k.
Questão: *Existe um conjunto de vértices* $V' \subseteq V$*, de tamanho máximo* k*, tal que não existem dois vértices adjacentes contidos em* V'*?*

O algoritmo FPT mais conhecido para resolver o problema da k-Cobertura por Vértices é o algoritmo de Balasubramanian *et al.* [41] cuja a complexidade de tempo é $O(kn + 1.324718^k k^2)$ e o melhor algoritmo para o problema do Conjunto Independente é devido a Robson [42] cuja complexidade de tempo é $O(1.21^n)$.

Os problemas tratáveis por parâmetro fixo formam a classe de problemas denominada FPT (*Fixed-Parameter Tractability*). Diz-se que um problema parametrizável é tratável por parâmetro fixo ou FPT quando existe um algoritmo que o resolve em tempo $O(f(k)n^\alpha)$, em que α é uma constante e f é uma função arbitrária [11]. Formalmente, temos a definição a seguir.

Definição 3.5 *(Problema Tratável por Parâmetro Fixo* [8, 11]*) Um problema parametrizável L é tratável por parâmetro fixo se existe um algoritmo FPT que determina se $(x, y) \in L$ em tempo $O(f(k)n^{\alpha})$, onde $f : N \to N$ é uma função arbritária nos inteiros não-negativos; n é o tamanho da parte principal da entrada x, ou seja, $|x| = n$; k é o tamanho do parâmetro y, ou seja, $|y| = k$; e α é uma constante independente de n e k.*

Os algoritmos que resolvem os problemas da classe FPT são chamados de algoritmos FPT, pois os problemas podem ser resolvidos eficientemente para valores pequenos do parâmetro fixo. Logo, o fato de um problema ser tratável por parâmetro fixo não significa, necessariamente, que exista um algoritmo FPT eficiente. Por exemplo, poderia ter um problema FPT com um algoritmo FPT para resolvê-lo de complexidade de tempo $O(f(k)n^{\alpha})$ tal que a função arbritária fosse $f(k) = 2^{2^{2^{2^k}}}$. Consequentemente, inviável na prática.

Assim, para incluir os problemas aparentemente intratáveis por parâmetro fixo, Downey e Fellows[38] definiram em uma hierarquia das classes de complexidade com a hierarquia definida como W, de forma que FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq$... $\subseteq$ W[P]. Tal sequência é comumente chamada de w-hierarquia. A classe $W[t]$ pode ser descrito em termos de problemas que são completos para eles, ou seja, um problema D é completo para uma classe de complexidade $\mathcal{E}$ se $D \in \mathcal{E}$ e todo problema nesta classe pode ser reduzido para D [43]. A classe $W[1]$ é análoga à classe NP da Complexidade Computacional clássica [13]. Uma lista com diversos problemas NP-Completos podem ser encontrados em [22] e parte desses problemas são tratáveis por parâmetro fixo, podendo ser encontrados em [11].

Agora serão apresentadas algumas técnicas computacionais engenhosas e amplamente utilizadas para o desenvolvimento de algoritmos FPT que resolvem problemas tratáveis por parâmetro fixo [11, 15]. Essas técnicas elementares, que na prática levam a algoritmos exponenciais, no pior caso, a obter melhores tempos de execução, mas a complexidade permanece inalterada.

- Redução ao núcleo do problema. Esta técnica reduz um problema de instância I com parâmetro k, transformando I em tempo polinomial em uma nova instância I' (núcleo do problema) com parâmetro k' tal que o tamanho de I' é limitado por uma função que depende somente de k', $k' < k$, e (I, k) tem uma solução se, e somente se, (I', k') tem uma solução. A idéia é reduzir uma instância do problema através de um conjunto de regras distintas, na qual são aplicadas repetidamente de forma que a aplicação das regras de redução possa ser feita em tempo polinomial, resultando em um nova instância limitada por um parâmetro que, possivelmente, deverá limitar o tamanho da árvore limitada de busca.

  Os algoritmos de redução ao núcleo do problema são típicos de problemas FPT. Downey, Fellows e Stege [13] mostraram que um problema é tratável por parâmetro fixo se, e somente se, é redutível ao núcleo do problema [13]. O uso desta técnica resulta em uma conexão aditiva entre as contribuições $f(k)$ e n^k, ou seja, $O(f(k) + n^k)$ [44]. Tal conexão aditiva não altera a complexidade de um algoritmo FPT tão pouco a Definição 3.5. Apesar disso, existem muitos problemas FPT, para os

quais ainda faltam algoritmos de redução ao núcleo do problema, por exemplo, os problemas de Duplicação de Genes [45], Inconsistência de Quarteto Mínimo [46] e Mediana do *Breakpoint* [47]. Fellows *et al.* [48] propuseram um novo método o qual permite mostrar que muitos problemas não têm núcleo do problema de tamanho polinomial com respeito à Teoria da Complexidade. A importância do método de redução ao núcleo do problema pode ser visto em alguns algoritmos FPT: o algoritmo de Niedermeier e Rossmanith [1], o algoritmo de Fernau [3], o algoritmo de Abu-Khzam [4] e o algoritmo de Balasubramanian *et al.* [41].

- Árvore limitada de busca. O objetivo é computar uma árvore de tamanho exponencial de forma exaustiva, em que o tamanho da árvore é limitado em uma função do parâmetro k. O parâmetro é usado para o corte da ramificação da árvore de busca, mantendo o tamanho da árvore gerenciável e facilitando a aplicação eficiente das regras de ramificações em cada nó da árvore.

  A construção da árvore limitada de busca utiliza um algoritmo que ramifica cada nó da árvore em vários ramos de forma eficiente, armazenando uma instância do problema, e pode realizar algum processamento nos nós da árvore, em busca de uma possível solução. Geralmente, o nó da raiz armazena a instância resultante da fase de redução ao núcleo do problema. A busca em profundidade é preferível na árvore limitada devido ao espaço de busca que pode ser proporcional à altura da árvore. Com isso, pode-se ter o espaço de busca limitada por k, consequentemente, o algoritmo poderá ser eficiente para esse k.

  Algoritmos FPT baseados nas técnicas de redução ao núcleo do problema e árvore limitada de busca têm complexidade de tempo $O(t(k)p(k) + q(n,k))$, onde $t(k)$ é o tamanho da árvore limitada de busca, $p(k)$ é o tamanho do núcleo do problema, e $q(n,k)$ é o tempo necessário para a redução ao núcleo do problema.

  Por exemplo, alguns algoritmos FPT que utilizam o método de árvore limitada de busca são o algoritmo de Niedermeier e Rossmanith [1], o algoritmo de Fernau [3], o algoritmo de Balasubramanian *et al.* [41] e o algoritmo de Stege [45].

- Intercalação das técnicas da árvore limitada de busca e de redução ao núcleo do problema (*Re-kernelization* [15]). Deve-se definir algunas condições para que esta técnica possa ser aplicada. Suponha-se que o algoritmo FPT trabalhe em dois estágios: redução ao núcleo do problema e árvore limitada de busca. Seja a redução ao núcleo do problema utilizando $P(|I|)$ etapas e resultando em uma instância de tamanho no máximo $q(k)$, em que ambos P e q são polinomialmente limitados. A expansão de um nó na árvore limitada de busca gasta $R(|I|)$ etapas, em que R é também limitada por algum polinômio. O tamanho da árvore limitada de busca é limitada por $O(\xi^k)$. A complexidade de tempo do algoritmo é então

  $$O(P(|I|) + R(q(k)).\xi^k),$$

  com (I,k) sendo uma instância a ser resolvida.

  A intercalação procura combinar as técnicas de redução ao núcleo do problema e árvore limitada de busca, de forma a aplicar a etapa de redução ao núcleo do problema em alguma etapa da árvore limitada de busca, resultando em uma instância

menor do problema. Para expandir um nó em uma árvore de busca rotulada pela instância (I, k), verifica-se primeiro se $|I| > c.q(k)$, onde $c \geq 1$ é uma constante que pode ser escolhida com o objetivo de favorecer a otimização do tempo de execução, então aplica-se a redução ao núcleo do problema para obter uma instância (I', k'), com $|I'| \leq q(k)$, na qual será utilizado no lugar de (I, k). A complexidade de tempo aplicando esta técnica é

$$O(P(|I|) + \xi^k).$$

Por exemplo, alguns algoritmos FPT que utilizam o método de intercalação das técnicas são o algoritmo de Niedermeier e Rossmanith [1] e o algoritmo de Fernau [3].

3.3.1 O Problema do 3-*Hitting Set*

A versão parametrizada para o problema do 3-*Hitting Set* [1, 3, 4] é um problema de otimização combinatorial, pertencente à classe NP-Completo [22], cujo o objetivo é computar um *hitting set* de tamanho no máximo k, tal que o *hitting set* é formado por pelo menos um elemento de cada subconjunto da coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3, conforme a Definição 3.6.

Definição 3.6 *(Problema do* 3*-Hitting Set* [1, 3, 4]*)*
Instância: *Dados um conjunto finito S, uma coleção finita C de subconjuntos de tamanho no máximo 3 de S.*
Parâmetro: *Inteiro positivo k.*
Questão: *Existe um subconjunto $S' \subseteq S$, de tamanho máximo k, tal que S' contém pelo menos um elemento de cada subconjunto em C?*

O algoritmo FPT mais conhecido para o problema do 3-*Hitting Set* é o algoritmo de Niedermeier e Rossmanith [1] de complexidade de tempo $O(2.270^k k^3 + n)$ e o algoritmo mais eficiente para o 3-*Hitting Set*, é o algoritmo de Wahlström [31] cuja a complexidade de tempo é $O(p(n) * 2.0775^k)$ para um polinômio p com n variáveis.

O algoritmo de Niedermeier e Rossmanith [1] requer um número relativamente pequeno de casos distintos, tornando menos complexo sua implementação e análise. Com isso, é de se esperar, quanto à sua eficiência e competitividade com relação aos algoritmos FPT que resolvem o problema da k-Cobertura por Vértices.

Niedermeier e Rossmanith [1] desenvolveram o primeiro método de redução ao núcleo do problema para o problema do 3-*Hitting Set* tal que o tamanho do núcleo do problema é $O(k^3)$ elementos. Além disso, existem outros trabalhos que visam melhorar a eficiência dos algoritmos FPT que resolvem o problema do 3-*Hitting Set*, entre eles estão os trabalhos de Xuan Cai [49], Faisal Nabih Abu-Khzam [4], Henning Fernau [3] e David Bryant [26].

1. Xuan Cai [49] estabeleceu os limites inferiores e superiores no tamanho do núcleo do problema para o problema restrito do 3-*Hitting Set* em hipergrafo 3-uniforme, pela dualidade paramétrica, mostrando que o 3-*Hitting Set* admite núcleo do problema de tamanho linear.

2. Faisal Nabih Abu-Khzam estudou diversos métodos de redução ao núcleo do problema, entre eles, o método de redução ao núcleo do problema para o 3-*Hitting Set* [4], que é a combinação do método de redução *crown* com outras regras de redução. Tal método de redução ao núcleo do problema para o 3-*Hitting Set*, reduz uma instância do problema a uma nova instância equivalente, cujo o tamanho desta nova instância é no máximo $5k^2 + k$ elementos.

3. Henning Fernau [3] desenvolveu uma heurística que pode diminuir a complexidade de alguns algoritmos FPT, em especial, para o algoritmo FPT de Niedermeier e Rossmanith [1] para o problema do 3-*Hitting Set*, na qual, forneceu uma melhoria nos resultados de Niedermeier e Rossmanith [1]. O algoritmo de Fernau [3] possui complexidade de tempo $O(2.179^k)$. Basicamente, Henning Fernau realizou uma análise melhor do algoritmo da árvore limitada de busca, através da heurística *priorities*, utilizando um segundo parâmetro auxiliar, permitindo uma outra análise, menos complicada da árvore limitada de busca.

4. David Bryant [26] desenvolveu um algoritmo para o problema do 3-*Hitting Set*, o algoritmo de Bryant *et al.* [26] cuja a complexidade de tempo é $O(2.5616^k + n)$.

Observe que pode-se utilizar um algoritmo trivial para o problema do 3-*Hitting Set* que testa todas as possibilidades de cobertura para cada subconjunto, cuja complexidade de tempo é $O(3^k + n)$. O algoritmo utiliza uma árvore de busca de tamanho 3^k que pode ser obtida da seguinte forma: como cada subconjunto tem que ser coberto pelo *hitting set*, ramifica-se cada nó da árvore em três filhos, escolhendo pelo menos um dos três elementos de algum subconjunto que pertence a C. Assim, constrói-se uma árvore de busca de altura k e tamanho 3^k. A generalização para o *d-Hitting Set* tem a complexidade de tempo $O(d^k + n)$, em que d é o tamanho máximo dos subconjuntos, conforme a Definição 3.7. Sabe-se que o *d-Hitting Set* é $W[2]$-Completo [11]. Niedermeier e Rossmanith [3] fizeram uma melhoria neste algoritmo, obtendo um algoritmo de complexidade de tempo $O(\alpha^k + n)$ onde $\alpha = d - 1 + O(d^{-1})$.

Definição 3.7 *(Problema do d-Hitting Set* [1, 3, 4]*)*
Instância: *Um hipergrafo $G = (V, E)$ com arestas de grau limitada por d.*
Parâmetro: *Inteiro positivo k.*
Questão: *Existe um hitting set de tamanho no máximo k: $\exists C \subseteq V$ tal que $\forall e \in E$ $(C \cap e \neq \emptyset)$?*

O problema do *Hitting Set* (sem restrição no tamanho dos subconjuntos) possui uma relação direta com o problema da Cobertura por Vértices Mínima [50]. Porém, não há uma relação direta do 3-*Hitting Set* com a Cobertura por Vértices Mínima[3].

O 3-*Hitting Set* pode ser visto como o problema da Cobertura por Vértices para hipergrafos [3], interpretando os elementos de S como vértices e os subconjuntos de tamanho 3 como hiperarestas de forma que uma hiperaresta está conectada a três vértices ao invés de dois e a questão é determinar um subconjunto mínimo de vértices que cobre todas as arestas.

3.4 Conceitos Básicos da Teoria dos Grafos (Hipergrafos)

Um hipergrafo é uma generalização de um grafo em que as arestas são chamadas de hiperarestas e podem conectar qualquer (sub)conjunto de vértices, conforme a Definição 3.8. A Figura 3.2 ilustra um hipergrafo com hiperarestas não-dirigidos. A seguir são apresentadas algumas definições para hipergrafos, que serão utilizadas no desenvolvimento de alguns algoritmos FPT. Mais detalhes sobre hipergrafos podem ser vistos em [51].

Definição 3.8 *Um hipergrafo $G = (V, E)$ é definido por um par V e E, em que:*

1. *V é um conjunto não vazio de vértices.*
2. *E é um conjunto de hiperarestas, tal que, cada hiperaresta é um subconjunto de vértices de V.*

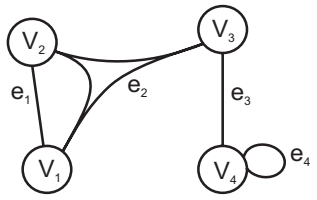


Figura 3.2: Um hipergrafo G=(V,E), com V=$\{V_1,V_2,V_3,V_4\}$ e E=$\{\{V_1,V_2\}, \{V_1,V_2,V_3\}, \{V_3,V_4\},\{V_4\}\}$.

Definição 3.9 *O grau de uma hiperaresta e é o número de vértices incidentes em e. Denota-se como $\delta(e)$.*

Definição 3.10 *O grau de um vértice v é número de arestas incidentes em v. Denota-se como $\delta(v)$.*

Definição 3.11 *Um hipergrafo é d-regular, se todos os vértices têm grau d.*

Capítulo 4

Algoritmos FPT para o 3-*Hitting Set*

4.1 Introdução

Neste capítulo são apresentados algoritmos FPT para o problema do 3-*Hitting Set* e que serão utilizados nas implementações. Estes algoritmos recebem como entrada uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k. Na Seção 4.2 é apresentado o algoritmo de Fernau [3], bem como os conceitos básicos utilizados no desenvolvimento do algoritmo. Na Seção 4.3 será abordado o algoritmo de Abu-Khzam [4] e alguns conceitos básicos necessários para a construção do algoritmo. Na Seção 4.4 será descrito o algoritmo FPT de Niedermeier e Rossmanith [1] e alguns conceitos básicos utilizados no desenvolvimento do algoritmo. Por fim, na Seção 4.5 será mostrado um algoritmo FPT paralelo para o problema do 3-*Hitting Set* construído a partir de uma adaptação do algorimo FPT paralelo de Cheetham *et al.* [2] para o problema da k-Cobertura por Vértices.

4.2 Conceitos Básicos do Algoritmo FPT de Henning Fernau

Neste trabalho, o algoritmo FPT de Fernau [3] no modelo *top-down* foi desenvolvido para resolver o problema do 3-*Hitting Set*, para uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k [1]. Para facilitar a introdução dos conceitos necessários para o desenvolvimento do algoritmo de Fernau [3] serão utilizados algumas terminologias da Teoria dos Grafos. Para isso, a instância (S, C, k) será interpretado como um hipergrafo em que os vértices e as arestas são os elementos de S e C, respectivamente.

Definição 4.1 *(Dominação de hiperaresta) Uma hiperaresta e é dominada por outra hiperaresta f, se $f \subset e$. Neste caso, remove-se a hiperaresta e.*

[1] As definições, lemas, corolários e teoremas foram retirados de [3].

Definição 4.2 *(Arestas tiny (auto-laço)) As arestas tiny são hiperarestas de grau 1 tal que os vértices que os constituem têm grau 1 e não estão conectados a outros vértices. Então, adiciona-se os correspodentes vértices ao hitting set e remove-se tais arestas.*

Definição 4.3 *(Dominação de vértice) Um vértice x é dominado por um outro vértice y se, para cada hiperaresta que contenha x também contém y. Então, pode-se simplesmente remover x do conjunto de vértices e todas as arestas que contém x.*

Definição 4.4 *Para um vértice v, $\delta^d(v)$ é o número de hiperarestas de grau d que contém v.*

Definição 4.5 *Instância "simples" é uma instância com vértices de grau no máximo 2.*

Primeiramente, deve-se observar que o algoritmo FPT de Fernau [3] para o 3-*Hitting Set* é um algoritmo simples, em comparação a outros algoritmos FPT para o problema do 3-*Hitting Set* (por exemplo, o algoritmo de Niedermeier e Rossmanith [1]), devido ao projeto *top-down* e, principalmente, à heurística *priorities*. Porém, requer uma análise mais sofisticada da árvore limitada de busca.

A heurística *priorities* permite realizar uma melhor análise do algoritmo da árvore limita de busca por meio de um segundo parâmetro auxiliar, auxiliando a prover uma melhoria, no caso em que o ganho em termos do parâmetro principal não é possível. No caso do algotimo FPT de Fernau [3] para o 3-*Hitting Set*, o parâmetro auxiliar é em função do número de arestas. Basicamente, prefere-se ramificar "escolhendo" arestas de "grau pequeno", conforme a Definição 4.6.

Definição 4.6 *Define-se a heurística priorities da seguinte maneira:*

1. *(H0) Preferimos "arestas pequenas". Mais fornalmente, seja $E_0 = \{e \in E \mid \delta(e) = 2\}$. Se $E_0 = \emptyset$ então $E_0 = E$.*
2. *(H1) Escolha algum $x \in V_i$ de grau máximo.*
3. *(H2) Maximize $V_i = \{x \in \bigcup_{e \in E_0} \mid \delta^3(x) - \delta^d(x)$ é máximo$\}$.*

Essa preferência por aresta de "grau pequeno" pode ser visto como uma "estratégia gulosa", pois ao escolher arestas de "grau pequeno" com vértice de grau máximo, favorece a ramificação da árvore, explorando a possibilidade de haver muitas arestas de "grau pequeno" no começo da execução do algoritmo e que serão introduzidas também durante a execução do algoritmo, pela remoção de algums vértices que reduzirá o grau da aresta de todas as arestas incidente em tais vértices. Assim, tal estratégia poderá cobrir mais arestas do que, provavelmente, se optasse por aresta de "grau alto". Além de gerar ramificações mais simples devido a combinação do projeto *top-down* e da heurística *priorities*, diminuindo a complexidade do algoritmo da árvore limitada de busca e favorecendo

aplicação das técnicas de podas em algoritmos de ramificação e poda, como é o caso do algoritmo da árvore limitada de busca.

O algoritmo também utiliza as duas últimas regras de redução de Niedermeier e Rossmanith [1] e todas as regras de redução utilizadas por Wahlström [31]. As regras de redução permitem simplificar as instâncias, favorecendo as aplicações de métodos como a heurística. Neste trabalho, as regras de redução (Definições 4.1, 4.2 e 4.3.) serão sempre aplicadas exaustivamente, no começo de cada chamada recursiva do algoritmo. Note que em instância reduzida nenhuma das regras de redução será aplicada. As regras de redução utilizadas neste Capítulo poderão ser encontradas também em [1, 31].

Teorema 4.1 3-*Hitting Set pode ser resolvido em tempo* $O(2.1788^k + n^3)$. [2]

4.2.1 Algoritmo FPT de Henning Fernau

O algoritmo recebe uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k. Em cada nó da árvore limitada de busca, a escolha dos elementos a adicionar na cobertura parcial depende de uma busca em profundidade, considerando as escolhas dos elementos de acordo com a heurística *priorities*, que poderá levar a várias ramificações.

ALGORITMO 3-*Hitting Set*
Entrada: Uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3, um inteiro positivo k e um solução parcial CSP($=\emptyset$) .
Saída: Um subconjunto $S' \subseteq S$, $|S'| \leq k$, tal que S' contém pelo menos um elemento de cada subconjunto em C, se existir ou uma mensagem, se tal subconjunto não existir.

1. **enquanto** for possível reduzir **faça**
1.1. **enquanto** existir subconjuntos dominantes **faça**
Remova-os de C.
1.2. **enquanto** existir *singletons* ($\{x\}$) **faça**
$S' = S' \cup \{x\}; \langle C = C \setminus \{C_i\}\ \forall C_i\ /\ x \in C_i, k = k - 1\rangle$
1.3. **enquanto** existir elementos dominados **faça**
Remova-os de cada subconjunto de C.
2. **se** (S, C, k) é uma instância simples
Resolva em tempo polinomial e retorne a solução S'.
3. **senão**
Escolha um conjunto $C_i \in C$ tal que $x \in C_i$ e x tem grau máximo, de acordo com, a heurística *priorities*.
$S' = \emptyset$
$C' = C \setminus \{C_i\}\ \forall C_i\ /\ x \in C_i$
$S' =$ 3-*Hitting Set*$(C',\ S,\ k-1,$ CSP$\cup\{x\})$
4. **se** $S' = \emptyset$

[2] Uma análise da árvore limitada de busca pode ser encontrada com mais detalhes em [3].

$C'_i = C_i \setminus \{x\}\ \forall C_i\ /\ x \in C_i$
$S' =$ 3-*Hitting Set*$(C',\ S,\ k,$ CSP$)$
retorna S'

No Passo 1 aplicam-se as regras de redução exaustivamente antes de cada chamada recursiva, garantido pelo laço, cujo o tempo gasto nesta etapa é $O(n^3)$. Suponha-se que no começo de cada caso não existe nenhuma instância a ser reduzida, nesta fase são removidas da coleção C de subconjuntos, os conjuntos dominados e os elementos dominados; e removem-se também os *singletons* (conjunto de tamanho 1 ou arestas *tiny*), adicionando-os à cobertura por conjuntos parcial CSP e são removidos os elementos da coleção C de subconjuntos correspondentes aos elementos que estão na cobertura parcial e k é atualizado.

Após cada aplicação das regras de redução, no Passo 2, é verificado se existe instância "simples" na instância reduzida. Caso exista, utiliza-se o algoritmo de Norman e Rabin [52] para computar a cobertura por conjuntos cuja o tempo gasto nesta etapa é $O(n^4)$. Primeiramente, interpretam-se os elementos de grau 2 como conjuntos e os conjuntos como elementos. Em seguida é computado um emparelhamento máximo M [3] em C de S tal que quaisquer dois conjuntos de M não tenham elementos em comum. Então, para cada elemento x não coberto por M, escolha um conjunto C_i que contém x e adicione C_i a M'. A união $M \cup M'$ é a cobertura por conjuntos, S'.

Nos Passos 3 a 4 são realizadas buscas em profundidade na árvore limitada de busca com o objetivo de ramificar a árvore em função da heurística *priorities*, em busca de uma possível solução do problema. Um nó da árvore tem uma CSP com mais elementos e uma coleção com menos subconjuntos do que seu pai, visto que a cada nova ramificação um possível elemento x é adicionado à cobertura parcial, faz-se a remoção dos correspondentes subconjuntos da qual o elemento pertencia. Quando não é adicionado o elemento x à cobertura parcial, então novos subconjuntos de tamanho menores ("grau pequeno") são criados, pois, remove-se o elemento x de todos os subconjuntos que o contém e, não é atualizado o parâmetro k. Os elementos que são adicionados à cobertura podem cobrir um ou mais conjuntos, de acordo com as possibilidades de escolhas dos subconjuntos para serem analisados durante a busca em profundidade. Se o nó em questão tem uma instância com uma cobertura parcial de tamanho menor ou igual a k e com uma coleção C' resultante vazia, significa que é encontrado um *hitting set* de tamanho menor ou igual a k para a coleção C de S.

A ramificação de um nó resume-se a possibilidade do elemento x estar ou não na cobertura parcial, garantindo, assim, a geração de uma árvore limitada de busca binária. Por meio de uma análise de ramificação do comportamento da árvore, pode-se estimar o tamanho da árvore, $T(k) \leq 2.1788^k$ (Passo 3 a 4). Sabe-se que a complexidade do algoritmo de redução ao núcleo do problema é $O(n^3)$ (Passo 1). Do Teorema 4.1 tem-se que a complexidade de tempo do Algoritmo FPT de Fernau [3] é $O(2.1788^k + n^3)$. Uma análise detalhada da complexidade da árvore limitada de busca pode ser vista em [3].

[3] Utiliza-se o algoritmo de Edmond para um emparelhamento máximo para um grafo geral [53].

Exemplo de Execução do Algoritmo

Na Figura 4.1 é apresentada um exemplo de execução do algoritmo de Fernau [3]. Por questão didática e simplicidade será apresentada a coleção $C = \{\{V_1, V_2\}, \{V_1, V_4\}, \{V_2, V_3\}, \{V_3, V_4\}, \{V_3, V_5\}, \{V_4, V_5\}, \{V_5, V_6\}, \{V_5, V_7\}, \{V_6, V_7\}, \{V_6, V_8\}, \{V_6, V_{10}\}, \{V_7, V_8\}, \{V_8\}, \{V_8, V_{10}\}, \{V_8, V_9, V_{10}\}\}$ de subconjuntos de tamanho no máximo 3 de um conjunto finito $S = \{V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}\}$ na representação de hipergrafos, com $k = 5$, conforme apresentado na Figura 4.1(a). Note que será explicado todo o exemplo de execução do algoritmo na notação de conjuntos com figuras, utilizando a representação de hipergrafo para ilustrar a coleção C de S, em várias etapas. O objetivo é computar um *hitting set* S' com $|S'| \leq 5$ tal que $k \geq 0$ e $C = \emptyset$.

Inicia-se aplicando as regras de redução, conforme são vistos nas Figuras 4.1(b) e 4.1(c). Primeiramente, remove-se o conjunto $\{V_8, V_9, V_{10}\}$ de C, pela regra da dominação de conjuntos, conforme as Figuras 4.1(b) e 4.1(c). Em seguida, remove-se o conjunto $\{V_8\}$ de C, pela regra dos conjuntos de tamanho 1 e é adicionado o elemento V_8 ao *hitting set* S', $S' = \{V_8\}$ e $k = 4$. Além de remover todos os conjuntos que contém V_8, conforme ilustram as Figuras 4.1(c) e 4.1(d).

Note que não existe uma instância "simples", pois o elemento V_5 tem grau maior que 2, conforme visto na Figura 4.1(d). Então, inicia-se a construção da árvore limitada de busca T binária, escolhendo o conjunto $\{V_5, V_6\}$ de acordo com a heurística *priorities*, conforme é mostrado na Figura 4.1(e). Ramifica-se a árvore T, adicionando o elemento V_5 ao *hitting set* S', $S' = \{V_5, V_8\}$ e $k = 3$, e removendo todos os conjuntos que o contém.

Na próxima iteração do algoritmo, aplicam-se as regras de redução. Verificou-se que é possível aplicar a regra da dominação de elementos. Logo, remove-se o elemento V_7 de todos os conjuntos que o contém, gerando um conjunto de tamanho 1, $\{V_6\}$, que será coberto somente pelo elemento V_6. Então, é adicionado V_6 ao *hitting set S'*, $S' = \{V_5, V_6, V_8\}$ e $k = 2$.

Em seguida, verificou-se que é possível aplicar o algoritmo de Norman e Rabin [52], pois existe somente elementos de grau no máximo 2 ($\delta(V_1) = \delta(V_2) = \delta(V_3) = \delta(V_4) \leq 2$) em C de S, conforme mostra a Figura 4.1(e). O algoritmo adicionará os elementos V_1 e V_3 ao *hitting set* S', $S' = \{V_1, V_3, V_5, V_6, V_8\}$ e $k = 0$. Como a coleção C é vazia e $k = 0$, tem-se um *hitting set* S' válido de tamanho menor ou igual a 5. Portanto, retorna-se $S' = \{V_1, V_3, V_5, V_6, V_8\}$. Observe que se tivesse optado por não adicionar o elemento V_5 ao *hitting set* S', teria um *hitting set* S' tal que $|S'| > 5$, mais precisamente, $S' = \{V_1, V_3, V_4, V_6, V_7, V_8\}$, conforme a Figura 4.1(e).

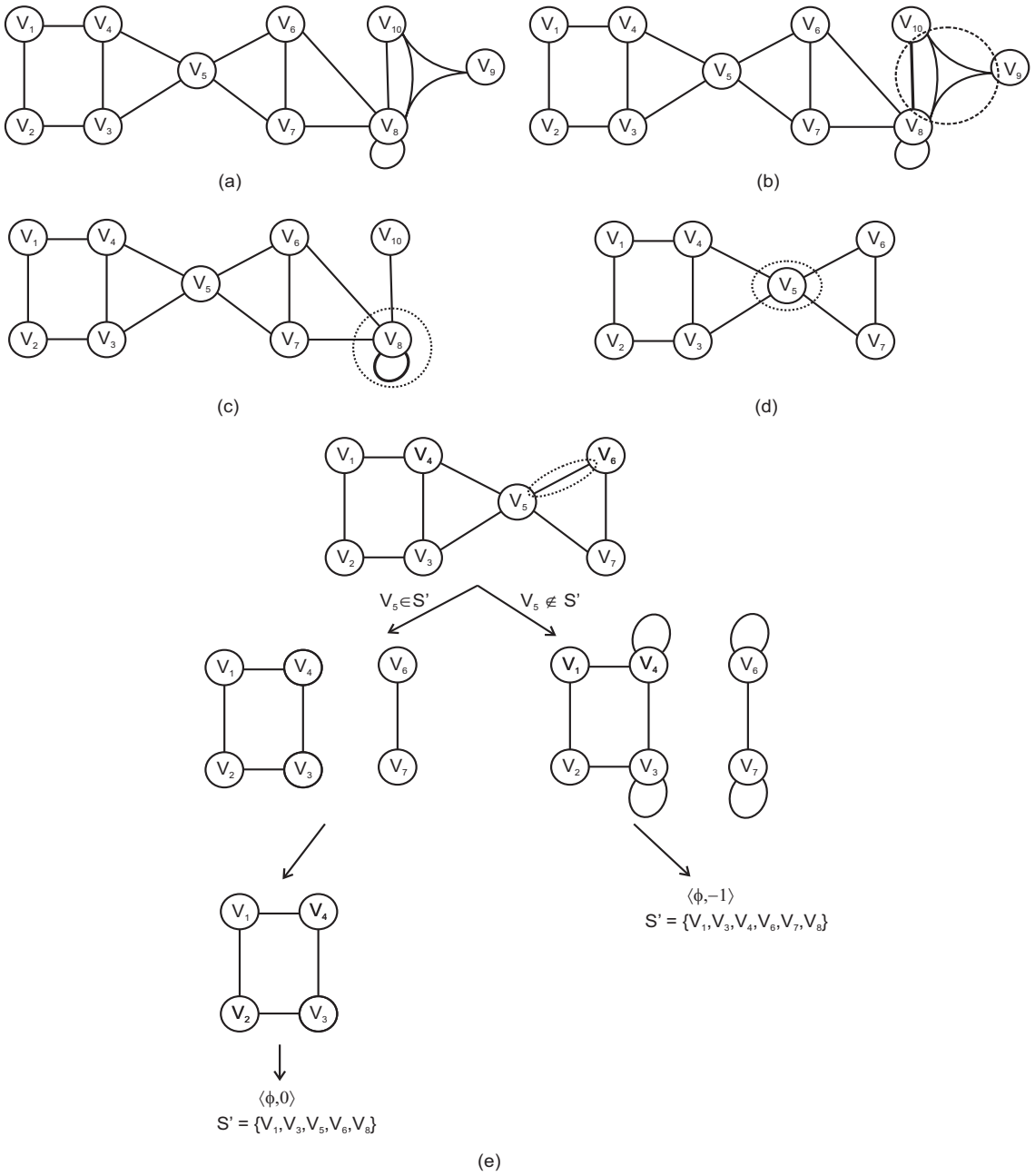


Figura 4.1: Exemplo de execução do Algoritmo de Fernau [3]. (a) Uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3. (b) e (c) Aplicação das regras de redução. (d) e (e) A árvore limitada de busca.

4.3 Conceitos Básicos do Algoritmo FPT de Abu-Khzam

O algoritmo FPT de Abu-Khzam pode resolver o problema do 3-*Hitting Set* utilizando uma nova técnica, redução *crown* (decomposição *crown* ou estrutura *crown*) [4]. A técnica de redução *crown* pode-se combinar com outras técnicas como emparelhamento em (hiper)grafo. Será mostrado a aplicação da técnica de redução *crown* por meio do algoritmo de redução ao núcleo do problema de Abu-Khzam [4] para o problema do 3-*Hitting Set*.

Neste trabalho, o algoritmo de Abu-Khzam [4] para o 3-*Hitting Set* foi projetado como um problema de conjuntos. Basta interpretar as hiperarestas como conjuntos (coleção C) e os vértices como elementos (conjunto S). Note que a técnica de redução *crown* pode ser aplicada também em problemas que não envolvam grafos (Frank Dehne *et al.* [16]). Mais adiante serão mostrados os passos necessários para que seja possível desenvolver o algoritmo de Abu-Khzam [4] para uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k.

Antes de introduzir a técnica de redução *crown* para o 3-*Hitting Set* serão apresentados alguns conceitos necessários para entendimento desta técnica e do algoritmo de Abu-Khzam [4]. Observe que os conceitos apresentados nesta seção serão apresentados para um problema em hipergrafo (3-*Hitting Set*), segue que os hipergrafos referenciados nesta seção serão definidos, como $G = (V, E)$.

Definição 4.7 *Um conjunto independente I de um hipergrafo é um conjunto de vértices que não têm arestas incidentes em dois vértices de I, no sentido em que dois deles não pertencem a um elemento de E.*

Definição 4.8 *$N_H(A)$ é o conjunto de vértices adjacentes de A em H, onde A é o conjunto de vértices e H é um grafo ou subgrafo.*

A seguir será definida a técnica de redução *crown* para o problema do 3-*Hitting Set* que é baseada em uma extensão da notação de *crown* para hipergrafo. Um *crown* em um hipergrafo consiste de um conjunto independente I e um conjunto H que contém todos os vértices adjacentes a I, conforme a Definição 4.9.

[4] As definições, lemas, corolários e teoremas foram retirados de [4].

Definição 4.9 *Um crown em um hipergrafo para o problema do* 3-*Hitting Set é definida pela tripla* (H, I, M) *tal que:*

- I *é um conjunto independente.*
- $H = \{A \subset V | A \cup \{x\} \in E$ *para algum* $x \in I\}$.
- M *é um emparelhamento do grafo bipartido* $(I \cup H, E)$ *onde* $E = \{(x, A) | x \in I$ *e* $A \in H$ *e* $(\{x\} \cup A) \in E\}$.
- *Todo elemento de* H *é emparelhado sobre* M.

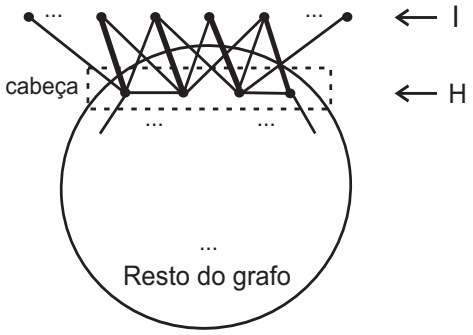


Figura 4.2: Uma decomposição *crown* de largura 4. As arestas em negrito são elementos de M [17].

Note que o emparelhamento M é induzido pelas arestas entre I e H tal que os elementos de H são todos cobertos por M. O conjunto H é chamado de coroa (cabeça) de um *crown* (H, I, M) e $|H|$ é a largura do *crown*. A Figura 4.2 exemplifica a Definição 4.9, ilustrando o principal resultado para o algoritmo de Abu-Khzam [4], que é a definição de uma regra de redução *crown* para o 3-*Hitting Set*, que consiste da remoção de todos os elementos de I, bem como as arestas incidentes em I e o rearranjo de E. Observe que regra de redução *crown* pode ser vista como uma generalização da regra dos vértices de grau 1, visto em [17].

Além disso, a regra de redução *crown* de Abu-Khzam [4] para o 3-*Hitting Set*, por si próprio, não pode ser vista como uma técnica de redução ao núcleo do problema, pois é combinada com as regras de redução: regra do vértice de grau alto (Definição 4.10) e regra dos vértices (ou arestas) de grau 1 ("*singletons*"), de forma a "auxiliar" a aplicação destas. Ambas, as regras de redução são utilizadas por Niedermeier e Rossmanith em [1] e serão aplicadas de forma combinada com a regra redução *crown* para o desenvolvimento do algoritmo FPT de redução ao núcleo do problema de Abu-Khzam [4]. Observe que a Definição 4.10 é uma generalização da regra de redução definida por Buss em [54].

Definição 4.10 *(Regra do vértice de grau alto) Se* $x \in V$ *tem mais que* k *arestas cuja a interseção entre os pares de arestas é* $\{x\}$*, então:*

- *Adicione* x *a qualquer solução potencial de* (V, E, k).

- *Remova todos os elementos de $E'(x)$.*
- *Decremente k.*

Lema 4.1 *Um grafo G tem uma decomposição crown de tamanho positivo se, e somente se, G tem um conjunto indepedente I que satisfaz $|N_G(I)| < |I|$.*

Definição 4.11 *Seja (V, E, k) uma instância do 3-Hitting Set. Dois elementos de E são fracamente conexos se eles não compartilham mais que um vértice em comum.*

Definição 4.12 *Seja (V, E, k) uma instância do 3-Hitting Set. Um subconjunto W de E é uma coleção maximal de arestas fracamente conexas se toda aresta de E está ou em W ou compartilham pelo menos dois vértices com pelo menos um elemento de W.*

Teorema 4.2 *3-Hitting Set tem um núcleo (kernel) com $5k^2 + k$ elementos.*

4.3.1 Algoritmo FPT de Abu-Khzam

O algoritmo FPT de Abu-Khzam [4] apresenta um método de redução ao núcleo do problema, utilizando a técnica de redução *crown* para o 3-*Hitting Set* combinado com algumas regras de redução utilizados por Niedermeier e Rossmanith em [1] e Buss em [54]. O algoritmo recebe uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k.

ALGORITMO Redução ao Núcleo do Problema para o 3-*Hitting Set*
Entrada: $\langle S, C, k\rangle$ onde C é uma coleção de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k.
Saída: Ou uma mensagem, se $\langle S, C\rangle$ não tem um *hitting set* de tamanho $\leq k$, ou uma instância $\langle S, C', k'\rangle$ e S' uma solução parcial com $k' = k - |S'|$ e $|S'| \leq 5k^2 + k$.

1. **enquanto** C contém *singletons* (arestas de grau 1) $\{x\}$
   $S' = S' \cup x$
   $C = C \setminus \{x\}$
   $k = k - 1$
2. Adicione os elementos (vértices) de grau maior que k em S' e atualize k.
3. Construa o conjunto maximal W de conjuntos (arestas) fracamente conexos.
4. **se** $|W| > k^2$
   retorna "Não existe um *hitting set* válido"
5. Seja I o conjunto independente e $H = \{y | \forall y \in C_i$ **e** $\forall x \in C_i$ **e** $x \in I$ **e** $x \neq y\}$ (grafo bipartido $G_{I,H}$).
6. **se** $|I| > |H|$
   Invoque *construct_crown* [23] [5] na coleção C' de subconjuntos de C,

[5] Utiliza-se o procedimento *construct_crown* descrito em [17].

induzidos pelos elementos entre I e H, para obter um *crown* (H', I', M)
Remova todos os elementos de I' de cada subconjunto de
$C(C = C \setminus \{C_i\}\ \forall C_i\ /\ x \in I')$.
7. **retorna** "A (possível) nova instância"

Nos Passos 1 a 2 aplicam-se as regras de redução. Primeiramente, são adicionados os elementos dos conjuntos de tamanho 1 (*singleton* ou aresta de grau 1) ao *hitting set* S', pois sabe-se que o conjunto de tamanho 1 poderá ser coberto somente pelo elemento que o contém. Além de decrementar o parâmetro k e remover todos os conjuntos incidentes em tais elementos. Em seguida, são adicionados os elementos de grau maior que k ao *hitting set* S', pois sabe-se que tais elementos pertencem a qualquer *hitting set*. Removem-se os conjuntos correspondentes aos elementos que estão em S', bem como os elementos que estão em S' e atualiza-se o parâmetro k.

No Passo 3 é construído a coleção maximal W de conjuntos (arestas fracamente conexas). A construção de W pode ser feita trivialmente, em tempo polinomial, escolhendo, primeiramente, os conjuntos de tamanho 1, e gulosamente são adicionados os elementos de C em W, suponha-se que os conjuntos de tamanho 2 serão adicionados antes de qualquer conjunto de tamanho 3. Para que, no Passo 4, seja possível verificar se existe um *hitting set* de tamanho menor ou igual a k, pois sabe-se que k elementos podem cobrir no máximo $k.k$ conjuntos de tamanho 2. Assim, se o tamanho do conjunto W for maior que $k.k$ significa que são necessários mais que k elementos para cobrir W. Sendo assim, não existe um *hitting set* válido de tamanho menor ou igual a k, para a coleção C de S.

Do Passo 5 em diante é aplicada a regra de redução *crown* que consiste da remoção de todos os elementos de I em C, bem como o rearranjo de C. Inicia-se construindo o conjunto independente I de elementos tal que dois elementos de I não compartilham nenhum elemento em comum na coleção C e o conjunto H de elementos que pertencem aos mesmos conjuntos das quais os elementos de I pertencem. Observe que foi trabalhado um problema de grafo como se fosse um problema de conjuntos.

Finalmente, no Passo 6, verifica-se se é possível aplicar a regra de redução *crown*. Com isso, garante-se a aplicação da regra de redução *crown*, utilizando o procedimento *construct_crown* que computa um *crown* (H', I', M) por meio dos conjuntos H e I. Assim, pode-se aplicar a regra de redução *crown* na instância (S, C, k), removendo todos os elementos de I' de todos os subconjuntos de C, bem como todos os conjuntos em C que contém elementos de I'. Independentemente do Passo 6, é retornado uma possível instância reduzida do problema (Passo 7) que poderá ser ou um *hitting set* de tamanho menor ou igual a k, ou mensagem, caso não exista uma solução de tamanho menor ou igual a k.

O tempo de execução do algoritmo é dominado pelo procedimento *construct_crown* que computa um *crown* (H, I, M) em tempo $O(n^{2.5})$, pois gasta-se $O(|C|)$ para cada uma das operações: remover os *singletons*, remover os elementos de grau maior que k e computar o conjunto maximal W. Portanto, o algoritmo tem complexidade de tempo $O(n^{2.5})$.

Exemplo de Execução do Algoritmo

Na Figura 4.3 é apresentada um exemplo de execução do algoritmo de Abu-Khzam [4]. Por questão didática e simplicidade, será utilizada a coleção $C = \{\{V_1, V_7\}, \{V_2, V_7\}, \{V_2, V_8\}, \{V_3, V_7\}, \{V_3, V_8\}, \{V_3, V_9\}, \{V_4, V_8\}, \{V_4, V_9\}, \{V_4, V_{10}\}, \{V_5, V_9\}, \{V_5, V_{10}\}, \{V_6, V_{10}\}, \{V_7, V_8\}, \{V_7, V_{11}\}, \{V_7, V_{12}\}, \{V_8, V_{12}\}, \{V_9, V_{10}\}, \{V_9, V_{12}\}, \{V_9, V_{13}\}, \{V_{10}, V_{12}\}, \{V_{10}, V_{14}\}, \{V_{10}, V_{20}\}, \{V_{11}\}, \{V_{12}, V_{13}\}, \{V_{12}, V_{14}\}, \{V_{12}, V_{15}\}, \{V_{12}, V_{16}\}, \{V_{13}, V_{14}\}, \{V_{14}, V_{16}\}, \{V_{15}, V_{16}\}, \{V_{16}, V_{17}\}, \{V_{17}, V_{18}, V_{19}\}\}$ de subconjuntos de tamanho no máximo 3 de um conjunto finito $S = \{V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}, V_{20}\}$ na representação de hipergrafo e um parâmetro $k = 7$, conforme a Figura 4.3(a) e a Figura 4.3(b). Note que será explicado todo o exemplo de execução do algoritmo na notação de conjuntos com figuras, utilizando a representação de hipergrafo para ilustrar a coleção C, em várias etapas. O objetivo é fazer a redução ao núcleo do problema para a coleção C de S de forma que, ao final da execução do algoritmo, seja possível ter uma solução parcial de tamanho menor ou igual a 7, ou seja $k \leq 7$, ou uma mensagem, caso não exista solução válida.

Nos Passos 1 a 2, aplicam-se as regras de redução (regra dos conjuntos de tamanho 1 e regra dos elementos de grau maior que k), conforme a Figura 4.3(b). Primeiramente, verifica se existem conjuntos de tamanho 1. Como existe somente o conjunto $\{V_{11}\}$ de tamanho 1 então é removido o elemento V_{11}, bem como todos os conjuntos que contém o elemento V_{11}, além de adicionar o elemento V_{11} ao *hitting set* S' e atualizar o parâmentro k, $k = 6$. Em seguida, verifica-se que existe somente o elemento V_{12} de grau maior que 6, então o elemento V_{12} é removido, bem como todos os conjuntos que o contem e, adiciona-se o elemento V_{12} ao *hitting set* S' e k é atualizado, $k = 5$.

Nos Passos 3 a 6 será aplicado a regra de redução *crown* de Abu-Khzam. No Passo 3 é computado o conjunto maximal $W = \{\{V_1, V_7\}, \{V_2, V_7\}, \{V_2, V_8\}, \{V_3, V_7\}, \{V_3, V_8\}, \{V_3, V_9\}, \{V_4, V_8\}, \{V_4, V_9\}, \{V_4, V_{10}\}, \{V_5, V_9\}, \{V_5, V_{10}\}, \{V_6, V_{10}\}, \{V_7, V_8\}, \{V_9, V_{10}\}, \{V_9, V_{13}\}, \{V_{10}, V_{14}\}, \{V_{10}, V_{20}\}, \{V_{13}, V_{14}\}, \{V_{14}, V_{16}\}, \{V_{15}, V_{16}\}, \{V_{16}, V_{17}\}\}$ de conjuntos fracamente conexo, ou seja, conjuntos de tamanho 2 em C, conforme a Figura 4.3(c). Como $|W| < 5^2$ então pode-se ter um *hitting set* de tamanho menor ou igual a 7.

No Passo 5 será construído o conjunto independente $I = \{V_1, V_2, V_3, V_4, V_5, V_6, V_{20}, V_{15}, V_{19}\}$ e o conjunto $H = \{V_7, V_8, V_9, V_{10}, V_{16}, V_{17}, V_{18}\}$. Assim, pode-se verificar se é possível a construção da estrutura *crown* sobre a coleção de conjuntos $C' = \{\{V_1, V_7\}, \{V_2, V_7\}, \{V_2, V_8\}, \{V_3, V_7\}, \{V_3, V_8\}, \{V_3, V_9\}, \{V_4, V_8\}, \{V_4, V_9\}, \{V_4, V_{10}\}, \{V_5, V_9\}, \{V_5, V_{10}\}, \{V_6, V_{10}\}, \{V_7, V_8\}, \{V_9, V_{10}\}, \{V_{15}, V_{16}\}, \{V_{16}, V_{17}\}, \{V_{17}, V_{18}, V_{19}\}\}$ de C. Como $|I| > |H|$, constrói-se a estrutura *crown* (H', I', M) tal que $I' = \{V_1, V_6, V_{20}\}$, $H' = \{V_7, V_{10}\}$ e $M = \{\{V_1, V_7\}, \{V_6, V_{10}\}\}$. Através desta estrutura, aplica-se a regra de redução *crown* de Abu-Khzam, removendo todos os elementos de $I' = \{V_1, V_6, V_{20}\}$ de todos os conjuntos na coleção C de subconjuntos, conforme mostra a Figura 4.3(g). Portanto, tem-se que a nova instância (S, C, k) é formada por $C = \{\{V_2, V_7\}, \{V_2, V_8\}, \{V_3, V_7\}, \{V_3, V_8\}, \{V_3, V_9\}, \{V_4, V_8\}, \{V_4, V_9\}, \{V_4, V_{10}\}, \{V_5, V_9\}, \{V_5, V_{10}\}, \{V_7, V_8\}, \{V_9, V_{10}\}, \{V_9, V_{13}\}, \{V_{10}, V_{14}\}, \{V_{13}, V_{14}\}, \{V_{14}, V_{16}\}, \{V_{15}, V_{16}\}, \{V_{16}, V_{17}\}, \{V_{17}, V_{18}, V_{19}\}\}$, $S = \{V_2, V_3, V_4, V_5, V_8, V_9, V_{10}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}\}$ e $k = 5$, $S' = \{V_{11}, V_{12}\}$.

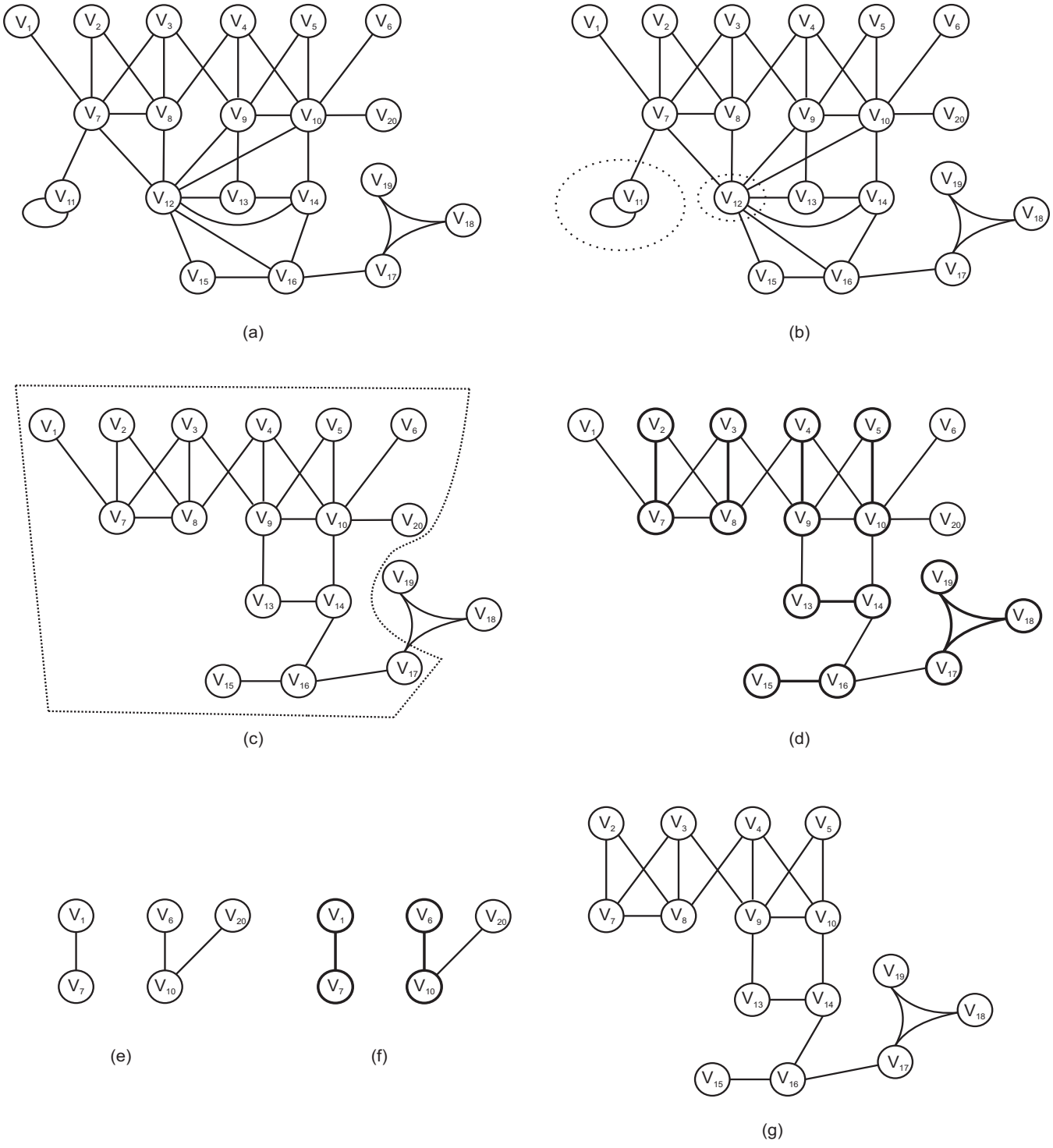


Figura 4.3: Exemplo de execução do Algoritmo de Abu-Khzam [4]. (a) Uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3. (b) Aplicação das regras de redução. (c) O subgrafo delimitado pela linhas pontilhadas é a coleção W. (d) As arestas em negrito são os elementos de M. (e) As arestas que conectam os vértices não cobertos por M. (f) As arestas em negrito são os elementos de M. (g) O grafo (coleção C de S) resultante da execução do Algoritmo de Abu-Khzam [4].

4.4 Conceitos Básicos do Algoritmo FPT de Niedermeier e Rossmanith

O algoritmo FPT de Niedermeier e Rossmanith [1] resolve o problema do 3-*Hitting Set*, para uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k. A seguir, serão introduzidos alguns conceitos básicos, necessários para que se possa desenvolver o algoritmo de Niedermeier e Rossmanith [1]. [6].

Definição 4.13 *O grau de um elemento é o número de subconjuntos em que o elemento ocorre.*

Definição 4.14 *Um singleton é um conjunto com um único elemento.*

Definição 4.15 *Um elemento $y \in S$ é dominado por um elemento $x \in S$ se para cada subconjunto que contenha y também contém x.*

Definição 4.16 *A coleção de subconjuntos é d-regular se cada elemento x tem exatamente grau d.*

O algoritmo FPT de Niedermeier e Rossmanith [1] é baseado nas duas técnicas elementares da Complexidade Parametrizada: redução ao núcleo do problema e árvore limitada de busca. O algoritmo de redução ao núcleo do problema, proposto por Niedermeier e Rossmanith, reduz, em tempo polinomial, uma instância de entrada de tamanho $|C| = n$, em uma nova instância de tamanho $O(k^3)$, removendo todos os elementos de grau maior que k^2, tal que o tamanho da nova instância do problema é dependente exclusivamente do parâmetro k, conforme descrito no Lema 4.2. Assim, o algoritmo da árvore limitada de busca trabalhará em instâncias dependentes somente do parâmetro k. O algoritmo é organizado em cinco casos simples e distintos, baseados nos graus dos elementos do conjunto S e no tamanho dos subconjuntos da coleção C de S, resumidos a seguir.

1. Caso 1: Composto por três subcasos simples: *singletons*, elementos de grau 1 e elementos dominantes (Definições 4.13, 4.14 e 4.15).
2. Caso 2: Quando existe algum subconjunto de tamanho 2.
3. Caso 3: Quando existem elementos de grau 3.
4. Caso 4: Quando existem elementos de grau pelo menos 4.
5. Caso 5: Quando existe uma coleção de subconjuntos 2-regulares (Definição 4.16).

[6] As definições, lemas e teoremas foram retirados de [1].

Lema 4.2 *Existe um núcleo do problema de tamanho $O(k^3)$ para o 3-Hitting Set que pode ser obtido em tempo linear.*

Prova. *Primeiramente, considere dois elementos fixos, $x, y \in S$:*

Afirmação 1*. Pode haver no máximo k subconjuntos de tamanho 3 na coleção C que contêm ambos x e y.*

Suponha que existem mais que k subconjuntos contendo x e y. Como cada conjunto aparece uma única vez em C, isto implica que existem mais que k diferentes "terceiros" elementos nos conjuntos correspondentes. Consequentemente, para cobrir os mais de k conjuntos diferentes com no máximo k elementos do conjunto base S, é necessário trazer pelo menos um x e um y ao hitting set S'. Isto significa que todos os conjuntos, contendo ambos x e y, podem ser removidos da coleção C. Logo, prova-se a Afirmação 1.

A seguir, considere que existe somente um elemento fixo $x \in S$:

Afirmação 2 *Pode haver no máximo k^2 subconjuntos de tamanho 3 na coleção C que contêm x.*

Suponha que existem mais que k^2 subconjuntos contendo x. Da Afirmação 1, sabe-se que x ocorre em um subconjunto junto com outro elemento y no máximo k vezes. Consequentemente, se houvesse mais que k^2 subconjuntos contendo x, estes não poderiam ser cobertos por algum $S' \subseteq S$ com $|S'| \leq k$ sem que $x \in S'$. Portanto, x deve estar em S' e o correspondente conjunto removido. Isto prova a Afirmação 2.

Agora, da Afirmação 2 pode-se concluir que cada elemento x de S pode ocorrer no máximo k^2 subconjuntos em C. (Senão x deveria estar no hitting set S'.) Evidentemente, pois $|S'| \leq k$. Isto significa (desde que C tem um hitting set de tamanho $\leq k$) que C pode consistir de no máximo $k.k^2 = k^3$ subconjuntos de tamanho 3. Este é o tamanho do núcleo do problema. Finalmente, pode-se facilmente computar em quantos conjuntos cada elemento ocorre e remover, em tempo linear, todos os elementos correspondentes aos subconjuntos que ocorrem em mais que k^2 subconjuntos.

□ [7]

Para estimar o tamanho da árvore limitada de busca, utilizam-se relações de recorrências descritas na Tabela 4.1 que resumem, em parte, a estrutura do algoritmo, listando os limites superiores para T_k e B_k, correspondentes ao tamanho da árvore limitada de busca. A recorrência B_k refere-se à existência de subconjuntos de tamanho 2. Enquanto que T_k refere-se somente aos subconjuntos de tamanho 3. Dessas relações de recorrência, Niedermeier e Rossmanith chegaram a uma análise simplificada, utilizando a fórmula de estimativa $T(k) \leq c^k$, em que c é número de ramificações que podem ser obtidas através da análise de ramificação do comportamento da árvore limitada de busca, como descrito por Niedermeier e Rossmanith [1] [8]. Por meio destas análises, obtém-se que o tamanho da árvore limitada de busca é $O(2.270^k)$.

[7] A demonstração deste lema foi retirado de [1].

[8] Fernau [3] mostra com mais detalhes a análise da complexidade de uma árvore limitada de busca.

$$B_k \leq \begin{cases} 2B_{k-1} & \text{Caso 2} \\ T_{k-1} + T_{k-2} + T_{k-3} & \text{Caso 2} \\ 2B_{k-2} + T_{k-1} & \text{Caso 2} \end{cases}$$

$$T_k \leq \begin{cases} B_k + T_{k-2} + T_{k-3} & \text{Caso 3} \\ 4B_{k-2} + T_{k-1} & \text{Caso 3} \\ B_{k-1} + T_{k-1} + T_{k-2} & \text{Caso 4} \\ 8B_{k-3} + T_{k-1} & \text{Caso 4} \\ 3B_{k-1} & \text{Caso 5} \end{cases}$$

Tabela 4.1. Limites superiores de T_k e B_k nos vários casos.

Resumindo, o tempo para reduzir uma instância do problema de tamanho n a uma nova instância de tamanho $O(k^3)$ é $O(n)$ (Lema 4.2) e o tamanho da árvore limitada de busca é $O(2.270^k)$. Segue que a complexidade de tempo do algoritmo FPT de Niedermeier e Rossmanith para o problema do 3-*Hitting Set* é $O(2.270^k k^3 + n)$, pois, para cada nó da árvore, é necessário processar a coleção C de subconjuntos. Com a aplicação da técnica de intercalação [15], tem-se que a complexidade de tempo é $O(2.270^k + n)$, conforme mostrado no Teorema 4.4.

Teorema 4.3 3-*Hitting Set pode ser resolvido em tempo* $O(2.270^k + n)$.

Prova. *No Lema 4.2 provou-se uma redução ao núcleo do problema de tamanho* $O(k^3)$*, em que a complexidade de tempo é* $O(n)$*. Na Subseção 4.4, a árvore de busca tem o tamanho limitado pelas recorrências da Tabela 4.1. Destas recorrências tem-se que o tamanho da árvore é* $O(2.270^k)$*. Como para cada nó da árvore de busca o tempo para processar a coleção de subconjuntos é linear no tamanho da entrada, obtém-se a complexidade de tempo* $O(2.270^k k^3 + n)$*. Agora, aplicando a técnica de intercalação de [15] o tempo de execução é reduzido para* $O(2.270^k + n)$*, substituindo* k^3 *por uma constante pequena.*

□ [9]

4.4.1 Algoritmo FPT de Niedermeier e Rossmanith

O algoritmo recebe uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k. Em cada nó da árvore limitada de busca, a escolha dos elementos a adicionar na cobertura parcial depende de uma busca em profundidade, considerando os cincos casos em que as escolhas dos elementos são em função dos subconjuntos escolhidos e do grau dos elementos, que poderá levar a várias ramificações.

[9] A demonstração deste teorema foi retirado de [1].

ALGORITMO 3-***Hitting Set***
Entrada: Uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k.
Saída: Um subconjunto $S' \subseteq S$, $|S'| \leq k$, tal que S' contém pelo menos um elemento de cada subconjunto em C, se existir ou uma mensagem, se tal subconjunto não existir.

1. **enquanto** $\exists x \in S$, tal que $\delta(x) \geq k^2 + 1$ **faça**
   $S' = S' \cup \{x\}; C = C \setminus \{C_i\}\ \forall C_i\ /\ x \in C_i$
2. **se** $|S'| > k$
   retorna "Não existe um *hitting set*";
3. Seja C' uma coleção de subconjuntos de tamanho no máximo 3 de S, resultante da remoção dos subconjuntos de C correspondentes aos elementos que estão em S'.
4. $k' = k - |S'|$
5. Seja T a árvore de busca cujo nó armazena $\langle C', k' \rangle$ e S'. Fazemos busca em profundidade na árvore T.
6. **para** cada nó da árvore T **faça**
7. Seja r o nó atual da árvore. Seja $\langle C'', k'' \rangle$ a instância reduzida do problema e CSP a cobertura por subconjuntos parcial armazenados em r.
8. **se** $k'' \geq 0$
9. **se** C'' for vazio
   retorna CSP
10. **enquanto** for possível **faça**
10.1. **enquanto** existir *singletons* ($\{x\}$) **faça**
   CSP = CSP $\cup \{x\}; \langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1 \rangle$
10.2. **enquanto** existir elementos de grau 1 **faça**
   Remova-os de cada subconjunto de C''.
10.3. **enquanto** existir elementos dominados **faça**
   Remova-os de cada subconjunto de C''.
11. **se** $\exists i$, tal que $|C''_i| = 2$
11.1. **se** $\exists x \in S$, tal que $\delta(x) = 2$ e b pode faltar no segundo; seja então $\{x, y\}$ e $\{x, a, b\}$, tais subconjuntos; o nó r gera dois filhos assim:
   (a) CSP = CSP $\cup \{y\}; \langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ y \in C''_i, k''' = k'' - 1 \rangle$;
   Remova x de $\{x, a, b\}$.
   (b) CSP = CSP $\cup \{x\}; \langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1 \rangle$;
   Remova y de qualquer outro subconjunto.
11.2. **senão se** $\exists x \in S$, tal que $\delta(x) = 3$ **e** $\exists i$, tal que $|C''_i| = 2$ **e** $\exists j$, tal que $|C''_j| = 3$; seja então $\{x, y_1\}$, $\{x, y_2\}$ e $\{x, a, b\}$, tais subconjuntos; o nó r gera dois filhos assim:
   (a) CSP = CSP $\cup \{x\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1 \rangle$
   (b) CSP = CSP $\cup \{y_1, y_2\}; \langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ y_1 \in C''_i$ ou $y_2 \in C''_i$, $k''' = k'' - 2 \rangle$
11.3. **senão se** existem três subconjuntos **e** $\exists x \in S$, tal que $\delta(x) \geq 3$ **e** $\{a, b\} \cap \{c, d\} \neq \emptyset$; seja então $\{x, y\}$, $\{x, a, b\}$ e $\{x, a, c\}$, tais subconjuntos; o nó r gera três filhos assim:
   (a) CSP = CSP $\cup \{x\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1 \rangle$
   (b) CSP = CSP $\cup \{y, a\}; \langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ y \in C''_i$ ou $a \in C''_i$, $k''' = k'' - 2 \rangle$
   (c) CSP = CSP $\cup \{y, b, c\}; \langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ y \in C''_i$ ou $b \in C''_i$ ou $c \in C''_i, k''' = k'' - 3 \rangle$

11.4. **senão se** existem três subconjuntos **e** $\exists x \in S$, tal que $\delta(x) \geq 3$ **e** $\{a,b\} \cap \{c,d\} = \emptyset$; seja então $\{x,y\}$, $\{x,a,b\}$ e $\{x,c,d\}$, tais subconjuntos; o nó r gera três filhos assim:
(a) CSP = CSP $\cup \{x\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1\rangle$
(b) CSP = CSP $\cup \{y,a\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ y \in C''_i$ ou $a \in C''_i$, $k''' = k'' - 2\rangle$; Remova x de $\{x,c,d\}$.
(c) CSP = CSP $\cup \{y,b\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ y \in C''_i$ ou $b \in C''_i$, $k''' = k'' - 2\rangle$; Remova x de $\{x,c,d\}$.

12. **se** $\exists x \in S$, tal que $\delta(x) = 3$ **e** $\forall i$, tal que $|C''_i| = 3$; seja então $\{x,a_1,a_2\}$, $\{x,b_1,b_2\}$ e $\{x,c_1,c_2\}$, tais subconjuntos; o nó r pode gerar oito filhos, realizando as seguintes operações:

12.1. **se** $\exists x \in S$, tal que $\delta(x) = 3$ **e** $\exists y \in S$, tal que $\delta(y) \geq 2$; o nó r gera três filhos assim:
(a) CSP = CSP $\cup \{a_1,c_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i$ ou $c_1 \in C''_i$, $k''' = k'' - 2\rangle$
(b) CSP = CSP $\cup \{a_2,b_2,c_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_2 \in C''_i$ ou $b_2 \in C''_i$ ou $c_1 \in C''_i, k''' = k'' - 3\rangle$
(c) Remova c_1 de $\{x,c_1,c_2\}$.

12.2. **senão se** existe somente um elemento de grau 3; o nó r pode gerar cinco filhos assim:
(a) CSP = CSP $\cup \{x\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1\rangle$
(b) CSP = CSP $\cup \{a_1,b_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i$ ou $b_1 \in C''_i$, $k''' = k'' - 2\rangle$; Remova x de $\{x,c_1,c_2\}$.
(c) CSP = CSP $\cup \{a_1,b_2\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i$ ou $b_2 \in C''_i$, $k''' = k'' - 2\rangle$; Remova x de $\{x,c_1,c_2\}$.
(d) CSP = CSP $\cup \{a_2,b_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_2 \in C''_i$ ou $b_1 \in C''_i$, $k''' = k'' - 2\rangle$; Remova x de $\{x,c_1,c_2\}$.
(e) CSP = CSP $\cup \{a_2,b_2\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_2 \in C''_i$ ou $b_2 \in C''_i$, $k''' = k'' - 2\rangle$; Remova x de $\{x,c_1,c_2\}$.

13. **se** $\exists x \in S$, tal que $\delta(x) \geq 4$ **e** $\forall i$, tal que $|C''_i| = 3$; seja então $\{x,a_1,a_2\}$, $\{x,b_1,b_2\}$, $\{x,c_1,c_2\}$ e $\{x,d_1,d_2\}$, tais subconjuntos; o nó r pode gerar doze filhos, realizando as seguintes operações:

13.1. **se** $\{a_1,a_2\} \cap \{b_1,b_2\} \cap \{c_1,c_2\} \cap \{d_1,d_2\} \neq \emptyset$; o nó r gera três filhos:
(a) CSP = CSP $\cup \{x\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1\rangle$
(b) CSP = CSP $\cup \{a_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i, k''' = k'' - 1\rangle$ Remova x de $\{x,c_1,c_2\}$.
(c) CSP = CSP $\cup \{a_2,b_2\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_2 \in C''_i$ ou $b_2 \in C''_i$, $k''' = k'' - 2\rangle$

13.2. **senão se** $\{a_1,a_2\} \cap \{b_1,b_2\} \cap \{c_1,c_2\} \cap \{d_1,d_2\} = \emptyset$; o nó r gera nove filhos:
(a) CSP = CSP $\cup \{x\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ x \in C''_i, k''' = k'' - 1\rangle$
(b) CSP = CSP $\cup \{a_1,b_1,c_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i$ ou $b_1 \in C''_i$ ou $c_1 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x,d_1,d_2\}$.
(c) CSP = CSP $\cup \{a_1,b_1,c_2\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i$ ou $b_1 \in C''_i$ ou $c_2 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x,d_1,d_2\}$.
(d) CSP = CSP $\cup \{a_1,b_2,c_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i$ ou $b_2 \in C''_i$ ou $c_1 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x,d_1,d_2\}$.
(e) CSP = CSP $\cup \{a_1,b_2,c_2\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_1 \in C''_i$ ou $b_2 \in C''_i$ ou $c_2 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x,d_1,d_2\}$.
(f) CSP = CSP $\cup \{a_2,b_1,c_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i\ /\ a_2 \in C''_i$ ou $b_1 \in C''_i$

ou $c_1 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x, d_1, d_2\}$.
(g) CSP = CSP $\cup \{a_2, b_1, c_2\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i$ / $a_2 \in C''_i$ ou $b_1 \in C''_i$
ou $c_2 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x, d_1, d_2\}$.
(h) CSP = CSP $\cup \{a_2, b_2, c_1\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i$ / $a_2 \in C''_i$ ou $b_2 \in C''_i$
ou $c_1 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x, d_1, d_2\}$.
(i) CSP = CSP $\cup \{a_2, b_2, c_2\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i$ / $a_2 \in C''_i$ ou $b_2 \in C''_i$
ou $c_2 \in C''_i, k''' = k'' - 3\rangle$; Remova x de $\{x, d_1, d_2\}$.

14. **se** existe uma coleção 2-regular **e** $\forall i$, tal que $|C''_i| = 3$; seja então $\{x, a, b\}$ e $\{x, c, d\}$, tais subconjuntos; o nó r gera três filhos assim:
(a) CSP = CSP $\cup \{a\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i$ / $a \in C''_i, k''' = k'' - 1\rangle$
Remova x de $\{x, c, d\}$.
(b) CSP = CSP $\cup \{x\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i$ / $x \in C''_i, k''' = k'' - 1\rangle$
Remova a de qualquer outro subconjunto.
(c) CSP = CSP $\cup \{b\}$; $\langle C''' = C'' \setminus \{C''_i\}\ \forall C''_i$ / $b \in C''_i, k''' = k'' - 1\rangle$
Remova x de $\{x, c, d\}$.

15. **senão**
Vá para o próximo nó válido da busca em profundidade na árvore T.
16. **retorna** "Não existe um *hitting set*".

Os primeiros quatros passos do algoritmo representam a fase de redução ao núcleo do problema, cujo tempo gasto para reduzir a instância original ($\langle C, k\rangle$) ao núcleo do problema de tamanho $O(k^3)$ ($\langle C'', k''\rangle$) é $O(n)$. Nesta fase, serão removidos os elementos que pertencem a pelo menos $k^2 + 1$ subconjuntos de C, adicionando-os ao *hitting set* S' e são removidos os elementos da coleção C correspondentes aos elementos que estão no *hitting set*, gerando C'.

A fase da árvore limitada de busca começa a partir do Passo 5. O nó raiz da árvore armazena a instância $\langle C', k'\rangle$ e o *hitting set* S' como cobertura parcial (Passo 5) resultante da fase de redução ao núcleo do problema. É realizada uma busca em profundidade na árvore com o objetivo de construir uma árvore com uma possível solução do problema. Observe que não é necessário gerar todos os nós da árvore (aplica-se o método de *backtracking*), somente um caminho da raiz até o nó em questão é computado a cada possibilidade de solução.

Para percorrer todos os nós da árvore T existe um laço dos Passos 6 a 15. Cada nó da árvore T armazena uma instância (Passo 7) reduzida do problema ($\langle C'', k''\rangle$) e uma cobertura parcial (CSP). Neste laço, a altura da árvore é limitada por k'' (Passo 8), restringindo o espaço de busca, ou seja, não adianta continuar a percorrer a árvore quando o nó em questão tem um k'' menor que zero, tendo em vista que haverá um *hitting set* de tamanho maior que k. Se o nó em questão tem uma instância com uma cobertura parcial de tamanho menor ou igual a k e com uma coleção C'' resultante vazia, significa que há um *hitting set* de tamanho menor ou igual a k para a coleção C de S (Passo 9).

Cada caso do algoritmo (com exceção do Caso 1) resulta em uma ou mais ramificações do nó atual a qual leva a algumas recorrências, que poderá ser vista com mais detalhes na Tabela 4.1. As relações de recorrências desses casos serão utilizadas para calcular o tamanho da árvore T, ou seja, a quantidade de nós desta árvore limitada de busca.

Neste algoritmo, os nós da árvore podem ter 1, 2, 3, 4, 5, 6, 7, 8 ou 9 filhos, dependendo do caso selecionado. A quantidade de elementos selecionados para serem adicionados à cobertura parcial também podem variar de acordo com o caso selecionado, podendo variar de um a três elementos adicionados à cobertura e em alguns casos nenhum elemento é adicionado à

cobertura. Os elementos adicionados à cobertura podem cobrir um ou mais conjuntos de acordo com as possibilidades de escolhas dos subconjuntos para serem analisados durante a busca em profundidade. Basicamente, existem cinco casos a considerar, numerados de 1 a 5. Seguem rigorosamente a ordem a seguir:

- **Caso 1** (Passo 10). Neste caso, a busca em profundidade resulta em um caso simples seguindo a ordem de importância: *singletons*, elementos de grau 1 e por fim elementos dominantes.
  - **Subcaso 1.1**. Primeiramente, se existe *singleton* ($\{x\}$), então adiciona-se esse elemento ao *hitting set* e remove-se todos os conjuntos que contenham o elemento x, como é ilustrado na Figura 4.4.

Figura 4.4: Um *singleton* $\{x\}$.

  - **Subcaso 1.2**. Seguinte a ordem de importância acima, suponha que existe um elemento $x \in S$ que ocorre somente em um conjunto. Seja um conjunto de tamanho 3 ($\{x, y, z\}$) tal que $\delta(y) \geq 0$ e $\delta(z) \geq 0$, como mostra a Figura 4.5. Como x ocorre somente neste conjunto e não cobrirá outros conjuntos, então remove-se x de $\{x, y, z\}$.

Figura 4.5: Seja o elemento x que ocorre somente no conjunto $\{x, y, z\}$. Então o elemento x possui grau 1.

  - **Subcaso 1.3**. Por fim, caso haja um elemento $x \in S$ dominado pelo elemento $y \in S$, então remove-se x de todos os subconjuntos, pois não faria sentido levar x ao *hitting set* e não levar y ao *hitting set*, como visto na Figura 4.6.

Figura 4.6: O elemento x é dominado pelo elemento y.

- **Caso 2** (Passo 11). A busca em profundidade resulta em pelo menos um subconjunto de tamanho 2 em C''. Conforme escolha destes conjuntos, têm-se quatro subcasos a considerar. Sem perda de generalidade, as ramificações serão realizadas de acordo com a possibilidade do elemento x estar ou não no *hitting set*, resultando nas recorrências

$$B_k \leq 2B_{k-1},\ B_k \leq T_{k-1} + T_{k-2} + T_{k-3} \text{ e } B_k \leq 2B_{k-2} + T_{k-1},$$

  vistas na Tabela 4.1.

– **Subcaso 2.1**. Primeiramente, suponha que existem dois subconjuntos

$$\{x, y\} \text{ e } \{x, a, b\},$$

tal que o elemento x ocorre somente nestes dois subconjuntos e b pode faltar no segundo conjunto ($\{x, a, b\}$). Se $y \in S'$, então o subconjunto $\{x, y\}$ está coberto e remove-se o elemento x de $\{x, a, b\}$. Como existe um subconjunto de tamanho 2 em C'', o nó atual terá uma subárvore de tamanho B_{k-1} nós, conforme é mostrado na Figura 4.7. Senão se $x \in S'$, os subconjuntos $\{x, y\}$ e $\{x, a, b\}$ estão cobertos. Como não há elementos que ocorram somente em um subconjunto, então o elemento y ocorre em qualquer outro subconjunto na qual é removido. Assim, o nó atual terá uma subárvore de tamanho B_{k-1} nós, como é visto na Figura 4.7.

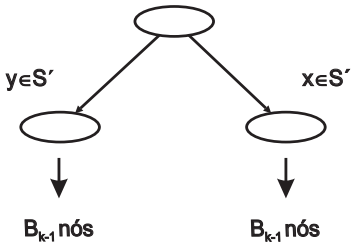


Figura 4.7: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

– **Subcaso 2.2**. Segundo, considere que existem três subconjuntos

$$\{x, y_1\},\ \{x, y_2\} \text{ e } \{x, a, b\},$$

onde x ocorre em pelo menos 2 subconjuntos de tamanho 2 e um subconjunto de tamanho 3. Se $x \in S'$, os subconjuntos $\{x, y_1\}$, $\{x, y_2\}$ e $\{x, a, b\}$ estão cobertos. Como existem somente subconjuntos de tamanho 3, o nó atual terá uma subárvore de tamanho T_{k-1} nós, conforme mostra a Figura 4.8. Caso contrário, se $y_1, y_2 \in S'$, então os subconjuntos $\{x, y_1\}$ e $\{x, y_2\}$ estão cobertos. Como somente os subconjuntos $\{x, y_1\}$ e $\{x, y_2\}$ foram cobertos e não há remoção de x do subconjunto $\{x, a, b\}$, o nó atual terá uma subárvore de tamanho T_{k-2} nós, conforme é visto na Figura 4.8.

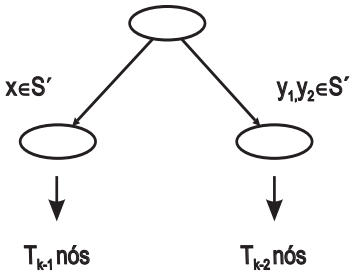


Figura 4.8: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

– **Subcaso 2.3**. Terceiro, suponha que existem três subconjuntos

$$\{x, y\},\ \{x, a, b\} \text{ e } \{x, a, c\},$$

onde o elemento x ocorre em outros subconjuntos e $b \neq c$. Se $x \in S'$, os subconjuntos $\{x, y\}$, $\{x, a, b\}$ e $\{x, a, c\}$ estão cobertos. Então, o nó atual terá uma subárvore de tamanho T_{k-1} nós, pois existem somente subconjuntos de tamanho 3 em C'', como mostra a Figura 4.9. Senão se $y, a \in S'$, os subconjuntos $\{x, y\}$, $\{x, a, b\}$ e $\{x, a, c\}$ estão cobertos. Com isso, há somente subconjuntos de tamanho 3, o nó atual terá uma subárvore de tamanho T_{k-2} nós, como é visto na Figura 4.9. Por fim, se $y, b, c \in S'$, os subconjuntos $\{x, y\}$, $\{x, a, b\}$ e $\{x, a, c\}$ estão cobertos. Visto que existem somente subconjuntos de tamanho 3, o nó atual terá uma subárvore de tamanho T_{k-3} nós, como é mostrado na Figura 4.9.

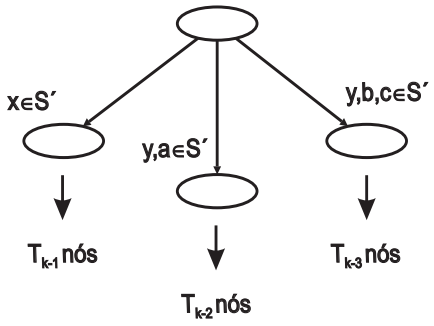


Figura 4.9: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

– **Subcaso 2.4**. Por último, caso haja três subconjuntos

$$\{x, y\},\ \{x, a, b\} \text{ e } \{x, c, d\},$$

onde o elemento x pode ocorrer em outros subconjuntos e $\{a, b\} \cap \{c, d\} = \emptyset$. Se $x \in S'$, os subconjuntos $\{x, y\}$, $\{x, a, b\}$ e $\{x, c, d\}$ estão cobertos, então cobrem-se todos os subconjuntos restando somente subconjuntos de tamanho 3 em C'', o nó atual terá uma subárvore de tamanho T_{k-1} nós, conforme mostra a Figura 4.10. Senão se $y, a \in S'$, somente os subconjuntos $\{x, y\}$ e $\{x, a, b\}$ estão cobertos e remove-se x do subconjunto $\{x, c, d\}$. Logo, o nó atual terá uma subárvore de tamanho B_{k-2} nós, como é visto na Figura 4.10. Por fim, se $y, b \in S'$, os subconjuntos $\{x, y\}$ e $\{x, a, b\}$ estão cobertos, e remove-se x do subconjunto $\{x, c, d\}$. Como existe um subconjunto de tamanho 2, o nó atual terá uma subárvore de tamanho B_{k-2} nós, conforme mostra a Figura 4.10.

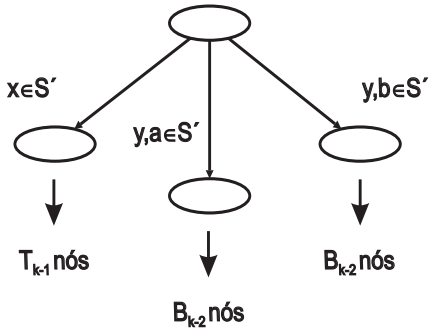


Figura 4.10: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

- **Caso 3** (Passo 12). A busca em profundidade resulta em um elemento de grau 3 na qual existem somente subconjuntos de tamanho 3 em C''. Sejam, então,

$$\{x, a_1, a_2\},\ \{x, b_1, b_2\} \text{ e } \{x, c_1, c_2\}$$

tais subconjuntos. As ramificações serão realizadas de acordo com a possibilidade do elemento x estar ou não no *hitting set*, resultando nas recorrências

$$T_k \leq B_k + T_{k-2} + T_{k-3} \text{ e } T_k \leq 4B_{k-2} + T_{k-1},$$

vistas na Tabela 4.1.

  - **Subcaso 3.1**. Primeiramente, suponha que existe um elemento de grau 3 e outro elemento que ocorre em pelo menos dois destes subconjuntos, tal que $a_1 = b_1$ com $a_1 \neq c_1$ e $a_1 \neq c_2$. Se $a_1, c_1 \in S'$, os subconjuntos $\{x, a_1, a_2\}$, $\{x, b_1, b_2\}$ e $\{x, c_1, c_2\}$ estão cobertos, pois $a_1 = b_1$. Visto que existem somente subconjuntos de tamanho 3 em C'', o nó atual terá uma subárvore de tamanho T_{k-1} nós, como mostra a Figura 4.11. Senão se $c_1 \in S'$, o subconjunto $\{x, c_1, c_2\}$ está coberto. Porém, x é dominado por $a_1 (a_1 = b_1)$. Logo, existem dois *singletons* $\{a_2\}$ e $\{b_2\}$, então $a_2, b_2 \in S'$. Como existem somente subconjuntos de tamanho 3, o nó atual terá uma subárvore de tamanho T_{k-3} nós, conforme é visto na Figura 4.11. Por último, remove-se c_1 de $\{x, c_1, c_2\}$, restando um subconjunto de tamanho 2. Logo, o nó atual terá uma subárvore de tamanho B_k nós, como mostra a Figura 4.11.

  - **Subcaso 3.2**. Por fim, considere que existe somente um elemento de grau 3 e que a_1, a_2, b_1, b_2, c_1, c_2 são todos pares diferentes. Se $x \in S'$, os subconjuntos $\{x, a_1, a_2\}$, $\{x, b_1, b_2\}$ e $\{x, c_1, c_2\}$ estão cobertos. Então, o nó atual terá uma subárvore de tamanho T_{k-1} nós, como mostra a Figura 4.12, pois existem somente subconjuntos de tamanho 3 em C''. Caso contrário, as ramificações dos nós serão realizadas de acordo com a possibilidade de a_1 ou a_2 estar em S'', e b_1 ou b_2 estar em S'. Como somente os subconjuntos $\{x, a_1, a_2\}$ e $\{x, b_1, b_2\}$ serão cobertos e é removido x do subconjunto $\{x, c_1, c_2\}$, os nós atuais terão subárvores de tamanho B_{k-2} nós, como é visto na Figura 4.12.

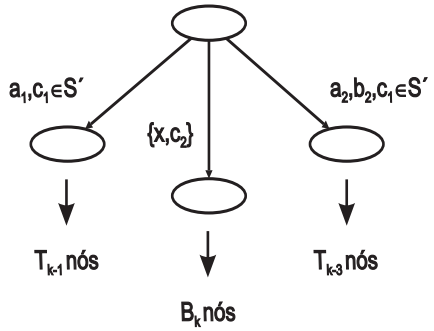


Figura 4.11: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

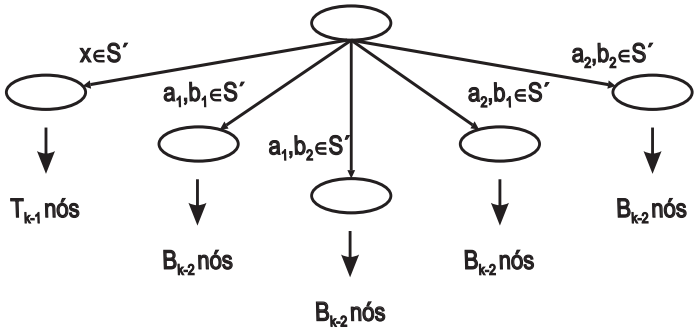


Figura 4.12: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

- **Caso 4** (Passo 13). A busca em profundidade resulta em um elemento de grau pelo menos 4 em C''. Sejam, então,

$$\{x, a_1, a_2\},\ \{x, b_1, b_2\},\ \{x, c_1, c_2\} \text{ e } \{x, d_1, d_2\}$$

tais subconjuntos. As ramificações serão realizadas de acordo com a possibilidade do elemento x estar ou não no *hitting set*, resultando nas recorrências

$$T_k \leq B_{k-1} + T_{k-1} + T_{k-2} \text{ e } T_k \leq 8B_{k-3} + T_{k-1},$$

vistas na Tabela 4.1.

  - **Subcaso 4.1**. Primeiramente, suponha que $\{a_1, a_2\} \cap \{b_1, b_2\} \cap \{c_1, c_2\} \cap \{d_1, d_2\} \neq \emptyset$ tal que $a_1 = b_1$. Se $x \in S'$, os subconjuntos $\{x, a_1, a_2\}$, $\{x, b_1, b_2\}$, $\{x, c_1, c_2\}$ e $\{x, d_1, d_2\}$ estão cobertos. Como existem somente subconjuntos de tamanho 3 em C'', o nó atual terá uma subárvore de tamanho T_{k-1} nós, como mostra a Figura 4.13. Senão se $a_1 \in S'$, os subconjuntos $\{x, a_1, a_2\}$ e $\{x, b_1, b_2\}$ estão cobertos, pois $a_1 = b_1$. Além disso, remove-se x de $\{x, c_1, c_2\}$, restando um subconjunto de tamanho 2 ($\{c1, c2\}$) e outro subconjunto de tamanho 3 ($\{x, d_1, d_2\}$). Então, o nó atual terá uma subárvore de tamanho B_{k-1} nós, como é visto na Figura 4.13. Por último, se

$a_2, b_2 \in S'$, os subconjuntos $\{x, a_1, a_2\}$ e $\{x, b_1, b_2\}$ estão cobertos. Suponha-se que x não é dominado por a_1 ou vice e versa, restando somente a_2 e b_2 para cobrirem os subconjuntos $\{x, a_1, a_2\}$ e $\{x, b_1, b_2\}$. Como somente os subconjuntos $\{x, a_1, a_2\}$ e $\{x, b_1, b_2\}$ foram cobertos e não há remoção de x dos subconjuntos $\{x, c_1, c_2\}$ e $\{x, d_1, d_2\}$, o nó atual terá uma subárvore de tamanho T_{k-2} nós, como é visto na Figura 4.13.

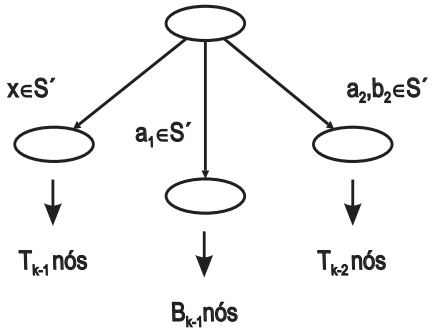


Figura 4.13: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

– **Subcaso 4.2**. Por fim, considere que $\{a_1, a_2\} \cap \{b_1, b_2\} \cap \{c_1, c_2\} \cap \{d_1, d_2\} = \emptyset$. Se $x \in S'$, os subconjuntos $\{x, a_1, a_2\}$, $\{x, b_1, b_2\}$, $\{x, c_1, c_2\}$ e $\{x, d_1, d_2\}$ estão cobertos. Como existem somente subconjuntos de tamanho 3, o nó atual terá uma subárvore de tamanho T_{k-1} nós, conforme mostra a Figura 4.14. Caso contrário, se $x \notin S'$, as ramificações dos nós serão realizadas de acordo com a possibilidade de a_1 ou a_2 estar em S', b_1 ou b_2 estar em S', e c_1 ou c_2 estar em S'. Visto que somente os subconjuntos $\{x, a_1, a_2\}$, $\{x, b_1, b_2\}$ e $\{x, c_1, c_2\}$ foram cobertos e há remoção de x do subconjunto $\{x, d_1, d_2\}$, os nós atuais terão subárvores de tamanho B_{k-3} nós, como é visto na Figura 4.14.

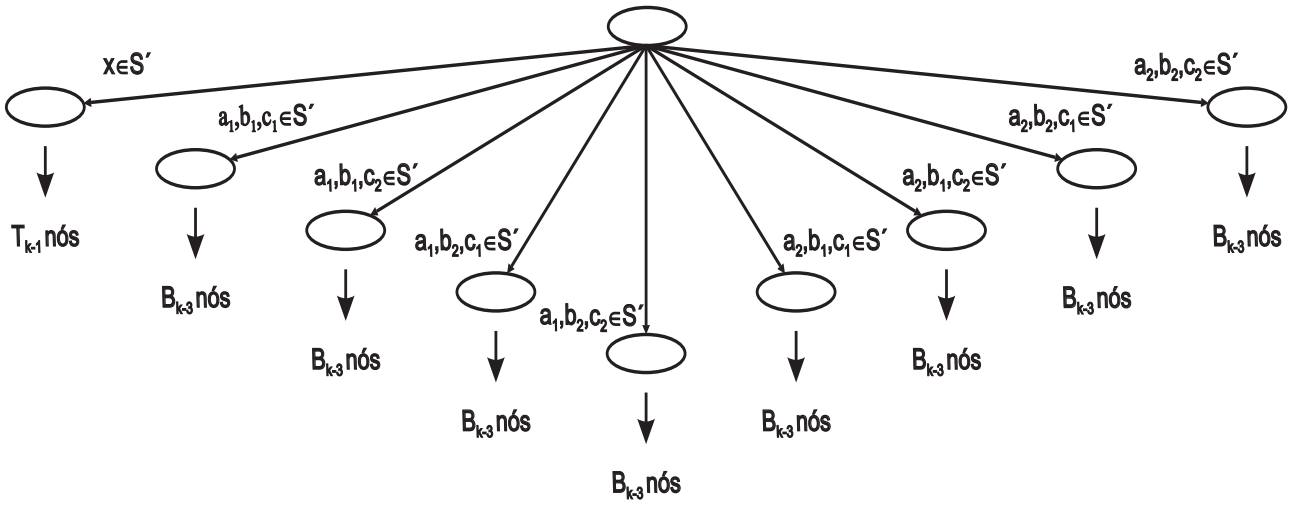


Figura 4.14: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

- **Caso 5** (Passo 14). A busca em profundidade resulta em uma coleção 2-regular em C''. Sejam, então,

$$\{x, a, b\} \text{ e } \{x, c, d\}$$

tais subconjuntos e suponha que a,b,c,d são todos pares diferentes, caso contrário pode haver elementos dominantes. As ramificações serão realizadas de acordo com a possibilidade do elemento a estar ou não no *hitting set*, resultando na recorrência

$$T_k \leq 3B_{k-1},$$

vista na Tabela 4.1.

Se $a \in S'$, o conjunto $\{x, a, b\}$ está coberto. Então deve-se remover x de $\{x, c, d\}$, pois x é dominado depois de remover o subconjunto $\{x, a, b\}$ de C''. Logo, o nó atual terá uma subárvore de tamanho B_{k-1} nós, como é visto na Figura 4.15. Senão se $x \in S'$, os subconjuntos $\{x, a, b\}$ e $\{x, c, d\}$ estão cobertos. Como é possível haver 2-regularidade, remove-se o elemento a de qualquer outro subconjunto de C''. Então, o nó atual terá uma subárvore de tamanho B_{k-1} nós, conforme mostra a Figura 4.15. Por fim, se $b \in S'$, o subconjunto $\{x, a, b\}$ está coberto. Além disso, remove-se o elemento x do subconjunto $\{x, c, d\}$, devido ao fato de x ser dominado depois que o subconjunto $\{x, a, b\}$ ter sido removido de C''. Então, o nó atual terá uma subárvore de tamanho B_{k-1} nós, como é visto na Figura 4.15.

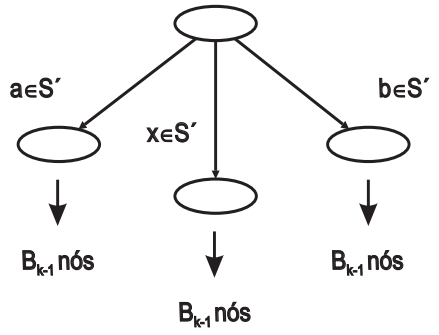


Figura 4.15: Conforme a possibilidade de x estar ou não no *hitting set*, poderá haver as ramificações na árvore limitada de busca.

Os cincos casos descritos garantem a geração de uma árvore limitada de busca, conforme a Tabela 4.1. Por meio de uma análise da árvore, utilizando a Tabela 4.1, pode-se estimar que o tamanho da árvore de busca é $T(k) \leq 2.270^k$. Sabe-se que a complexidade do algoritmo de redução ao núcleo do problema é $O(n)$. Do Teorema 4.4 tem-se que a complexidade de tempo do Algoritmo FPT de Niedermeier e Rossmanith é $O(2.270^k k^3 + n)$. Agora, aplicando a técnica de intercalação, tem-se que a complexidade de tempo do Algoritmo FPT de Niedermeier e Rossmanith é $O(2.270^k + n)$.

Exemplo da Execução do Algoritmo

Na Figura 4.16 será apresentada um exemplo da execução do algoritmo de Niedermeier e Rossmanith [1] para o problema do 3-*Hitting Set*. Seja uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3, apresentada na Figura 4.16(a), deseja-se determinar se existe um *hitting set* em C de tamanho menor ou igual a 8, ou seja, $k \leq 8$.

A fase de redução ao núcleo do problema resulta na coleção C', mostrada na Figura 4.16(b), na qual são adicionados os elementos de grau pelo menos $k^2 + 1$ (o elemento x) ao *hitting set* parcial S' e são removidos os elementos da coleção C correspondentes aos elementos que estão em S'. O elemento x é adicionado a S', pois sabe-se que o elemento x deve pertencer a qualquer *hitting set* de tamanho menor ou igual a 8. Com isso, o valor de k' passa a ser 7.

Na Figura 4.16(b) é apresentada a árvore limitada de busca gerada pelo algoritmo de Niedermeier e Rossmanith [1]. O nó raiz da árvore de busca armazena a instância resultante da redução ao núcleo do problema ($\langle C', 7\rangle$) e um *hitting set* parcial composto por $\{x\}$. Primeiramente, verifica-se que não existem *singletons*, elementos de grau 1 ou elementos dominantes (Caso 1) em C'. A busca em profundidade resulta em um conjunto de tamanho 2 e outro conjunto de tamanho 3 ($\{b, c\}$ e $\{c, d, f\}$), e o elemento c que ocorre somente nestes dois conjuntos, então, neste caso, são gerados dois nós filhos, conforme a possibilidade do elemento c estar ou não no *hitting set* (Subcaso 2.1), e com a possibilidade de adição de um elemento ao *hitting set* para cada ramificação do nó atual. Será mostrado somente o nó filho, NÓ 2. Então, o NÓ 2 é visitado.

A coleção contida no NÓ 2 resulta da remoção dos conjuntos $\{b, c\}$ e $\{c, d, f\}$, bem como de todos os conjuntos que contenham o elemento c e remove-se o elemento b do conjunto $\{b, p, n\}$. Como o elemento c é adicionado ao *hitting set*, pode-se adicionar, no máximo, mais seis elementos ao *hitting set* e ainda obter um *hitting set* de tamanho válido. Como existe um elemento de grau 1, o elemento b na coleção C'', então remove-se o elemento b do conjunto $\{b, p, d\}$. A busca em profundidade resulta em dois conjuntos de tamanho 2 e um conjunto de tamanho 3, $\{p, n\}$, $\{p, d\}$ e $\{p, e, f\}$ (Subcaso 2.2). Neste caso, são gerados dois nós filhos conforme a possibilidade do elemento p estar ou não no *hitting set*, e com a possibilidade de adição de um elemento (o elemento p) ao *hitting set*, para o nó filho, NÓ 3, e dois elementos ao *hitting set*, para os nós filhos que não serão ilustrados neste exemplo. Então, o NÓ 3 é visitado.

A coleção contida no NÓ 3 resulta da remoção dos conjuntos $\{p, n\}$, $\{p, d\}$ e $\{p, e, f\}$, bem como de todos os conjuntos que contenham o elemento p. Como o elemento p é adicionado ao *hitting set* pode-se adicionar no máximo mais cinco elementos ao *hitting set* e ainda obter um *hitting set* de tamanho válido. Como existem somente conjuntos de tamanho 3 e existe um elemento de grau 3 (elemento e) então a busca em profundidade resulta na seleção de três conjuntos, $\{e, f, m\}$, $\{e, h, j\}$ e $\{e, d, n\}$ (Subcaso 3.2). Neste caso, são gerados cinco nós filhos, conforme a possibilidade do elemento e estar ou não no *hitting set*, e com a possibilidade de adição de um elemento (o elemento e) ao *hitting set*, para a instância contida no nó filho, NÓ 4, e dois elementos ao *hitting set*, para cada nó filho, ilustrados pelos ramos nas quais não serão apresentados neste exemplo. Então, o NÓ 4 é visitado.

A coleção contida no NÓ 4 resulta da remoção dos conjuntos $\{e, f, m\}$, $\{e, h, j\}$ e $\{e, d, n\}$, bem como de todos os conjuntos que contenham o elemento e. Como o elemento e é adicionado ao *hitting set*, pode-se adicionar no máximo mais quatro elementos ao *hitting set* e ainda obter um *hitting set* de tamanho válido. Como existem elementos de grau 1, os elementos j, n, d e h na coleção C'', então tais elementos serão removidos dos respectivos conjuntos, $\{j, n, m\}$ e $\{d, h, f\}$.

Os elementos j e n são removidos do conjunto $\{j, n, m\}$, resultando em um *singleton* $\{m\}$ o qual é adicionado ao *hitting set* e remove-se o *singleton* $\{m\}$ da coleção C'', bem como todos os conjuntos que contenham o elemento m. O valor de k'' é atualizado para $k'' = 3$. Removem-se também d e h do conjunto $\{d, h, f\}$, resultando em um *singleton* $\{f\}$ o qual é adicionado ao *hitting set* e remove-se o conjunto $\{f\}$ da coleção C'', bem como todos os conjuntos que contenham o elemento f. O valor de k'' é atualizado para $k'' = 2$. A busca em profundidade resulta em uma coleção 2-regular, $\{i, v, l\}$ e $\{i, r, w\}$ (Caso 5). Neste caso, são gerados três nós filhos conforme a possibilidade do elemento v estar ou não no *hitting set*, com a possibilidade de adição de um elemento ao *hitting set*, para cada nó filho gerado. Será mostrado somente o nó filho, NÓ 5. Então, o NÓ 5 é visitado.

A coleção contida no NÓ 5 resulta da remoção do conjunto $\{i, v, l\}$ e da remoção do elemento i do conjunto $\{i, r, w\}$, bem como de todos os conjuntos que contenham o elemento v. Como o elemento v é adicionado ao *hitting set*, pode-se adicionar no máximo mais um elemento ao *hitting set* e ainda obter um *hitting set* de tamanho válido. Como existe um elemento de grau 1, o elemento l na coleção C'', então remove-se o elemento l do conjunto $\{l, r, s\}$. A busca em profundidade resulta em dois conjuntos de tamanho 2 (Subcaso 2.1). Neste caso, são gerados dois nós filhos, conforme a possibilidade do elemento s estar ou não no *hitting set*, e com a possibilidade de adição de um elemento ao *hitting set*, para cada nó filho gerado. O NÓ 6 é visitado, primeiramente.

A coleção contida no NÓ 6 resulta da remoção do conjunto $\{r, s\}$ e da remoção do elemento r do conjunto $\{r, w\}$, bem como de todos os conjuntos que contenham o elemento s. O elemento s é adicionado ao *hitting set*, mas o *hitting set* atingiu o tamanho máximo e ainda existe um conjunto na coleção C''. Então, interrompe-se o crescimento da árvore nesse nó, pois não obtém-se um *hitting set* de tamanho no máximo 8, e prossegue-se para o próximo nó da busca em profundidade, o NÓ 7.

A coleção contida no NÓ 7 resulta da remoção dos conjuntos $\{r, s\}$ e $\{r, w\}$, bem como de todos os conjuntos que contenham o elemento r. Como o elemento r é adicionado ao *hitting set*, não se pode mais adicionar nenhum elemento ao *hitting set* e ainda obter um *hitting set* de tamanho válido. Contudo, a coleção contida nesse nó é vazia. Assim, o *hitting set* formado por $\{x, c, p, e, m, f, v, r\}$ é de tamanho menor ou igual a 8 para a coleção C de S. Portanto, o *hitting set* $\{x, c, p, e, m, f, v, r\}$ é uma solução válida para a coleção C de S.

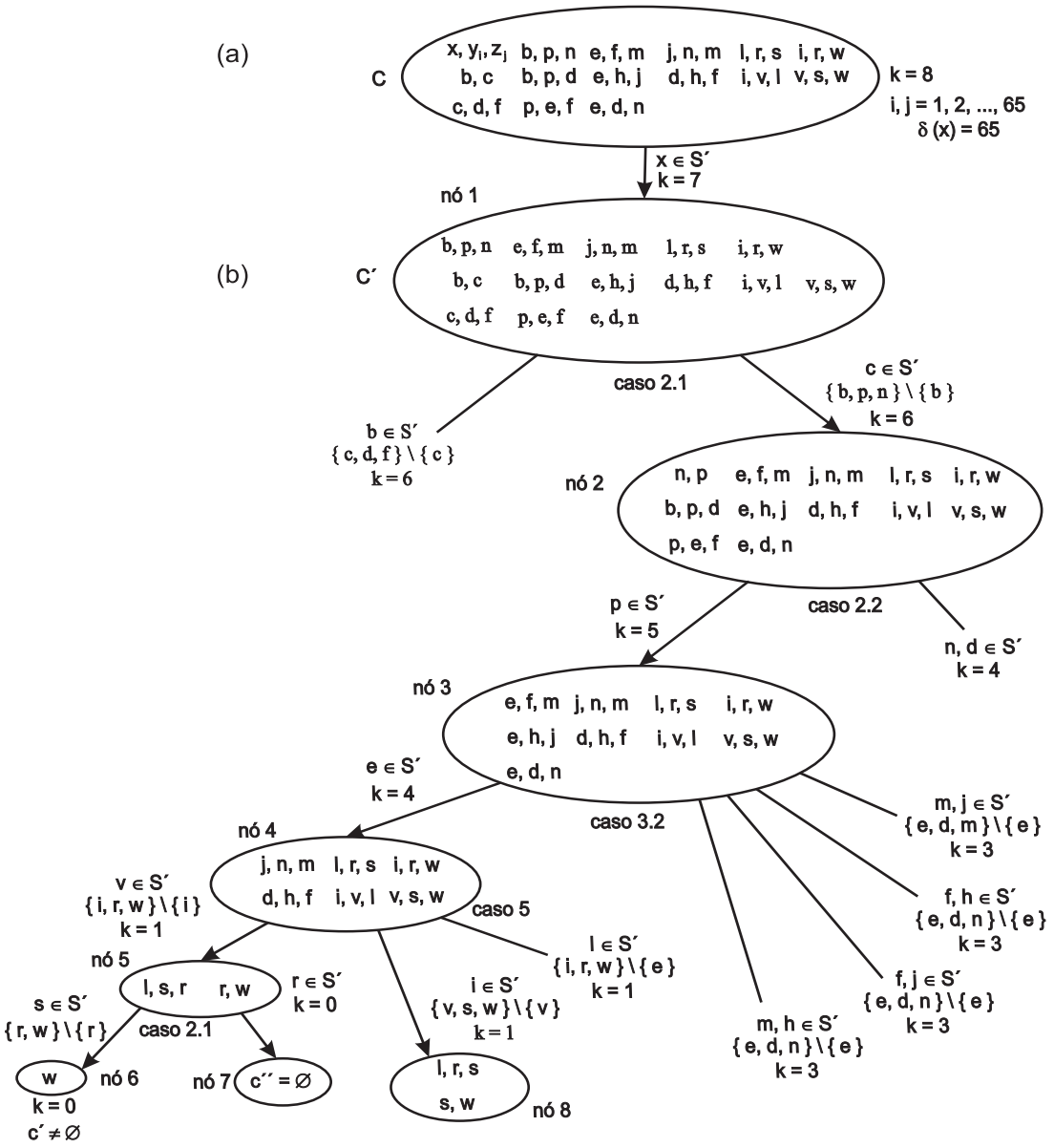


Figura 4.16: Exemplo da execução do Algoritmo de Niedermeier e Rossmanith [1]. (a) Uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3. (b) A árvore limitada de busca do Algoritmo de Niedermeier e Rossmanith [1].

4.5 Algoritmo de Cheetham *et al.*

O algoritmo BSP/CGM de Cheetham *et al.* [2] paraleliza as fases de redução ao núcleo do problema e árvore limitada de busca. Foram alteradas ambas as fases, substituindo os algoritmos de Balasubramanian *et al.* [41] e o algoritmo de Buss (método de Buss) [54] pelo algoritmo de Niedermeier e Rossmanith [1].

ALGORITMO: Paralelização da redução ao núcleo do problema (Versão adaptada do algoritmo de Cheetham *et al.* [2] para o 3-*Hitting Set*)
Entrada: Uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k.
Saída: Uma instância $\langle C', k' \rangle$ e S' ou um *hitting set* de tamanho no máximo k ou uma mensagem, se tal subconjunto não existir.

1. Simule a fase de redução ao núcleo do problema utilizando um algoritmo de redução ao núcleo do problema para o 3-*Hitting Set* (algoritmo de Niedermeier e Rossmanith [1]) em $O(1)$ ordenações paralelas de inteiros, usando ordenação determinística por amostragem [55].

2. Devolva $\langle C', k' \rangle$ ou um *hitting set* de tamanho menor ou igual a k para C de S ou escreva "Não existe um *hitting set* desejado".

Na fase de redução ao núcleo do problema foi utilizado o mesmo método sequencial de redução ao núcleo do problema proposto por Niedermeier e Rossmanith [1]. Na versão paralela, cada processador P_i, $0 \leq i \leq p-1$ é responsável por computar o grau dos elementos rotulados de $i * (n/p)$ a $((i+1) * (n/p)) - 1$, recebe m/p conjuntos da coleção C de S, ordena-as pelo primeiro elemento do conjunto e informa aos demais processadores quais elementos locais têm grau maior que k^2. Assim todos os processadores serão capazes de remover os conjuntos que contêm tais elementos. Em seguida, cada processador trocará mensagens informando os conjuntos resultantes, de forma que cada processador P_i tenha uma cópia da instância final da fase de redução paralela ao núcleo do problema.

ALGORITMO: Paralelização da árvore limitada de busca (Versão adaptada do algoritmo de Cheetham *et al.* para o 3-*Hitting Set*)
Entrada: Uma coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e um inteiro positivo k.
Saída: Um subconjunto $S' \subseteq S$, $|S'| \leq k$, tal que S' contém pelo menos um elemento de cada subconjunto em C, se existir; ou uma mensagem, se tal subconjunto não existir.

1. Seja a árvore t-ária completa T, $|t| > 1$, gerada pelo algoritmo de Niedermeier e Rossmanith [1], sendo que tal algoritmo é executado de forma determinística e usando busca em largura até que existam p folhas γ_o a γ_i. O nó raiz de T armazena $\langle C', k' \rangle$ e um *hitting set* parcial. Cada processador P_i, $0 \leq i \leq p-1$ percorre o caminho único que leva da raiz de T até a folha γ_i. Sejam $\langle C'', k'' \rangle$, as instâncias associadas a cada folha γ_i da árvore T.

2. Cada processador P_i executa localmente o algoritmo de Niedermeier e Rossmanith [1] sequencial a partir da folha γ_i, gerando uma subárvore cujo nó raiz é a folha γ_i da seguinte forma:

Sempre que houver ramificação em um nó de subárvore, o processador P_i escolhe aleatoriamente um dos nós filhos gerados e executa o algoritmo recursivamente até que uma solução seja encontrada. Se atingir uma folha de sua subárvore limitada de busca, o processador P_i volta na árvore e escolhe outro nó filho ainda não explorado (através da aplicação do método de *backtracking*). Esse processo é repetido até encontrar uma solução (nesse caso temos que notificar os outros processadores para terminarem) ou até que a subárvore de raiz γ_i seja completamente percorrida.

Na fase da árvore limitada de busca paralela, é gerada a árvore t-ária completa T, $|t| > 1$, com altura $h = log_b p$ e com p folhas (γ_o a γ_i) onde b é o fator de ramificação (por exemplo, binário ou ternário), pelo algoritmo de Niedermeier e Rossmanith [1] de forma determinística, com o nó raiz armazenando um *hitting set* parcial e a instância $\langle C', k' \rangle$. Observe que no início desta fase, todos os processadores têm as mesmas instâncias geradas pela fase de redução paralela ao núcleo do problema.

A árvore T não é explicitamente criada pelos processadores (aplica-se o método de *backtracking*). Cada processador P_i, $0 \leq i \leq p-1$ percorre o único caminho entre a raiz e a folha γ_i, calculado pela fórmula $i/(p/b^{h+1})$, o algoritmo de Niedermeier e Rossmanith [1] é interrompido quando atinge um nó de nível $log_b p$ (folha γ_i). O elemento inicial da busca em profundidade deve ser o mesmo para todos os processadores, para que todos os processadores computem a mesma árvore T.

Em seguida, cada processador P_i executa localmente a instância $\langle C_i'', k_i'' \rangle$ referente à folha γ_i da árvore T usando o algoritmo de Niedermeier e Rossmanith [1], como apresentado na Figura 4.17. Esta fase será encerrada ou quando encontrar um *hitting set* para C de tamanho menor ou igual a k, ou se todos os nós da árvore foram visitados, o que significa que não foi possível encontrar um *hitting set* válido.

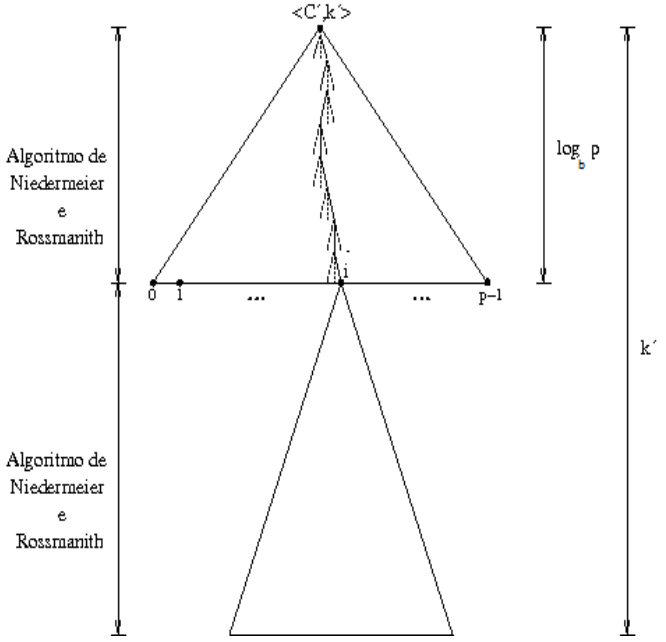


Figura 4.17: O processador P_i processa o caminho único até a folha γ_i usando o algoritmo de Niedermeier e Rossmanith [1]. Depois, P_i processa toda a subárvore de raíz γ_i usando o algoritmo de Niedermeier e Rossmanith [1].

Capítulo 5

Implementação

5.1 Introdução

Neste capítulo serão detalhadas as estruturas de dados utilizadas na solução do problema do 3-*Hitting Set*, bem como os detalhes de sua implementação. Foi implementada uma adaptação do algoritmo FPT paralelo de Cheetham *et al.* [2] no modelo BSP/CGM para resolver o problema do 3-*Hitting Set*, utilizando o algoritmo de Niedermeier e Rossmanith [1] nas fases de redução ao núcleo do problema e árvore limitada de busca. A implementação utiliza a linguagem C/C++ em conjunto com a biblioteca de comunicação MPI. Na Seção 5.2 serão apresentados os dados de entrada do programa. Por fim, na Seção 5.3 serão mostrados os detalhes da implementação do algoritmo de Niedermeier e Rossmanith [1].

5.2 Entrada do Programa

O programa recebe como entrada um arquivo texto e um número inteiro positivo. O arquivo texto descreve a coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 e o número inteiro positivo k determina o tamanho máximo desejado para um *hitting set* para a coleção C de S.

S = {1, 2, 3}	1 {1, 2} {1} {1, 2, 3}	1 2 3 -1
C = {{1}, {1, 2}, {1, 2, 3}}	2 {2, 1} {2, 1, 3}	1 -1 1 2 -1 1 2 3 -1
	3 {3, 2, 1}	

Figura 5.1: A coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3, a estrutura de dados e o arquivo de entrada do programa.

Na Figura 5.1 são apresentadas uma coleção C de S, uma representação abstrata da estrutura de dados utilizada pelo programa e um exemplo de arquivo de entrada que o descreve, respectivamente. Note que o programa interpreta os dados de entrada utilizando alguns conceitos básicos de grafos. A primeira linha do arquivo de entrada representa o conjunto finito S e

a partir da segunda linha tem-se a coleção C de subconjuntos. O número -1 delimita o final de uma linha ou o final de um conjunto.

5.3 Implementação Niedermeier e Rossmanith

Na implementação foram adaptadas as idéias apresentado por Cheetham *et al.* [2] para o problema do 3-*Hitting Set* descritas no capítulo anterior. Deve-se notar que o algoritmo de Cheetham *et al.* [2] resolve o problema da k-Cobertura por Vértices. Utilizou-se o algoritmo de Niedermeier e Rossmanith [1] nas fases de redução do núcleo do problema e construção da árvore limitada de busca, do algoritmo de Cheetham *et al.* [2]. Para processar cada nó resultante da árvore limitada de busca foi utilizado o algoritmo sequencial de Niedermeier e Rossmanith [1].

Na Figura 5.2 é ilustrada uma visão geral das estruturas de dados utilizadas pela implementação. Tais estruturas de dados utilizam alguns conceitos básicos de grafos. Para isso, interpretam-se os conjuntos como hiperarestas e os elementos como vértices. Com isso, foi possível utilizar listas de adjacências para representar a coleção C de S como um hipergrafo. Contudo, devido ao projeto do algoritmo de Niedermeier e Rossmanith [1] do capítulo anterior, serão referidos arestas como conjuntos e vértices como elementos. Note que a ordem dos elementos no conjunto é somente uma forma de agrupar os conjuntos nas listas de adjacências de forma que a lista do elemento $e \in S$ é composta por todos os subconjuntos que contêm tal elemento. As listas de grau de elementos e as listas de adjacências facilitam a seleção dos conjuntos e dos respectivos elementos, bem como a remoção dos conjuntos e os respectivos elementos.

Para facilitar a descrição da implementação, define-se n como o número de elementos do conjunto finito S, m como o número de conjuntos da coleção C e p como o número de processadores. Cada processador recebe um rótulo P_i, $0 \leq i \leq p-1$. Apenas o processador P_0 lê o arquivo de entrada.

Na fase de redução ao núcleo do problema, a coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3 é lido como uma lista de conjuntos (arestas) pelo processador P_0. Cada processador P_i, $0 \leq i \leq p-1$ é responsável por computar o grau dos elementos rotulados de $i * (n/p)$ a $((i+1) * (n/p)) - 1$, recebe m/p conjuntos da lista de conjuntos da coleção C, ordena-as pelo primeiro elemento do conjunto, informando aos demais processadores quais elementos locais têm grau maior que k^2. Assim todos os processadores serão capazes de remover os conjuntos que contêm tais elementos e os respectivos elementos. Em seguida, cada processador trocará mensagens enviando os conjuntos resultantes, de forma que cada processador tenha uma cópia da instância final da fase de redução ao núcleo.

Por fim, na fase da árvore limitada de busca, a lista de conjuntos é transformada em listas de adjacências, conforme ilustra a Figura 5.2. O algoritmo de Niedermeier e Rossmanith [1] gera uma árvore t-ária completa T, $|t| > 1$, com altura $h = log_b p$ e com p folhas (y_0 a $y_p - 1$), onde b é o fator de ramificação (por exemplo, binário ou ternário), de forma determinística, com o nó raiz armazenando o *hitting set* parcial e a instância $\langle C', k' \rangle$.

A árvore T não é explicitamente criada pelos processadores (o método de *backtracking* é aplicado). Cada processador P_i, $0 \leq i \leq p-1$ percorre o único caminho entre a raiz e a folha y_i calculado pela fórmula $i/(p/2^{h+1})$. O algoritmo de Niedermeier e Rossmanith [1] é interrompido quando atinge um nó de nível $log_b p$ (folha y_i). O elemento inicial da busca em profundidade deve ser o mesmo para todos os processadores.

Em seguida, cada processador P_i executa localmente a instância $\langle C_i'', k_i'' \rangle$ referente à folha y_i da árvore T usando o algoritmo de Niedermeier e Rossmanith [1]. Esta fase será encerrada ou quando encontrar um *hitting set* de tamanho menor ou igual a k para a coleção C de S; ou se todos os processadores percorreram toda a árvore significando que não foi possível encontrar um *hitting set* válido.

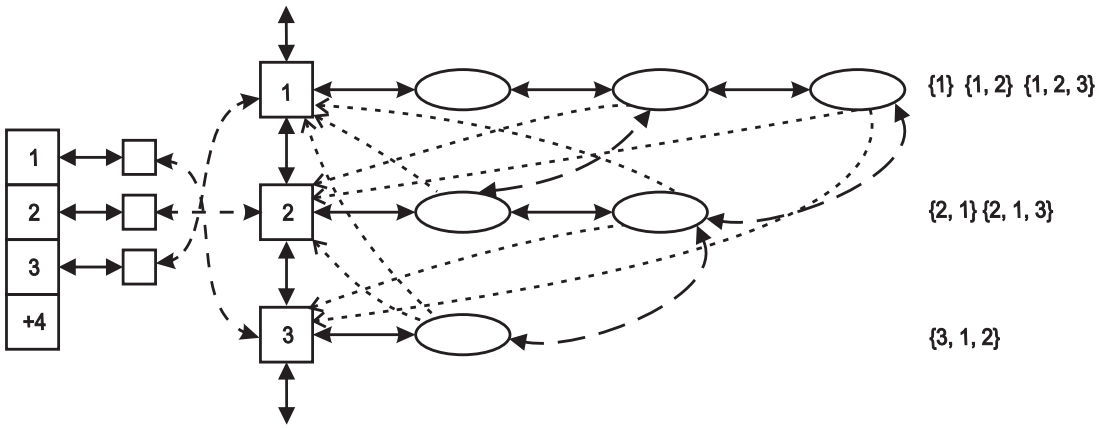


Figura 5.2: As listas de graus e as listas de adjacências representam a coleção C de subconjuntos, de um conjunto finito S, de tamanho no máximo 3.

Capítulo 6

Resultados Experimentais

6.1 Introdução

Neste capítulo são apresentados os resultados experimentais obtidos com a implementação BSP/CGM do algoritmo de Cheetham *et al.* [2] com a utilização do algoritmo de Niedermeier e Rossmanith [1] nas fases de redução ao núcleo do problema e árvore limitada de busca, na solução do problema do 3-*Hitting Set*. Tal implementação utiliza as estruturas de dados apresentadas no Capítulo 5. Na Seção 6.2 são apresentados o ambiente computacional e a metodologia utilizada para computar os resultados obtidos nos experimentos. Na Seção 6.3 são mostradas as coleções C de subconjuntos, de conjuntos finitos S, de tamanho no máximo 3 de entrada utilizadas nos experimentos. Por fim, na Seção 6.4 serão discutidos os resultados obtidos nas implementações comparando os ambientes computacionais.

6.2 Ambiente Computacional e Metodologia

Os algoritmos descritos foram implementados na linguagem C/C++ em conjunto com a biblioteca de troca de mensagens MPI. O MPI foi escolhido por ser uma interface padrão de troca de mensagens para programação paralela, propiciando a portabilidade e a compatibilidade. Além de favorecer o desenvolvimento de algoritmos paralelos no modelo BSP/CGM. O MPI inclui rotinas de comunicação ponto-a-ponto, rotinas de comunicação coletivas, computação global e sincronizações.

Os algoritmos BSP/CGM foram executados em um *cluster* composto por 12 nós: uma máquina AMD Athlon(tm) 1800+ 1GB de RAM; uma máquina Intel(R) Pentium(R) 4 CPU 1.70GHz 1GB de RAM; três máquinas Pentium IV 2.66GHz 512MB; uma máquina Pentium IV 2.8GHz 512MB; uma máquina Pentium IV 1.8GHz 480MB; quatro máquinas AMD Athlon(tm) 1.66GHz 480MB; uma máquina AMD Sempron(tm) 2600+ 480 MB. Os nós estavam conectados por um *fast-Ethernet switch* de 1Gb. Cada nó executava o sistema operacional Linux Fedora 6 com g++ 4.0 e MPI/LAM 7.1.2. Na Grade Computacional foi utilizado o mesmo ambiente descrito acima, com o *middleware* Grade InteGrade versão 0.2 RC4.

Todos os valores apresentados nos gráficos correspondem ao tempo médio de 30 experimentos. Os tempos obtidos são apresentados em segundos, incluindo o tempo para leitura dos dados de

entrada, desalocação das estruturas utilizadas e impressão da saída, e o tempo entre o início do primeiro processo até o final do último processo. O tempo de execução sequencial corresponde ao tempo de execução da implementação paralela em uma máquina do *cluster* com um único processador.

6.3 Dados de entrada

As coleções C de subconjuntos, de conjuntos finitos S, de tamanho no máximo 3 utilizadas nos testes foram gerados de forma aleatória. O programa gera os subconjuntos aleatoriamente a partir da especificação do intervalo dos elementos do conjunto finito S e da quantidade de conjuntos a ser gerados. Para cada conjunto a ser gerado, são sorteados, aleatoriamente, o tamanho do conjunto e depois os respectivos elementos. Note que não serão gerados os conjuntos de tamanho 1, pois optou-se em favorecer as possíveis ramificações da árvore limitada de busca.

Depois de construir a coleção C de S, o programa armazena em um arquivo texto, o conjunto finito S e a coleção C de subconjuntos.

Na Tabela 6.1 é apresentado um resumo das características das coleções C de S, utilizadas nos experimentos, que incluem o nome da coleção C de S, o número de subconjuntos (|C|), o número de elementos (|S|) e o número de subconjuntos de tamanho j ($|C_j|$), $1 \leq j \leq 3$.

C	$\|C\|$	S	$\|C^1\|$	$\|C^2\|$	$\|C^3\|$
C_A	1000	1, 2,... ,1000	0	520	480
C_B	10000	1, 2,... ,10000	0	5047	4953

Tabela 6.1: As coleções C de subconjuntos, de conjuntos finitos S, de tamanho no máximo 3 utilizadas nos experimentos e suas características. Note que $S = \{1, 2, 3, ..., n\}$ em que n é inteiro positivo; $|S|$ é o número de elementos do conjunto finito S; $|C|$ é o número de subconjuntos da coleção C; $|C^j|$ é o número de subconjuntos de tamanho j da coleção C, $1 \leq j \leq 3$;

6.4 Comparação dos Resultados

Nesta seção serão discutidos os resultados obtidos dos experimentos com as implementações do algolritmo de Niedermeier e Rossmanith [1] nas versões da árvore binária limitada de busca (Btree), da árvore ternária limitada de busca (Ttree) e da árvore t-ária limitada de busca (ND2F), $|t| > 1$. Foram comparados o desempenho das implementações para cada arquivo de teste (Tabela 6.1) executando em dois ambientes computacionais: *Cluster* e Integrade (Grade Computacional). Nas implementações Btree e ND2F, os arquivos de teste foram executados em ambientes computacionais utilizando-se 1, 2, 4 e 8 processadores; e na implementação Ttree, os arquivos de teste foram executados em ambientes computacionais utilizando-se 1, 3 e 9 processadores. O *speedup* refere-se à taxa de melhoria com relação à execução da implementação paralela em uma máquina com um único processador com os resultados obtidos da mesma implementação paralela sendo executados em um *Cluster* ou em uma Grade Computacional, com diversos nós.

Nas Figuras 6.1 e 6.2 são apresentadas os resultados das implementações, no ambiente *Cluster*, para os arquivos de teste descritos na Tabela 6.1. Observa-se que as implementações Btree

e Ttree obtiveram desempenho bastante semelhante. Pode-se presumir que a forma em que o algoritmo de Niedermeier e Rossmanith [1] constrói a árvore limitada de busca, favorece a construção da árvore binária e da árvore ternária, consequentemente, o tempo de execução de tais implementações (Btree e Ttree).

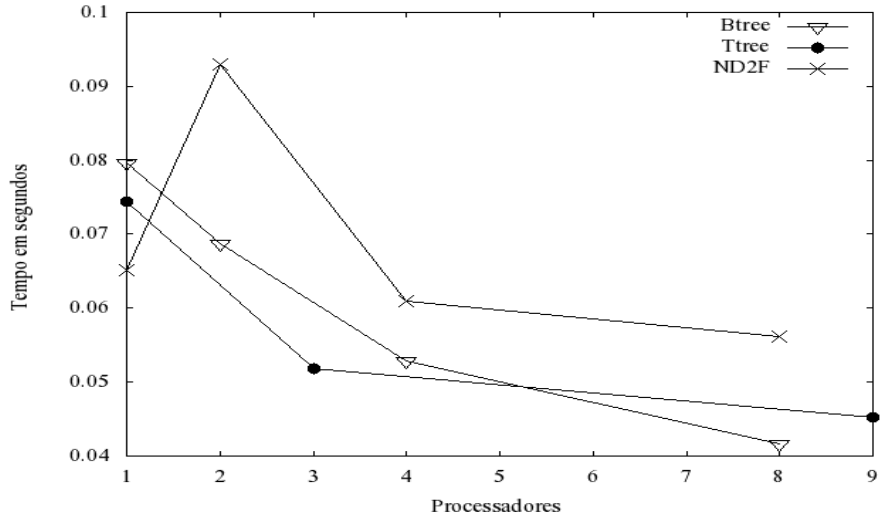


Figura 6.1: Gráfico Processadores x Tempo para o arquivo de entrada com 1000 conjuntos (Coleção C_A - Tabela 6.1).

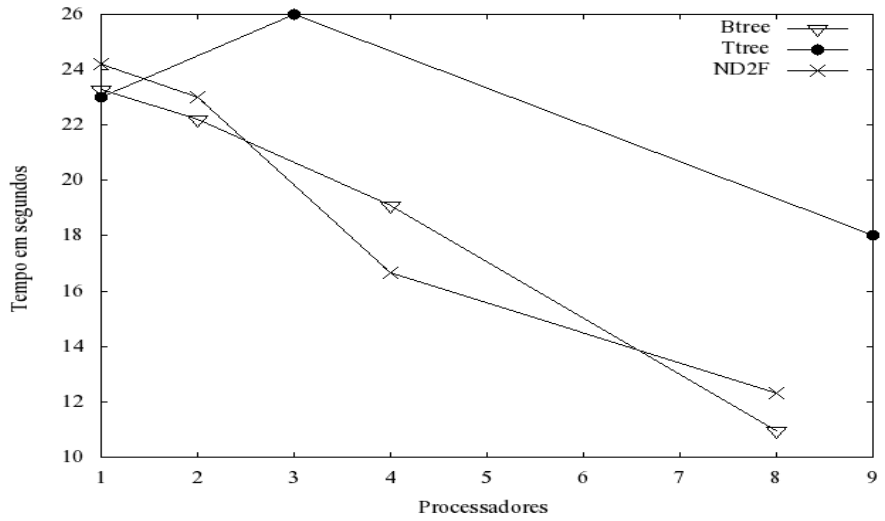


Figura 6.2: Gráfico Processadores x Tempo para o arquivo de entrada com 10000 conjuntos (Coleção C_B - Tabela 6.1).

As Figuras 6.3 e 6.4 ilustram os ganhos obtidos com a utilização de um número maior de processadores, no ambiente *Cluster*. Os resultados obtidos evidenciam que o número de processadores influenciam a quantidade de pontos iniciais de busca, consequentemente, o tempo de execução. Porém, deve-se ressaltar que a quantidade de processadores não garante a solução em um tempo menor, como poderá ser vista na Figura 6.5.

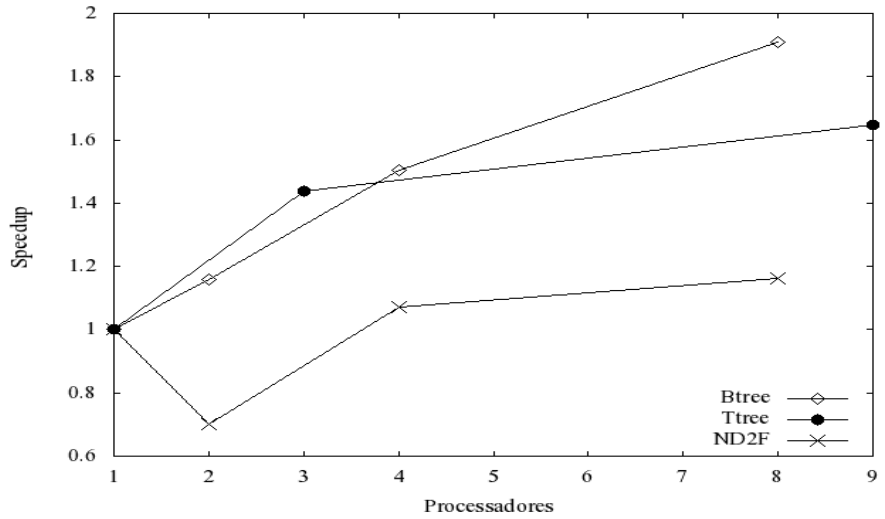


Figura 6.3: Gráfico Processadores x *Speedup* para o arquivo de entrada com 1000 conjuntos (Coleção C_A - Tabela 6.1).

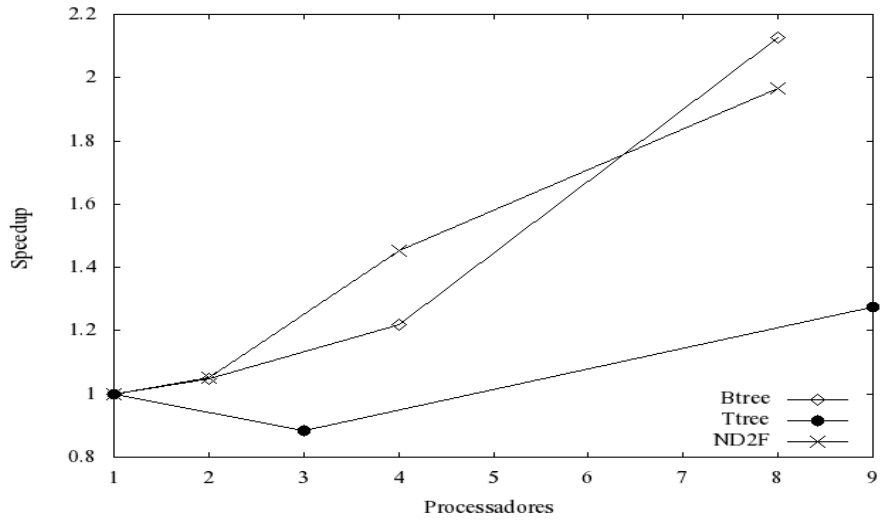


Figura 6.4: Gráfico Processadores x *Speedup* para o arquivo de entrada com 10000 conjuntos (Coleção C_B - Tabela 6.1).

Nas Figuras 6.5 e 6.6 pode-se observar o desempenho dos arquivos de teste (Tabela 6.1) em relação às implementações, no ambiente Integrade. Na Figura 6.5 pode-se notar que o custo computacional envolvido no ambiente Integrade com relação ao tamanho do arquivo de entrada, arquivos com pouco volume de dados (com pouco conjuntos), é um fator a ser considerado, pois à medida que o número de processadores aumenta, o desempenho piora. Contudo, ao utilizar um arquivo com grande volume de dados (com mais conjuntos), Figura 6.6, pode-se observar que o desempenho é considerado satisfatório e houve pouca variação de desempenho entre as implementações.

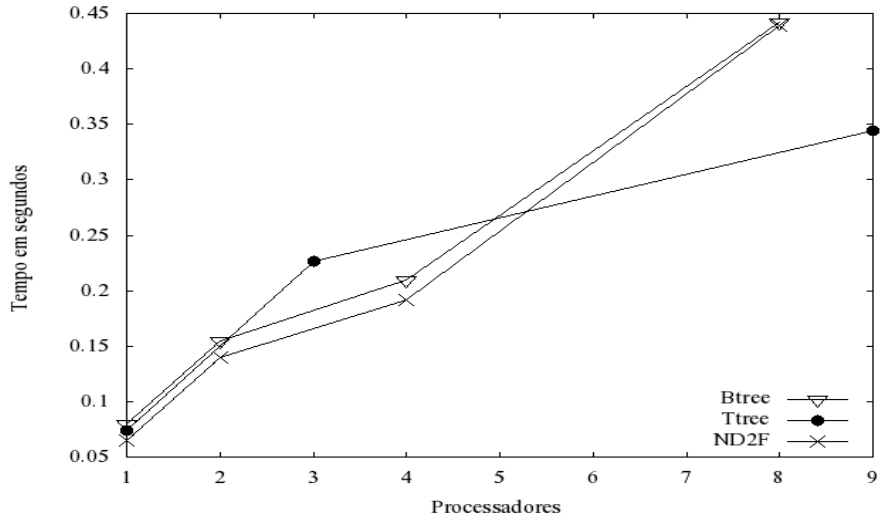


Figura 6.5: Gráfico Processadores x Tempo para o arquivo de entrada com 1000 conjuntos (Coleção C_A - Tabela 6.1).

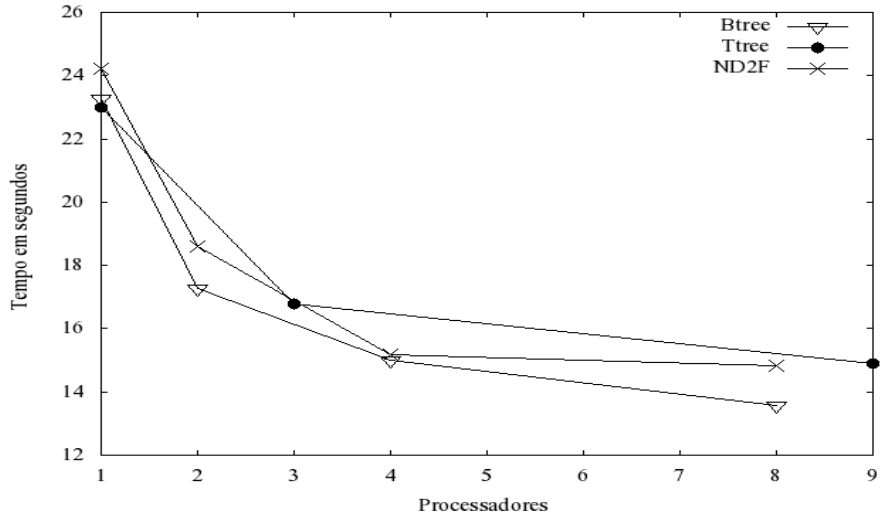


Figura 6.6: Gráfico Processadores x Tempo para o arquivo de entrada com 10000 conjuntos (Coleção C_B - Tabela 6.1).

As Figuras 6.7 e 6.8 apresentam os ganhos obtidos com a utilização de um número maior de processadores, no ambiente Integrade. Conforme discutido anteriormente (Figuras 6.5 e 6.6), a Figura 6.7 ilustra também, a medida que o número de processadores aumenta com relação ao tamanho do arquivo de entrada, com pouco volume de dados, o desempenho diminui consideravelmente. Já na Figura 6.8 observa-se que o aumenta no número de processadores, influencia no desempenho, obtendo bons resultados. Deve-se, portanto, observar que no ambiente Integrade, os custos computacionais envolvidos, principalmente, o custo de comunicação, pode tornar proibitiva a execução das implementações com arquivos de entrada com pouco volume de dados.

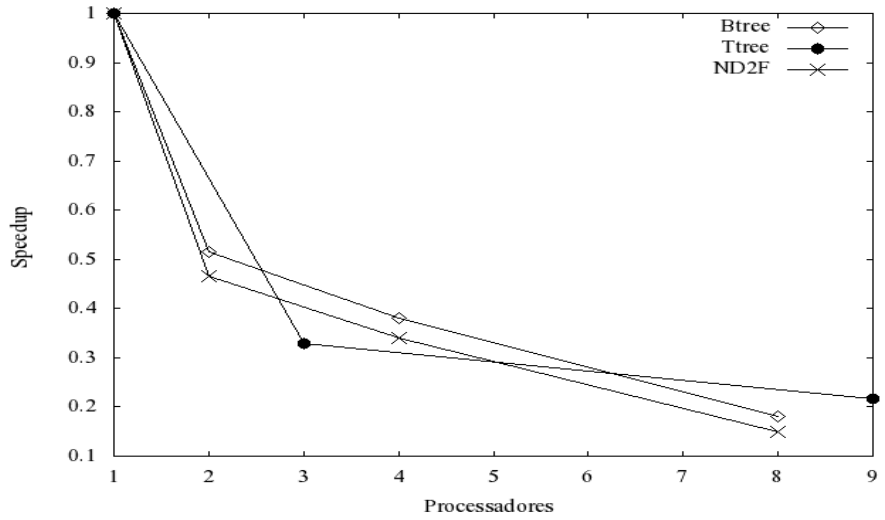


Figura 6.7: Gráfico Processadores x *Speedup* para o arquivo de entrada com 1000 conjuntos (Coleção C_A - Tabela 6.1).

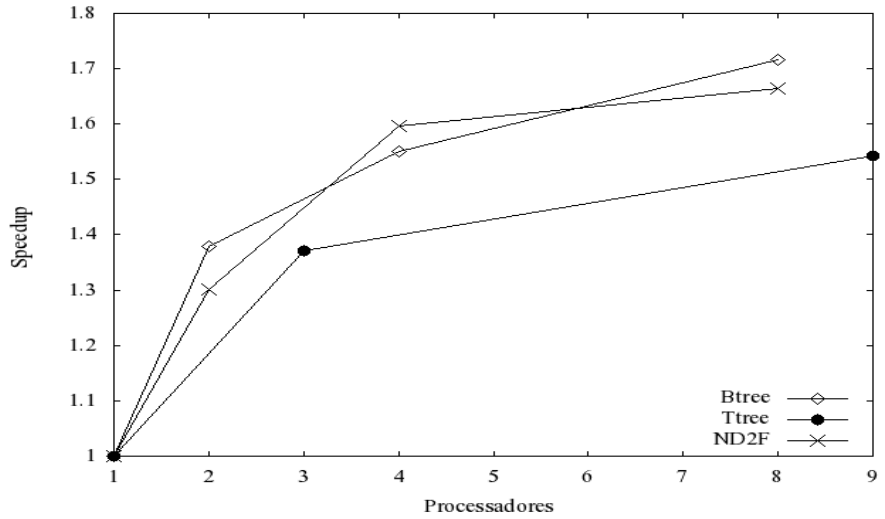


Figura 6.8: Gráfico Processadores x *Speedup* para o arquivo de entrada com 10000 conjuntos (Coleção C_B - Tabela 6.1).

As Figuras 6.9 a 6.14 apresentam uma visão de desempenho geral das implementações sobre os arquivos de teste (Tabela 6.1), comparando os ambientes: *Cluster* e Integrade (Grade Computacional).

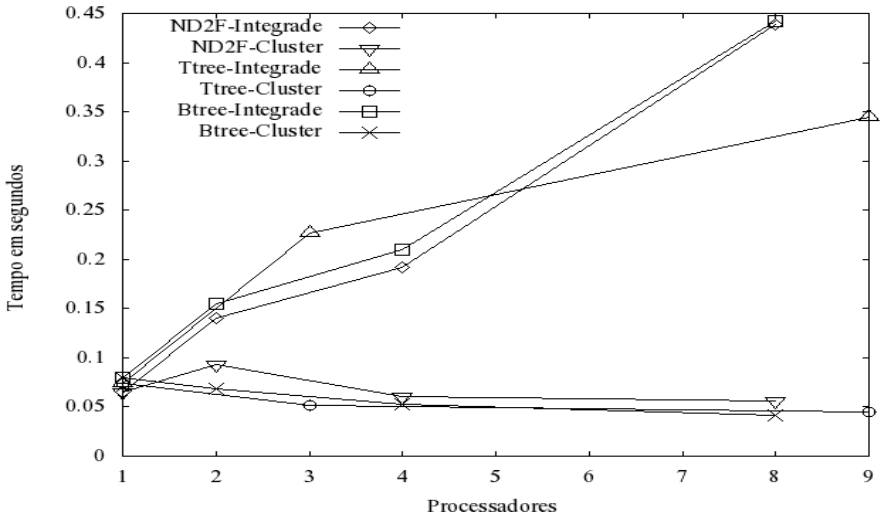


Figura 6.9: Gráfico Processadores x Tempo para o arquivo de entrada com 1000 conjuntos (Coleção C_A - Tabela 6.1).

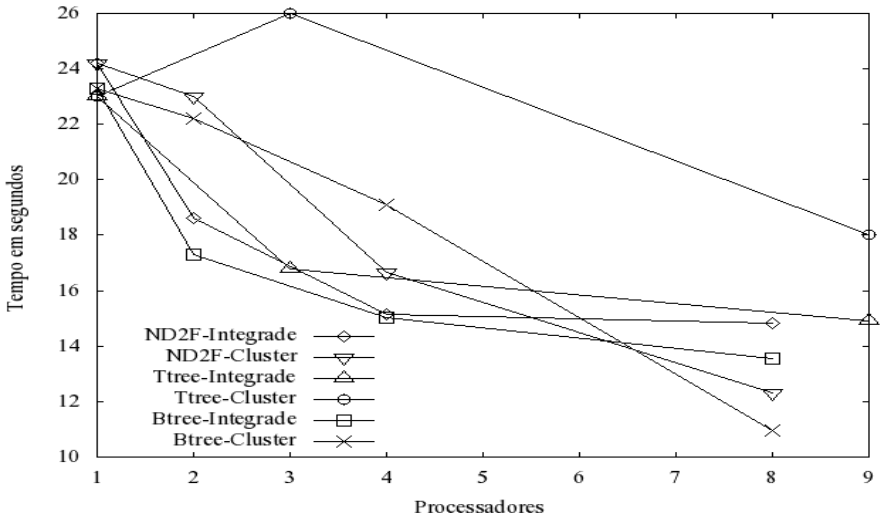


Figura 6.10: Gráfico Processadores x Tempo para o arquivo de entrada com 10000 conjuntos (Coleção C_B - Tabela 6.1).

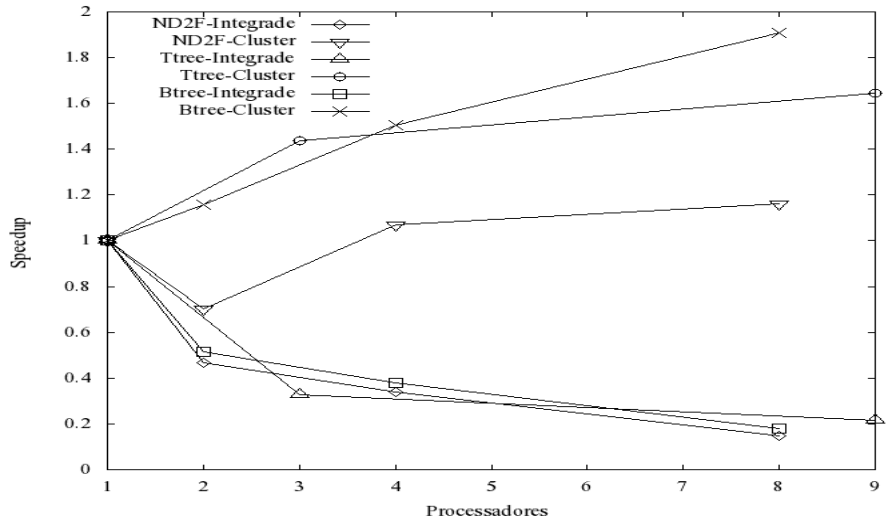


Figura 6.11: Gráfico Processadores x *Speedup* para o arquivo de entrada com 1000 conjuntos (Coleção C_A - Tabela 6.1).

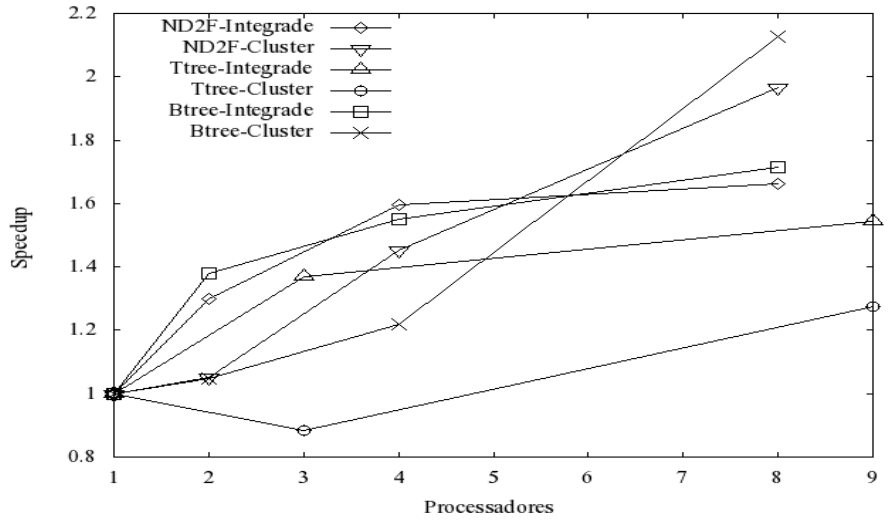


Figura 6.12: Gráfico Processadores x *Speedup* para o arquivo de entrada com 10000 conjuntos (Coleção C_B - Tabela 6.1).

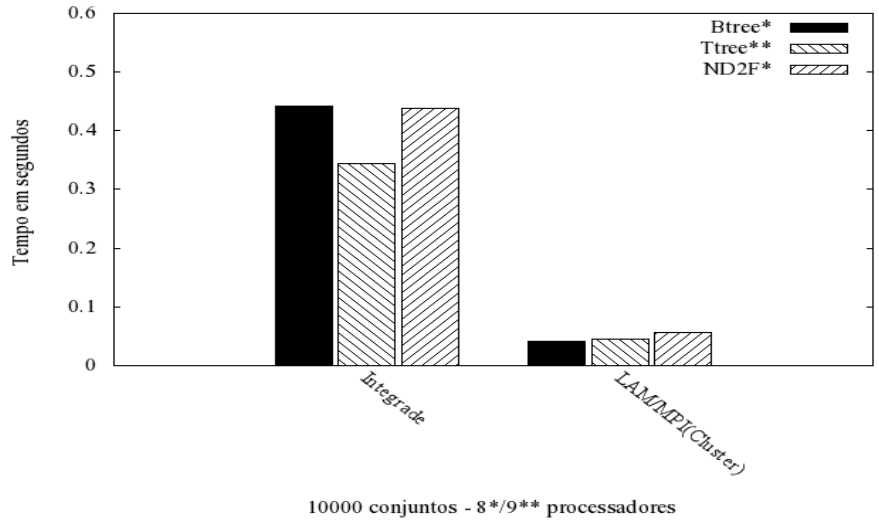


Figura 6.13: Gráfico InteGrade x LAM/MPI (*Cluster*). Execuções das implementações Btree e BD2F em 8* máquinas reais e da implementação Ttree em 9** máquinas reais, para o arquivo de entrada com 1000 conjuntos (Coleção C_A - Tabela 6.1).

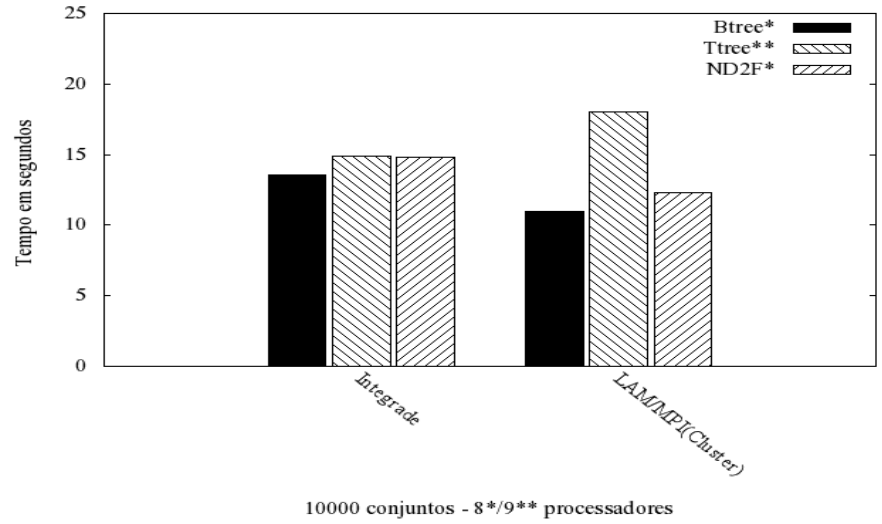


Figura 6.14: Gráfico InteGrade x LAM/MPI (*Cluster*). Execuções das implementações Btree e BD2F em 8* máquinas reais e da implementação Ttree em 9** máquinas reais, para o arquivo de entrada com 10000 conjuntos (Coleção C_B - Tabela 6.1).

De uma maneira geral, pode-se observar que as implementações, no ambiente *Cluster*, obtiveram os melhores desempenhos em comparação ao ambiente Integrade, destacando-se a implementação Btree, como mostram as Figuras 6.9 a 6.14. Um fator a considerar é a topologia da árvore limitada de busca, em que algoritmo de Niedermeier e Rossmanith [1] favorece a construção da árvore binária limitada de busca, devido à forma em que a árvore é ramificada. Além disso, na implementação ND2F pode ocorrer de alguns processadores ficarem sobrecarregados e outros subutilizados, devido ao fato da árvore ser construída de forma não determinística, prejudicando o resultado do experimento.

Deve-se destacar que os resultados são promissores para quase todas as situações de testes e as implementações mantiveram comportamentos estáveis, em ambos os ambientes computacionais.

Outro ponto a destacar, para arquivos de entrada com grande volume de dados, espera-se que o aumento do número de processadores possa trazer bons resultados, para ambos os ambientes computacionais.

Capítulo 7

Conclusão

O desenvolvimento de novos métodos para lidar com a intratabilidade visam contornar a falta de algoritmos eficientes que apresentem soluções exatas para todas as instâncias de problemas NP-Completos. A Complexidade Parametrizada vem sendo uma alternativa para resolver instâncias de problemas NP-Completos possibilitando bons resultados em relação ao desempenho e a garantia de soluções exatas. O foco deste método é procurar soluções exatas para problemas difíceis quando for possível fixar um parâmetro relativo ao tamanho da entrada, cuja a entrada pode ser dividida em duas partes: a parte principal e o parâmetro. Com isso, o crescimento exponencial da aparentemente inevitável explosão combinatorial dos problemas da classe de problemas tratáveis por parâmetro fixo (FPT) pode ser restrito ao parâmetro, tornando o método atrativo para pequenos valores do parâmetro.

A existência de tais algoritmos FPT eficientes, para um pequeno intervalo do parâmetro, na solução de instâncias de problemas NP-Completos, não implica que todos os problemas possam ser resolvidos com a utilização da Complexidade Parametrizada. Além disso, a complexidade exponencial do parâmetro ainda pode resultar em custos proibitivos. Com a utilização do paralelismo, através da combinação do modelo BSP/CGM, vem favorecendo o emprego de instâncias ainda maiores na solução de problemas FPT que eram, antes, impraticáveis sequencialmente.

O objetivo deste trabalho foi a utilização da computação paralela associada à Complexidade Parametrizada e aos problemas de otimização combinatorial. Foram desenvolvidos algoritmos FPT paralelos, no modelo BSP/CGM, para problemas de otimização combinatorial sendo estudado o problema do 3-*Hitting Set*.

Foram estudados três algoritmos FPT sequenciais para o problema do 3-*Hitting Set*, os algoritmos de Niedermeier e Rossmanith [1], Fernau [3] e Abu-Khzam [4], e um algoritmo FPT paralelo, o algoritmo de Cheetham *et al.* [2] para o problema da k-Cobertura por Vértices, sendo substituídos as fases do algoritmo de Cheetham *et al.* [2] pelo algoritmo de Niedermeier e Rossmanith [1] para resolver o problema do 3-*Hitting Set*. Contudo, devido às várias dificuldades nas implementações dos algoritmos de Fernau [3] e Abu-Khzam [4], em adaptar as estruturas de dados utilizadas pelas técnicas empregadas por esses algoritmos ao algoritmo de Cheetham *et al.* [2], os resultados experimentais preliminares obtidos foram considerados insatisfatórios com relação aos tempos esperados em comparação à implementação do algoritmo de Niedermeier e Rossmanith [1], sendo, portanto, descartados.

Os desempenhos obtidos de nossas implementações mostraram que a combinação do método

FPT, do paralelismo, e de *clusters*/grades computacionais apresentaram ganhos expressivos na solução de problemas FPT. O *speedup* calculado variou de 1,04 a 2,16 no melhor caso. Outro ponto a destacar, apesar dos resultados no ambiente *Cluster* serem melhores que no ambiente Integrade, os resultados em ambos os ambientes computacionais apresentaram resultados promissores, principalmente, quando foram utilizados arquivos de entrada com grandes volumes de dados. Nestas condições, os *speedup* obtidos mostraram uma evolução satisfatório de desempenho das implementações.

Trabalhos futuros incluem: (i) a substituição do método de redução ao núcleo do problema do algoritmo de Niedermeier e Rossmanith [1] pelo algoritmo de Abu-Khzam [4], por ser mais eficiente; (ii) a substituição do algoritmo da segunda fase da árvore limitada de busca pelo algoritmo de Fernau [3]; (iii) a aplicação da heurística *priorities* (do algoritmo de Fernau [3]), nos algoritmos da árvore limitada de busca, nos casos em que as escolhas dos elementos e conjuntos não sejam alterados pelas regras impostas pelos algoritmos de Niedermeier e Rossmanith [1] e Fernau [3]; e (iv) a utilização de escolhas aleatórias, na fase da árvore limitada de busca, nos casos em que elementos e conjuntos não seguem qualquer regra ou que não alterem regras impostas pelos algoritmos de Niedermeier e Rossmanith [1] e Fernau [3].

Bibliografia

[1] R. Niedermeier e P. Rossmanith. An efficient fixed-parameter algorithm for 3-Hitting Set. *J. Discrete Algorithms*, 1(1):89–102, 2003.

[2] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, e P. J. Taillon. Solving large FPT problems on coarse-grained parallel machines. *J. Comput. Syst. Sci.*, 67(4):691–706, 2003. ISSN 0022-0000.

[3] H. Fernau. A top-down approach to search-trees: Improved algorithmics for 3-Hitting Set. *Algorithmica (Springer New York)*, 2008.

[4] F. N. Abu-Khzam. Kernelization algorithms for d-Hitting Set Problems. In *WADS*, páginas 434–445. 2007.

[5] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997. ISBN 0-534-94968-1.

[6] Z. Michalewicz e D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2004. ISBN 3540224947.

[7] R. Motwani e P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. ISBN 0521474655.

[8] R. G. Downey e M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24:873–921, 1995.

[9] R. G. Downey e M. R. Fellows. Fixed-parameter tractability and completeness II: Completeness for W[1]. *A Theoretical Computer Science*, 141:109–131, 1995.

[10] R. G. Downey e M. R. Fellows. Parameterized computational feasibility. In *Feasible Mathematics II*, páginas 219–244. Birkhauser Boston, 1995.

[11] R. G. Downey e M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

[12] R. G. Downey e M. R. Fellows. Parameterized complexity after (almost) 10 years: Review and open questions. In *Combinatorics, Computation & Logic, DMTCS'99 and CATS'99*, volume 21, número 3, páginas 1–33. Australian Computer Science Communications, Springer-Verlag, 1999.

[13] R. G. Downey, M. R. Fellows, e U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 de *AMS-DIMACS Proceedings Series*, páginas 49–99. 1999.

[14] R. Niedermeier. Some prospects for efficient fixed parameter algorithms. In *SOFSEM '98: Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics*, páginas 168–185. Springer-Verlag, London, UK, 1998. ISBN 3-540-65260-4.

[15] R. Niedermeier e P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Inf. Process. Lett.*, 73(3-4):125–129, 2000. ISSN 0020-0190.

[16] F. K. H. A. Dehne, M. R. Fellows, F. A. Rosamond, e P. Shaw. Greedy localization, iterative compression, modeled crown reductions: New FPT techniques, an improved algorithm for set splitting, and a novel 2k kernelization for vertex cover. In *IWPEC*, páginas 271–280. 2004.

[17] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, 2006. ISBN 0-19-856607-7.

[18] F. Dehne, A. Fabri, e A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. In *Proceedings of the ACM 9th Annual Computational Geometry*, páginas 298–307. 1993.

[19] L. G. Valiant. A bridging model for parallel computation. *CACM: Communications of the ACM*, 33, 1990.

[20] H. Mongelli e R. C. Sakamoto. Implementações de algoritmos paralelos FPT para o problema da k-cobertura por vértices utilizando clusters e grades computacionais. *Workshop em Sistemas Computacionais de Alto de Desempenho (WSCAD)*, 1:155–162, 2007.

[21] E. J. Hanashiro. *O problema da k-cobertura por vértices: uma implementação FPT no modelo BSP/CGM*. mestrado, UFMS, Campo Grande, MS, Março 2004.

[22] M. R. Garey e D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

[23] B. Chor, M. Fellows, e D. W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *WG*, páginas 257–269. 2004.

[24] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, e A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *CIAC*, páginas 320–331. 2006.

[25] R. S. Anand, T. Erlebach, A. Hall, e S. K. Stefanakos. Call control with k rejections. In *SWAT '02: Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, páginas 308–317. Springer-Verlag, London, UK, 2002. ISBN 3-540-43866-1.

[26] D. Bryant, M. Fellows, V. Raman, e U. Stege. On the parameterized complexity of MAST and 3-Hitting Set. 1998. Unpublished manuscript.

[27] A. Goldchleger, F. Kon, A. Goldman, e M. Finger. Integrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. In *Middleware Workshops*, páginas 232–234. 2003.

[28] S. Götz. *Communication-Efficient Parallel Algorithms for Minimum Spanning Tree Computation*. Tese de doutoramento, University of Paderborn, Maio 1998.

[29] E. N. Cáceres, H. Mongelli, e S. W. Song. Topics in parallel algorithms using CGM/MPI. SBAC-PAD 2002 (Tutorial), 2002.

[30] H. R. Lewis e C. H. Papadimitriou. *Elementos de Teoria da Computação*. Bookman, Porto Alegre, RS, 2 edição, 2004. ISBN 85–7307–534–1.

[31] M. Wahlström. *Algorithms, Measures, and Upper Bounds for Satisfiability and Related Problems*. doutorado, Linköping, Linköping, Sweden, Março 2007.

[32] D. Ruchkys e S. Song. A parallel approximation hitting set algorithm for gene expression analysis. In *SBAC-PAD '02: Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SCAB-PAD'02)*, página 75. IEEE Computer Society, Washington, DC, USA, 2002. ISBN 0-7695-1772-2.

[33] F. Kuhn, P. von Rickenbach, R. Wattenhofer, E. Welzl, e A. Zollinger. Interference in cellular networks: The minimum membership set cover problem. In *COCOON*, páginas 188–198. 2005.

[34] J. A. Jones e M. J. Harrold. Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Trans. Software Eng.*, 29(3):195–209, 2003.

[35] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley Publishing Company, 1989.

[36] T. H. Cormen, C. E. Leiserson, e R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw Hill, 1996.

[37] C. H. Papadimitriou e K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc, 1982.

[38] R. Downey e M. Fellows. Fixed-parameter tractability and completeness III: some structural aspects of the w hierarchy. páginas 191–225, 1993.

[39] K. R. Abrahamson, R. G. Downey, e M. R. Fellows. Fixed-parameter tractability and completeness iv: On completeness for w[p] and pspace analogues. *Ann. Pure Appl. Logic*, 73(3):235–276, 1995.

[40] R. G. Downey, M. R. Fellows, e F. K. H. A. Dehne, editor. *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 de *Lecture Notes in Computer Science*. Springer, 2004. ISBN 3-540-23071-8.

[41] R. Balasubramanian, M. R. Fellows, e V. Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, 65:163–168, 1998.

[42] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[43] Zbigniew Lonc e Miroslaw Truszczyński. Fixed-parameter complexity of semantics for logic programs. *ACM Trans. Comput. Logic*, 4:91–119, January 2003. ISSN 1529-3785.

[44] L. Cai, J. Chen, R. G. Downey, e M.l R. Fellows. Advice classes of parameterized tractability. *Ann. Pure Appl. Logic*, 84(1):119–138, 1997.

[45] U. Stege. *Resolving Conflicts in Problems from Computacional Biology*. doutorado, ETH, Zürich, Switzerland, 1999.

[46] J. Gramm e R. Niedermeier. Minimum quartet inconsistency is fixed parameter tractable. In *Proc. 12th CPM*, páginas 241–256. Springer–Verlag LNCS 2089, 2001.

[47] J. Gramm e R. Niedermeier. Breakpoint medians and breakpoint phylogenies: A fixed-parameter approach. In *ECCB*, páginas 128–139. 2002.

[48] H. L. Bodlaender, R. G. Downey, M, R. Fellows, e D. Hermelin. On problems without polynomial kernels (extended abstract). In *ICALP '08: Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, páginas 563–574. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-70574-1.

[49] X. Cai. Linear kernelizations for restricted 3-hitting set problems. *CoRR*, abs/0809.0257, 2008.

[50] G. Ausiello, A. D'Atri, e M. Protasi. Structure preserving reductions among convex optimization problems. *J. Comput. Syst. Sci.*, 21(1):136–153, 1980.

[51] C. Berge. *Hypergraphs*. North Holland, 1989. ISBN 0444874895.

[52] R. Z. Norman e M. O. Rabin. An algorithm for a minimum cover of a graph. *Proc. Amer. Math. Soc.*, 10:315–319, 1959.

[53] J. Edmonds. Paths, Trees, and Flowers. *Canad. J. Math.*, 17:449–467, 1965.

[54] J. F. Buss e J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, 1993.

[55] A. Chan e F. K. H. A. Dehne. A note on coarse grained parallel integer sorting. *Parallel Processing Letters*, 9(4):533–538, 1999.