Propriedade dos Uns Consecutivos e Árvores PQR

Guilherme Pimentel Telles

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Propriedade dos Uns Consecutivos e Árvores PQR

Guilherme Pimentel Telles[1]

Dezembro de 1997

Banca Examinadora:

- João Meidanis (Orientador)
- Oscar Porto
  Departamento de Engenharia Elétrica – PUC-Rio
- Arnaldo Vieira Moura
  Instituto de Computação – Unicamp
- João Carlos Setubal (Suplente)
  Instituto de Computação – Unicamp

[1]Financiado pelo CNPq e pela FAPESP.

Propriedade dos Uns Consecutivos e Árvores PQR

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Guilherme Pimentel Telles e aprovada pela banca examinadora.

Campinas, 19 de dezembro de 1997.

João Meidanis (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

$\boxed{\text{Substituir pela página de assinaturas}}$

Agradecimentos

Ao Professor João Meidanis, *o* orientador, pela convivência agradável e grandiosa em ensinamentos.

Aos Professores Manoel Palhares, Cláudia Ferris e Pasteur Miranda Jr., pelo pontapé inicial.

Aos meus pais, irmãs, avós, tios, primos, amigos, amores, pelo apoio. Não é preciso de todos eles escrever os nomes aqui, pois já estão guardados em local seguro: no coração.

Ao meu pai, ao CNPq e à FAPESP pelo imprescindível apoio financeiro.

"Acreditar é o contrário de saber. Se você sabe alguma coisa, isto é, se você a viu, não precisa acreditar nela. Mas se você acredita nela, isto significa que não sabe, que não viu nada."

(Alain Daniélou)

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Símbolos, Operadores e Definições

Resumo

Neste trabalho formalizamos as Árvores PQR de Meidanis e Munuera e seu relacionamento com a propriedade dos uns consecutivos e com as Árvores PQ de Booth e Lueker. Mostramos que uma árvore PQR construída para uma coleção $\mathcal{C}$ de subconjuntos de um universo U é capaz de armazenar todas as permutações de U que verificam a propriedade dos uns consecutivos. Apresentamos dois algoritmos para construir as árvores PQR, um recursivo e outro não recursivo, e alguns problemas relativos à propriedade e às coleções de conjuntos que podem ser resolvidos através destas árvores. Analisamos, ainda, um conjunto de aplicações das Árvores PQ e consideramos a possibilidade de empregar as árvores PQR.

Abstract

In the present work we formalize Meidanis and Munuera's PQR trees and their relationship with the Consecutive Ones Property and with Booth and Lueker's PQ trees. We show that a PQR tree built for a collection $\mathcal{C}$ of subsets of a ground set U is able to store all permutations of U that verify the consecutive ones property. We introduce two algorithms that build the PQR trees, a recursive and a non recursive one, and some problems related to the consecutive ones property and to collections of sets that can be solved using them. We analyze some applications of the PQ trees and inspect the useness of the PQR trees.

Capítulo 1

Introdução

Uma coleção $\mathcal{C}$ de subconjuntos de um universo U possui a propriedade dos uns consecutivos — P1C se existir uma permutação α dos elementos de U tal que os elementos de todo conjunto $A \in \mathcal{C}$ aparecem consecutivamente em α.

Uma árvore PQR, definida originalmente por Meidanis e Munuera [MM96], é uma árvore enraizada e de aridade variável com três tipos diferentes de nós internos: P, Q e R. Suas folhas são elementos de um universo U.

Neste trabalho formalizamos as árvores PQR e mostramos que uma árvore sempre pode ser construída para uma coleção $\mathcal{C}$ de subconjuntos de U. Mostramos também que se a árvore construída não possui nós do tipo R, então a permutação de U dada pelas suas folhas verifica a P1C para a coleção $\mathcal{C}$. Desta forma estabelecemos a relação entre P1C, árvores PQR e árvores PQ de Booth e Lueker [BL76].

Apresentamos dois algoritmos para construir as árvores PQR, um recursivo e um não recursivo, baseados em novas coleções definidas por Meidanis e Munuera: $\overline{\mathcal{C}}$ e $\mathcal{C}^{\perp}$. Esses algoritmos não usam comparação de padrões como faz o algoritmo para as árvores PQ, evitando a complexidade de implementação dessa técnica. Analisamos também o custo do problema incremental, motivados pela construção de um algoritmo de tempo real.

Usando as árvores PQR, resolvemos alguns problemas motivados pela propriedade dos uns consecutivos e pelas coleções de conjuntos, supondo um modelo orientado a objetos para representá-las. Analisamos, ainda, um grupo de aplicações das árvores PQ e consideramos sua aplicabilidade às árvores PQR, e a possibilidade de usar as informações fornecidas pelos nós R quando a coleção não possui a P1C.

Este trabalho está organizado da seguinte forma. No Capítulo 2 apresentamos a propriedade dos uns consecutivos e motivação para o seu estudo. No Capítulo 3 formalizamos as árvores PQR e suas relações com a P1C, além de apresentar alguns resultados adicionais. No Capítulo 4 apresentamos os algoritmos para construir as árvores e analisamos o problema incremental. No Capítulo 5 resolvemos problemas relacionados à propriedade dos uns consecutivos e às coleções de conjuntos e analisamos um grupo de aplicações das árvores PQ. Finalmente, no Capítulo 6, apresentamos nossas considerações finais.

Capítulo 2

Propriedade dos Uns Consecutivos

Neste capítulo definimos a propriedade dos uns consecutivos, apresentamos motivação para o seu estudo e alguns problemas relacionados à propriedade que gostaríamos de solucionar. Introduzimos, ainda, definições e conceitos que serão necessários ao longo do texto.

2.1 Propriedade dos Uns Consecutivos

Uma matriz binária $n \times m$ (uma matriz cujas entradas são zeros ou uns com n linhas e m colunas) possui a **propriedade dos uns consecutivos** — P1C para colunas se suas linhas puderem ser permutadas de forma que os uns apareçam consecutivamente em suas colunas. Dizemos que uma permutação das linhas de uma matriz binária que verifica a P1C é uma **permutação válida**. Podemos observar que se uma matriz possui a P1C, então pelo menos duas permutações de suas linhas são válidas.

Exemplo 1 *A matriz na Figura 1(a) possui a propriedade dos uns consecutivos para colunas, como podemos observar após reorganização adequada das linhas, na Figura 1(b). Já a matriz na Figura 2 não possui a P1C.*

$$\begin{array}{c} a \\ b \\ c \\ d \end{array} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{array}{c} d \\ b \\ a \\ c \end{array} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

(a) (b)

Figura 1: Matriz que possui a P1C para colunas.

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figura 2: Matriz que não possui a P1C para colunas.

A P1C para linhas é definida de forma análoga: uma matriz possui a P1C para linhas se suas colunas puderem ser permutadas de forma que os uns nas linhas apareçam consecutivamente. Podemos perceber que uma definição é equivalente à outra se tomarmos a transposta da matriz.

Exemplo 2 *A matriz na Figura 3(a) possui a P1C para linhas. Na Figura 3(b), temos uma permutação válida das colunas daquela matriz.*

$$\begin{array}{c} \begin{array}{cccc} p & q & r & s \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array} \qquad \begin{array}{c} \begin{array}{cccc} s & q & p & r \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

(a) (b)

Figura 3: Matriz que possui a P1C para linhas.

2.2 Motivação

Vários problemas de interesse teórico e aplicado podem ser modelados através de uma matriz binária. Para alguns deles, a solução consiste exatamente em encontrar uma permutação válida para as linhas da matriz. Vamos citar alguns deles. Durante a descrição dos problemas e no resto do texto vamos adotar a mesma nomenclatura de teoria dos grafos de Bondy e Murty [BM76].

2.2.1 Reconhecimento de grafos intervalo

Os grafos intervalo podem ser reconhecidos pela verificação da P1C. Um grafo G é um grafo intervalo se seus vértices puderem ser colocados em correspondência um para um com um conjunto de intervalos sobre a reta real de forma que dois vértices em G sejam adjacentes se e somente se os intervalos correspondentes se interceptarem. O seguinte teorema, de Fulkerson e Gross [FG65], relaciona a P1C e o reconhecimento de grafos intervalo:

Teorema 3 *Um grafo $G = (V, E)$ é um grafo intervalo se e somente se a matriz incidência de cliques maximais $\times$ vértices possui a P1C para colunas.*

A matriz incidência de *cliques maximais* $\times$ *vértices* tem uma linha para cada clique maximal de G, uma coluna para cada vértice, o um na interseção de uma linha com coluna se o vértice representado pela coluna pertence ao clique representado pela linha e o zero, caso contrário.

2.2.2 Dobramento ótimo de PLAs

Um PLA (Programmable Logical Array) é um conjunto de portas lógicas AND e OR que implementa uma ou mais funções Booleanas na forma de soma de produtos. Este dispositivo pode ser visto como uma matriz, chamada matriz personalidade, onde cada coluna representa ora uma variável para a função, ora os resultados das funções que se quer implementar. Nas colunas que representam variáveis, cada linha calcula um produto das variáveis que têm o um na interseção com a linha ou seja, uma porta AND. Nas colunas que representam resultados das funções o cálculo é completado fazendo uma soma. O valor um na interseção de uma linha e uma coluna representa uma porta OR. O número de portas colocadas realmente em um PLA é bem menor que o número de interseções entre linhas e colunas e então, para minimizar seu tamanho, há interesse em construir uma matriz compacta que represente as mesmas funções, sobrepondo colunas sem que haja sobreposição de portas, dando origem ao problema do dobramento de PLAs. Resultados de Ferreira e Song [FS92] estabelecem que se uma submatriz especial da matriz personalidade, chamada matriz personalidade reduzida, possuir a P1C para colunas então o dobramento do PLA será ótimo.

2.2.3 Propriedade de recuperação consecutiva

Outra aplicação da P1C é na organização de arquivos. Suponhamos um conjunto R de registros organizados em um sistema de arquivos e um conjunto de consultas C tal que a cada consulta $c \in C$ está associado um subconjunto de R. Vamos supor que o meio de armazenamento permite que os registros possam ser armazenados de forma que o tempo gasto para acessar dois registros distintos é uma função monotonicamente crescente da distância entre eles. Se existe uma organização tal que os registros associados a cada consulta em C podem ser armazenados em posições adjacentes sem repetição de registros, dizemos que o conjunto de consultas possui a **propriedade de recuperação consecutiva** — PRC. Podemos perceber as vantagens dessa organização. Já que nenhum registro é replicado, não há necessidade de um mecanismo para garantir a consistência das diversas cópias. Além disso, o tempo gasto para recuperar os registros associados a cada consulta é proporcional ao número de registros. Uma matriz binária M pode ser construída para um par (R, C) associando a cada registro uma linha, a cada consulta uma coluna, atribuindo o valor um à interseção de uma linha com uma coluna se e somente se o registro está associado à consulta e, caso contrário, atribuindo zero à interseção. Em seu trabalho, Ghosh [Gho72] define esta propriedade e estabelece que um par (R, C) possui a PRC se e somente se a matriz M construída para o par possui a P1C para colunas.

2.2.4 Mapas físicos de DNA

Um dos problemas da biologia molecular é encontrar a posição correta de certos marcadores em um cromossomo, que pode ser visto como uma cadeia de caracteres. Marcadores, em geral, são seqüências pequenas e bem definidas. As informações sobre os marcadores são obtidas observando sobreposições de fragmentos, chamados clones, obtidos a partir de diversas cópias do cromossomo. Uma das técnicas para obter tais informações é a da hibridização. Nesta técnica, os marcadores são pequenas seqüências de DNA chamadas sondas. A hibridização consiste em verificar quais dentre um conjunto de sondas ocorrem nos clones. Esta informação permite construir uma matriz binária $clones \times sondas$, que tem uma entrada um na interseção de uma linha com uma coluna se a sonda ocorre no clone. Se esta matriz possuir a P1C para colunas, então é possível obter sua posição relativa [MS97].

Outras aplicações da propriedade são o reconhecimento de grafos bipartidos convexos [CY91] e arqueologia [Ken69]. A P1C é também condição necessária para o reconhecimento dos grafos arco circulares [Tuc71] e suficiente para o reconhecimento de matrizes totalmente unimodulares [FG65]. Ainda outras aplicações são citadas por Munuera [Mun96], em sua dissertação de mestrado.

2.3 Definições

Além das matrizes binárias, outra forma de caracterizar a propriedade é através de uma coleção $\mathcal{C}$ de subconjuntos de um conjunto U. O termo *coleção* será utilizado por nós com o significado particular de *conjunto de conjuntos* e as coleções serão representadas por maiúsculas caligráficas, como $\mathcal{A}, \mathcal{B}, \mathcal{C}$ etc. Antes que reformulemos a P1C nesta representação alternativa, são necessárias algumas definições sobre conjuntos.

Uma **permutação** de um conjunto U, $|U| = n$, é uma bijeção $\alpha : \{1, 2, \ldots, n\} \rightarrow U$. O conjunto de todas as permutações de U é notado $Perm(U)$. O **conjunto potência** de U, formado por todos os possíveis subconjuntos de U, é notado $\mathcal{P}(U)$.

Um conjunto $A \subseteq U$ é **consecutivo** em uma permutação α se todos os elementos de A aparecem em α consecutivamente. Uma coleção $\mathcal{A}$ é consecutiva em α se cada um dos conjuntos de $\mathcal{A}$ é consecutivo em α.

Exemplo 4 *Seja $U = \{a, b, c, d, e, f\}$ e uma permutação $\alpha = fdeacb$ dos elementos de U. O conjunto $A = \{d, c, a, e\}$ é consecutivo em α, mas o conjunto $B = \{f, e, b, c\}$ não é, pois o elemento d não pertence a B.*

Dizemos então que um par $(U, \mathcal{C})$, $\mathcal{C} \subseteq \mathcal{P}(U)$, possui a P1C se existir uma permutação α de U tal que todos os conjuntos em $\mathcal{C}$ sejam consecutivos em α. Se existe uma permutação que respeita essa restrição, tal permutação de U é dita **válida** para $\mathcal{C}$.

Ainda outras caracterizações para a propriedade são o grafo bipartido [Tuc72] e o conjunto de intervalos sobre a reta real [Esw75].

Ao longo deste texto notaremos conjuntos como uma lista de seus elementos, omitindo as vírgulas e chaves da notação tradicional, por simplicidade. Por exemplo, o conjunto $A = \{a, b, c, d\}$ será notado $abcd$. Uma permutação dos elementos de um conjunto será notada da mesma forma, mas nessa última a ordem dos elementos deve ser considerada. Se os elementos de um conjunto são números, por exemplo $U = \{1, 2, 3, 4\}$, notaremos uma permutação de seus elementos separando-os, por exemplo $\alpha = 3 \cdot 2 \cdot 1 \cdot 4$.

Podemos passar de uma representação como matriz para uma como coleção de conjuntos e vice-versa. Rotulando as linhas da matriz sem repetições, o conjunto U será o conjunto de linhas da matriz. Cada conjunto de $\mathcal{C}$ será uma coluna c da matriz que conterá o elemento da linha l se a interseção de l e c for uma entrada com valor um. Por outro lado, podemos construir uma matriz para um par $(U, \mathcal{C})$ com tantas linhas quantos são os elementos de U e uma coluna para cada conjunto de $\mathcal{C}$. Colocamos uma entrada um na interseção de uma linha com uma coluna se o elemento representado pela linha pertencer ao conjunto representado pela coluna e uma entrada zero, caso contrário.

Exemplo 5 *Para a matriz na Figura 4, podemos fazer $U = abcde$. Os conjuntos $Q = abd$, $R = acd$ e $S = cde$ formam uma coleção $\mathcal{C}$.*

Observamos que essa transformação não é biunívoca, pois se a matriz possui linhas iguais a coleção construída a partir dela não terá dois conjuntos iguais como elementos.

Algumas propriedades relacionadas à P1C são a propriedade dos uns circulares, a k-consecutividade de uns e a k-consecutividade circular de uns [Rys69].

2.4 Problemas Relacionados à P1C

Algumas questões envolvendo a P1C surgem naturalmente. Dada uma instância $(U, \mathcal{C})$ para a P1C gostaríamos de solucionar os seguintes problemas:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccc} Q & R & S \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Figura 4: Matriz que originou a coleção de conjuntos $\{abd, acd, cde\}$.

1. Como verificar se uma dada permutação α de U é válida.

2. Como gerar uma permutação válida de U em relação a $\mathcal{C}$, caso exista.

3. Como gerar todas as permutações válidas de U em relação a $\mathcal{C}$.

A busca de uma solução para o Problema 2 tem motivação clara, pois implica em verificar se um par $(U, \mathcal{C})$ possui a P1C. O Problema 3 encontra motivação, dentre outras, no trabalho de Greenberg e Istrail [GI95], que mostrou ser necessário conhecer todas as possíveis soluções para a construção de um mapa físico de DNA. Para o Problema 1 podemos imaginar que um algoritmo gerenciando um sistema de arquivos baseado na propriedade de recuperação consecutiva, ao ter seu conjunto de consultas aumentado vai precisar verificar se a organização dos registros continua possuindo a P1C.

O problema de como gerar uma permutação válida de U em relação a $\mathcal{C}$ foi solucionado com o algoritmo polinomial, $O(nm^2)$, apresentado no trabalho de Fulkerson e Gross [FG65], construindo um *grafo de sobreposições estritas* (Definição 67). Outro algoritmo é apresentado por Eswaran [Esw75] para o caso em que o grafo interseção da coleção $\mathcal{C}$ é completo. O trabalho de Hsu [Hsu92] apresenta uma alternativa para a construção de Fulkerson e Gross e obtém uma solução linear, $O(n+m+r)$. Uma solução para gerar todas as permutações válidas de U em relação a $\mathcal{C}$ surgiu no trabalho de Booth e Lueker [BL76], com a introdução das árvores PQ, que podem ser construídas em tempo $O(n+m+r)$ para coleções que possuem a P1C e são capazes de representar todas as permutações válidas de forma compacta. No trabalho de Oubiña e Favini-Dubost [OFD85] encontramos um algoritmo que manipula fórmulas, de maneira semelhante à manipulação das árvores PQ. Outros trabalhos com resultados importantes são o de Tucker [Tuc72], que caracteriza a propriedade com base em configurações proibidas para as matrizes e o de Ghosh [Gho72], que apresenta propriedades das coleções que possuem a propriedade. O trabalho de Chen e Yesha [CY91] encontra uma permutação válida para uma matriz em tempo $O(\log n + \log^2 n)$ com um algoritmo paralelo, usando $O(m^2 n + m^3)$ processadores.

Neste capítulo apresentamos um algoritmo para verificar se uma dada permutação de U é válida. No Capítulo 5 apresentamos soluções para os Problemas 2 e 3 usando as árvores PQR, que são definidas no Capítulo 3.

Ao solucionar estes problemas estaremos interessados na complexidade e nas estruturas de dados utilizadas pelos algoritmos propostos. Para analisar os algoritmos usaremos a notação O [CLR89]. Consideraremos que uma instância para o problema dos uns consecutivos representada como uma matriz tem seu tamanho dado por três parâmetros: n, o número de linhas da matriz, m, o número de colunas e r, o número total de entradas com valor um. Na representação por coleção de conjuntos, o tamanho de uma instância é dado por $n = |U|$, $m = |\mathcal{C}|$ e $r = \sum_{A \in \mathcal{C}} |A|$.

Suporemos ainda que os dados na entrada são organizados da seguinte forma. O conjunto universo U será sempre igual a $\{1, 2, \ldots, n\}$ para certo n. Esta restrição não causa perda de generalidade, pois qualquer outro conjunto de elementos pode ser colocado em bijeção com $\{1, 2, \ldots, n\}$. Representando os elementos de U dessa forma, podemos representar $\mathcal{C}$ por uma lista de m vetores de inteiros, cada vetor (conjunto de $\mathcal{C}$) tem em cada posição um elemento de U. Por exemplo, o conjunto $A = \{1, 5, 7, 8\}$ será representado como um vetor A de tal forma que $A[1] = 1$, $A[2] = 5$, $A[3] = 7$ e $A[4] = 8$.

2.4.1 Verificando a validade de uma permutação α

Na Figura 5 apresentamos um algoritmo que recebe $U, \mathcal{C}$ e α como entrada e verifica se α é válida para $\mathcal{C}$. Consideraremos que α é representada como um vetor que contém em cada uma das suas n posições um dos elementos de U. Por exemplo, se $\alpha = 2 \cdot 5 \cdot 3 \cdot 1 \cdot 4$, o vetor α será $\alpha[1] = 2$, $\alpha[2] = 5$, $\alpha[3] = 3$, $\alpha[4] = 1$ e $\alpha[5] = 4$.

Antes de processar a coleção $\mathcal{C}$ o algoritmo constrói o vetor γ, a inversa de α, que dá a posição dos elementos de U em α. Para a permutação α do exemplo anterior, $\gamma[1] = 4$, $\gamma[2] = 1$, $\gamma[3] = 3$, $\gamma[4] = 5$ e $\gamma[5] = 2$. Para construir γ, o algoritmo percorre o vetor α de 1 a n, escrevendo em $\gamma[\alpha[i]]$ o valor de i, $1 \leq i \leq n$.

Depois de construir γ, o algoritmo processa cada um dos conjuntos de $\mathcal{C}$, individualmente. Durante o processamento do conjunto S, cada um dos seus elementos é considerado. Duas variáveis mantêm informação sobre a maior a menor posições em α ocupadas por elementos de S. Depois de considerar todos os elementos de S, se o valor da diferença entre esses índices for igual ao número de elementos de S menos 1, α é válida para $\mathcal{C}$. Caso contrário, α não é válida, pois algum elemento entre o maior e o menor índice não pertence a S. Nesse processamento vamos considerar que $\mathcal{C}$ não contém conjuntos vazios. Uma inspeção simples dos conjuntos pode encontrar os que são vazios e eliminá-los.

```
Função Testaα (U, C, α)
    Inteiro Maior, Menor;
    Inteiro γ[n];

    Para i = 1 até n faça
        γ[α[i]] ← i;
    Para cada conjunto S ≠ ∅ em C faça
        Menor ← γ[S[1]];
        Maior ← γ[S[1]];
        i ← 2;
        Enquanto i ≤ |S| faça
            Se γ[S[i]] < Menor então
                Menor ← γ[S[i]];
            Se γ[S[i]] > Maior então
                Maior ← γ[S[i]];
            i ← i + 1;
        Se Maior − Menor > |S| − 1 então
            Retorne(falso);
    Retorne(verdadeiro);
```

Figura 5: Algoritmo para verificar se uma permutação de U é válida.

Lema 6 *O algoritmo na Figura 5 retorna verdadeiro se e somente se a permutação α é válida para $\mathcal{C}$.*

Prova: O algoritmo se baseia no fato de que se o conjunto S é consecutivo em α, as posições de seus elementos formam um intervalo de tamanho $|S|$. Portanto $Maior - Menor = |S| - 1$ para conjuntos S consecutivos em α. □

Lema 7 *O algoritmo na Figura 5 tem complexidade $O(n + m + r)$.*

Prova: A construção de γ leva $O(n)$ operações. Para cada conjunto em $\mathcal{C}$ as variáveis *Maior* e *Menor* são inicializadas e sua consecutividade em α é verificada fazendo uma subtração, operações que custam $O(m)$ para os m conjuntos. Para cada elemento dos conjuntos é feita uma atualização das variáveis *Maior* ou *Menor*. Para os r elementos o tempo gasto será $O(r)$. O custo do algoritmo será $O(n + m + r)$. □

Capítulo 3

Árvores PQR

Neste capítulo definimos as árvores PQR e em uma série de lemas, teoremas e definições, desenvolvemos teoria necessária à formalização de sua estrutura e construção. Enfocamos em especial o relacionamento entre as árvores PQR e a P1C e apresentamos resultados adicionais relacionando árvores, coleções e permutações. Os resultados marcados com o símbolo $\diamondsuit$ são essencias para a construção do algoritmo no Capítulo 4. Os não marcados estão relacionados principalmente à P1C e às coleções de conjuntos.

3.1 Árvores PQR

As árvores PQR, apresentadas por Meidanis e Munuera [MM96], são uma generalização das árvores PQ de Booth e Lueker [BL76].

Uma árvore PQR, construída sobre U é uma árvore enraizada, de aridade variável, que tem três tipos de nós internos, P, Q e R, e que tem como folhas elementos do conjunto U, sem repetições. Os nós do tipo P são representados como um círculo, os do tipo Q como um retângulo e os do tipo R como um círculo rotulado R. Nos três casos, os filhos de nós internos são ligados a eles por arestas. As folhas são representadas pelos próprios elementos de U.

Uma árvore PQR deve obedecer às seguintes restrições à sua estrutura interna:

- Cada elemento de U aparece no máximo uma vez como folha.
- Cada nó P tem pelo menos 2 filhos.
- Cada nó Q tem pelo menos 3 filhos.
- Cada nó R tem pelo menos 3 filhos.

Definimos o **domínio** de uma árvore PQR T, $Dom(T)$, como o conjunto formado pela união de suas folhas.

Exemplo 8 *Na Figura 6 (a) e (b) temos dois exemplos de árvores PQR. O domínio da árvore PQR na Figura 6 (b) é abcdefghi. O domínio da subárvore enraizada no nó R nessa mesma árvore é abc.*

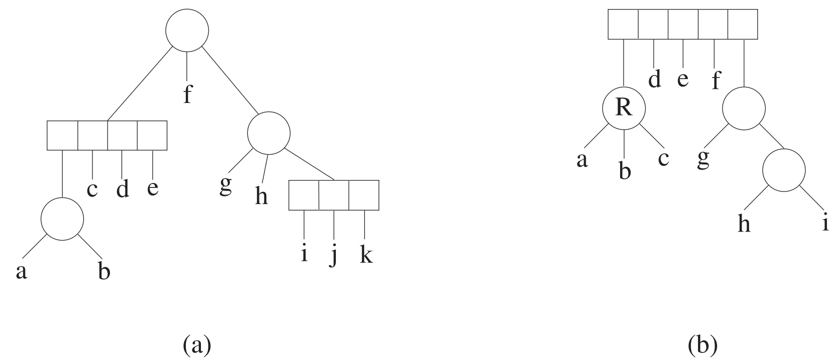

Figura 6: Exemplos de árvores PQR.

Uma árvore PQR é construída para um par $(U, \mathcal{C})$. Nas seções seguintes mostramos que se um par $(U, \mathcal{C})$ possui a P1C, a árvore PQR construída para o par não possui nós do tipo R e é capaz de armazenar todas as permutações válidas de U com relação a $\mathcal{C}$ de maneira compacta. Neste caso, ainda, uma árvore PQR será igual a uma árvore PQ construída para o mesmo par. Quando o par $(U, \mathcal{C})$ não possuir a P1C, os nós R têm potencial para fornecer informações sobre os conjuntos em $\mathcal{C}$ que obstruem a propriedade (Seção 5.5). As árvores PQ são nulas neste caso e não fornecem nenhuma informação que possa ser explorada.

O que permite que uma árvore PQR represente todas as permutações válidas de U em relação a $\mathcal{C}$ são **transformações de equivalência** que podemos aplicar sobre seus nós internos. As transformações a seguir são permitidas e determinam uma reorganização dos nós internos de um árvore PQR:

- Permutações arbitrárias dos filhos de um nó P.
- Reversão dos filhos de um nó Q.
- Permutações arbitrárias dos filhos de um nó R.

Dizemos que duas árvores PQR T e T' são **equivalentes**, $T \equiv T'$, se e somente se uma delas puder ser transformada na outra aplicando-se zero ou mais transformações de equivalência sobre seus nós internos.

Exemplo 9 *Na Figura 7 vemos duas árvores PQR equivalentes, T e T'.*

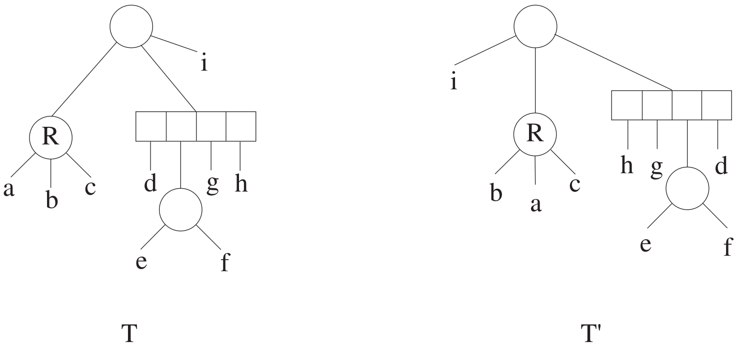

Figura 7: Árvores PQR equivalentes.

A **fronteira** de uma árvore PQR T, notada $Front(T)$, é a permutação das folhas da árvore obtida pela leitura destas folhas da esquerda para a direita. Ainda neste capítulo mostramos que a fronteira de uma árvore PQR para $(U,\mathcal{C})$ sem nós do tipo R é uma permutação válida de U para $\mathcal{C}$.

Exemplo 10 *Na Figura 7, a fronteira da árvore PQR T é abcdefghi e a da árvore T' é ibachgefd.*

Meidanis e Munuera introduzem duas novas coleções em seu trabalho, $\overline{\mathcal{C}}$ e $\mathcal{C}^{\perp}$, e uma caracterização dos nós internos da árvore com base nessas novas coleções: os conjuntos na interseção $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$ são domínios de subárvores de uma árvore PQR construída para uma instância $(U,\mathcal{C})$. Antes de apresentar as coleções $\overline{\mathcal{C}}$ e $\mathcal{C}^{\perp}$ vamos introduzir alguns conceitos e propriedades.

$\diamondsuit$ **Definição 11** *Dada uma árvore PQR T, o conjunto de todas as permutações de U iguais a fronteiras de árvores equivalentes a T, chamadas* **permutações compatíveis**, *é*

$$Compat(T) = \{\alpha : \alpha = Front(T'), T' \equiv T\}$$

$\diamondsuit$ **Definição 12** *O conjunto de todas as permutações de U válidas para uma coleção $\mathcal{C}$ é*

$$Valid(\mathcal{C}) = \{\alpha : \alpha \text{ é válida para } \mathcal{C}\}.$$

Lema 13 *Sejam $\mathcal{C},\mathcal{D} \subseteq \mathcal{P}(U)$. Se $\mathcal{C} \subseteq \mathcal{D}$, então*

$$Valid(\mathcal{D}) \subseteq Valid(\mathcal{C}).$$

Prova: Sejam $\mathcal{C}$ e $\mathcal{D}$ como no enunciado. Seja $\alpha \in Valid(\mathcal{D})$ uma permutação de U. Certamente os conjuntos em $\mathcal{C}$ são consecutivos em α, pois $\mathcal{C} \subseteq \mathcal{D}$. Logo $\alpha \in Valid(\mathcal{C})$ e $Valid(\mathcal{D}) \subseteq Valid(\mathcal{C})$. $\square$

♢ **Definição 14** *A coleção de todos os conjuntos consecutivos em uma permutação α é*

$$Consec(\alpha) = \{A : A \text{ é consecutivo em } \alpha\}.$$

Se π é um conjunto de permutações, definimos

$$Consec(\pi) = \bigcap_{\alpha \in \pi} Consec(\alpha).$$

Lema 15 *Seja $\mathcal{C}$ uma coleção tal que $Valid(\mathcal{C}) \neq \emptyset$ e A um conjunto tal que $A \notin Consec(Valid(\mathcal{C}))$. Então*

$$Valid(\mathcal{C} \cup \{A\}) \subset Valid(\mathcal{C}).$$

Prova: Pela Lema 13, $Valid(\mathcal{C} \cup \{A\}) \subseteq Valid(\mathcal{C})$. Como $A \notin Consec(Valid(\mathcal{C}))$, A não é consecutivo em alguma permutação $\alpha \in Valid(\mathcal{C})$. Logo $Valid(\mathcal{C} \cup \{A\}) \subset Valid(\mathcal{C})$. □

Lema 16 *Sejam $\pi, \pi' \subseteq Perm(U)$ dois conjuntos de permutações. Se $\pi \subseteq \pi'$ então*

$$Consec(\pi') \subseteq Consec(\pi).$$

Prova: Sejam π e π' como no enunciado. Seja $A \in Consec(\pi')$. Então as permutações em π são válidas para A, pois $\pi \subseteq \pi'$. Logo $A \in Consec(\pi)$ e $Consec(\pi') \subseteq Consec(\pi)$. □

♢ **Definição 17** *Um conjunto $A \subseteq U$ é* **trivial** *se $A = \emptyset$, $A = U$ ou $A = \{a\}$ para algum $a \in U$. Uma coleção é trivial se todos os seus conjuntos são triviais. A coleção $\mathcal{T}(U)$ é aquela formada por todos os subconjuntos triviais de U.*

3.1.1 A coleção $\overline{\mathcal{C}}$

A definição de $\overline{\mathcal{C}}$ vem da observação de que se os conjuntos na coleção $\mathcal{C}$ são todos consecutivos em alguma permutação α de U, outros conjuntos também devem ser consecutivos em α:

1. a **interseção**, $A \cap B$, de dois conjuntos em $\mathcal{C}$.

2. a **união não disjunta**, $A \cup B$ desde que $A \cap B \neq \emptyset$, de dois conjuntos em $\mathcal{C}$.

3. a **diferença não contida**, $A \setminus B$ desde que $B \not\subseteq A$, de dois conjuntos em $\mathcal{C}$.

Além destes conjuntos, os subconjuntos unitários $\{a\}$ para todo $a \in U$ e o próprio U são sempre consecutivos em qualquer permutação. Convencionamos que o conjunto vazio sempre é consecutivo em qualquer permutação, o que permite concluir que $\mathcal{T}(U)$ é sempre consecutiva em qualquer permutação.

Antes de demonstrar a validade dessas afirmações é necessário formalizar a notação a ser utilizada. Uma vez que uma permutação α é uma seqüência de elementos, vamos notar por $\alpha(i)$ o elemento de α em sua i-ésima posição e por $\alpha^{-1}(x)$ a posição ocupada pelo elemento $x \in U$ em α. Por $\alpha^{-1}(A)$ notaremos o conjunto de índices i tal que $\alpha(i) \in A$. Se A é consecutivo em α então notaremos o conjunto $\alpha^{-1}(A)$ na forma de um intervalo $[i..j]$ de índices de α, onde i e j são, respectivamente, a posição ocupada pelo primeiro e pelo último elemento de α que pertencem a A. Algumas destas notações já foram usadas na Seção 2.4.1.

$\diamondsuit$ **Lema 18** *Se os conjuntos A e B são consecutivos em α, então $A \cap B$ é consecutivo em α.*

Prova: Seja $\alpha^{-1}(A) = [i..j]$ e $\alpha^{-1}(B) = [k..l]$. Se $A \cap B = \emptyset$ este conjunto é consecutivo em α por convenção. Caso contrário a interseção desses conjuntos, dada pelo intervalo $[max(i,k)..min(j,l)]$, é consecutiva em α. $\square$

$\diamondsuit$ **Lema 19** *Se os conjuntos A e B são consecutivos em α e $A \cap B \neq \emptyset$, então $A \cup B$ é consecutivo em α.*

Prova: Seja $\alpha^{-1}(A) = [i..j]$ e $\alpha^{-1}(B) = [k..l]$. Suponhamos, sem perda de generalidade, $i \leq k$. Se $l \leq j$, então $B \subseteq A$ e $A \cup B = A$, que é consecutivo em α. Senão, dado que $A \cap B \neq \emptyset$, temos $\alpha^{-1}(A \cup B) = [i..l]$ e $A \cup B$ é consecutivo em α. $\square$

$\diamondsuit$ **Lema 20** *Se os conjuntos A e B são consecutivos em α e $B \not\subseteq A$, então $A \setminus B$ é consecutivo em α.*

Prova: Seja $\alpha^{-1}(A) = [i..j]$ e $\alpha^{-1}(B) = [k..l]$. Se $l < i$ ou $j < k$, os conjuntos são disjuntos e então $A \setminus B = A$. Se $i \leq k$ e $l \leq j$ então $B \subseteq A$. Consideremos agora os outros casos em que $A \cap B \neq \emptyset$:

- Se $k < i$ e $l \leq j \Longrightarrow \alpha^{-1}(A \setminus B) = [l+1..j]$.
- Se $k < i$ e $j < l \Longrightarrow \alpha^{-1}(A \setminus B) = \emptyset$.
- Se $i \leq k$ e $j < l \Longrightarrow \alpha^{-1}(A \setminus B) = [i..k-1]$.

Em todos estes casos $A \setminus B$ é consecutivo em α, o que conclui a prova. $\square$

Definimos agora a coleção $\overline{\mathcal{C}}$.

$\diamondsuit$ **Definição 21** *Uma coleção $\mathcal{C} \subseteq \mathcal{P}(U)$ é* **completa** *se esta coleção contiver todos os subconjuntos triviais de U e for fechada, i.e., contiver os conjuntos gerados, em relação às operações 1, 2 e 3 acima.*

$\diamondsuit$ **Definição 22** *Para uma coleção $\mathcal{C}$, seu* **fecho**, *$\overline{\mathcal{C}}$, é definido como a menor coleção completa que contém $\mathcal{C}$. Outra forma de defini-lo é como a interseção de todas as super-coleções completas de $\mathcal{C}$.*

Ainda outra forma de obtermos o fecho de uma coleção $\mathcal{C}$ é através da aplicação sucessiva das operações 1, 2 e 3 sobre $\mathcal{C} \cup \mathcal{T}(U)$ até que todos os conjuntos que possam ser formados pertençam ao fecho.

Exemplo 23 *Se tomarmos $U = abcde$ e a coleção $\mathcal{D} = \{ac, bde, ce\}$ então $\overline{\mathcal{D}} = \mathcal{T}(U) \cup \{ac, bde, ce, ace, bcde, bd\}$. Se $\mathcal{A} = \{ad\}$ então $\overline{\mathcal{A}} = \mathcal{A} \cup \mathcal{T}(U)$.*

Apresentamos agora, em um único teorema, um conjunto de resultados que são conseqüências diretas das definições. Na seqüência, alguns resultados relacionados à coleção $\overline{\mathcal{C}}$.

$\diamondsuit$ **Teorema 24** *Dadas duas coleções $\mathcal{C}$ e $\mathcal{D}$ sobre o mesmo conjunto U, temos:*

$$\begin{array}{l}
\mathcal{T}(U) \subseteq \overline{\mathcal{C}} \\
\mathcal{C} \subseteq \overline{\mathcal{C}} \\
\mathcal{C} \subseteq \mathcal{T}(U) \Longleftrightarrow \overline{\mathcal{C}} = \mathcal{T}(U) \\
\mathcal{C} \subseteq \mathcal{D} \Longrightarrow \overline{\mathcal{C}} \subseteq \overline{\mathcal{D}} \\
\overline{\overline{\mathcal{C}}} = \overline{\mathcal{C}}
\end{array}$$

Lema 25 *Se $\pi \subseteq Perm(U)$ então $Consec(\pi)$ é completa.*

Prova: Se π é vazio, $Consec(\pi) = \mathcal{P}(U)$ e é completa. Senão, sejam $A, B \in Consec(\pi)$. Pelos Lemas 18, 19 e 20, $A \cap B$, $A \cup B$ se $A \cap B \neq \emptyset$ e $A \setminus B$ se $B \not\subseteq A$ são consecutivos em π, logo pertencem a $Consec(\pi)$. Então $Consec(\pi)$ é completa. $\square$

$\diamondsuit$ **Teorema 26 (Meidanis e Munuera)** *Para toda coleção $\mathcal{C}$,*

$$Valid(\mathcal{C}) = Valid(\overline{\mathcal{C}}).$$

Prova: O fato que $\mathcal{C} \subseteq \overline{\mathcal{C}}$ garante que $Valid(\overline{\mathcal{C}}) \subseteq Valid(\mathcal{C})$ (Lema 13). Para provar que $Valid(\mathcal{C}) \subseteq Valid(\overline{\mathcal{C}})$, seja $\alpha \in Valid(\mathcal{C})$. Pelos Lemas 18, 19 e 20 podemos ver que os conjuntos em $\overline{\mathcal{C}}$ obtidos pelas operações descritas naqueles lemas aplicadas sobre $\mathcal{C}$ são todos consecutivos em α. Já que isto é válido também para todos os subconjuntos triviais, podemos concluir que $\alpha \in Valid(\overline{\mathcal{C}})$. □

O resultado anterior permite perceber que se temos uma árvore PQR T e a fronteira de T é válida para todos os conjuntos de $\mathcal{C}$, $\overline{\mathcal{C}}$ nos dá todos os possíveis domínios de subárvores de T. Por exemplo, consideremos a árvore T na Figura 8.

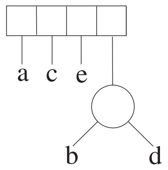

Figura 8: Árvore T.

Os conjuntos das coleções $\mathcal{D}$ e $\overline{\mathcal{D}}$ no Exemplo 23 são consecutivos na fronteira de T. Podemos verificar também que são consecutivos na fronteira de qualquer árvore equivalente a T. Mas podemos perceber que $\overline{\mathcal{D}}$ tem mais elementos do que T tem subárvores, e que alguns desses conjuntos em $\overline{\mathcal{D}}$ são partes do domínio de subárvores distintas, como $bcde$. Isso indica a necessidade de refinar esta coleção para que possamos usá-la para caracterizar subárvores de T, o que será feito pela coleção $\mathcal{D}^\perp$.

3.1.2 A coleção $\mathcal{C}^\perp$

Antes de definir a coleção $\mathcal{C}^\perp$ vamos fazer uma definição preliminar.

♢ **Definição 27** *Dados dois subconjuntos A e B de U dizemos que A e B são* **ortogonais**, *$A \perp B$, quando,*

- *$A \subseteq B$ ou*
- *$B \subseteq A$ ou então*
- *$A \cap B = \emptyset$.*

Dado um subconjunto $A \in U$ e uma coleção $\mathcal{C} \subseteq \mathcal{P}(U)$, notamos $A \perp \mathcal{C}$ quando $A \perp B$ para todo $B \in \mathcal{C}$. Tal conjunto A é dito ortogonal a $\mathcal{C}$. Da mesma forma, notamos $\mathcal{C} \perp \mathcal{D}$ quando queremos indicar que duas coleções são ortogonais, significando que $A \perp B$ para todo $A \in \mathcal{C}$ e para todo $B \in \mathcal{D}$.

$\diamondsuit$ **Definição 28** *A coleção $\mathcal{C}^\perp$ contém todos os conjuntos em $\mathcal{P}(U)$ ortogonais a $\mathcal{C}$,*

$$\mathcal{C}^\perp = \{A : A \in \mathcal{P}(U), A \perp \mathcal{C}\}$$

Exemplo 29 *Vamos supor $U = abcde$ e $\mathcal{D} = \{ac, bde, ce\}$. Então $\mathcal{D}^\perp = \mathcal{T}(U) \cup \{bd\}$.*

Algumas conseqüências imediatas das definições são reunidas no teorema abaixo.

$\diamondsuit$ **Teorema 30** *Dadas duas coleções $\mathcal{C}$ e $\mathcal{D}$ de subconjuntos do mesmo conjunto U, temos:*

$$\mathcal{T}(U) \subseteq \mathcal{C}^\perp$$
$$\mathcal{C} \subseteq \mathcal{T}(U) \Longleftrightarrow \mathcal{C}^\perp = \mathcal{P}(U)$$
$$\mathcal{C} \subseteq \mathcal{D} \Longrightarrow \mathcal{D}^\perp \subseteq \mathcal{C}^\perp \tag{1}$$
$$\mathcal{C} \perp \mathcal{D} \Longleftrightarrow \mathcal{C} \subseteq \mathcal{D}^\perp \Longleftrightarrow \mathcal{D} \subseteq \mathcal{C}^\perp \tag{2}$$

Observando a interseção de $\overline{\mathcal{D}}$ e $\mathcal{D}^\perp$, nos Exemplos 23 e 29, que é $\mathcal{T}(U) \cup \{db\}$, vemos que esse conjunto tem exatamente as fronteiras de subárvores de T, na Figura 8. A coleção $\mathcal{D}^\perp$ refinou os conjuntos em $\overline{\mathcal{D}}$ da maneira que queríamos, eliminando os conjuntos que são formados por partes de fronteiras de árvores distintas.

3.1.3 Relações entre $\overline{\mathcal{C}}$ e $\mathcal{C}^\perp$

Os resultados que apresentamos nesta seção ressaltam algumas propriedades importantes do relacionamento entre $\overline{\mathcal{C}}$ e $\mathcal{C}^\perp$.

$\diamondsuit$ **Lema 31** *Se $\mathcal{B} \perp \mathcal{C}$ então $\overline{\mathcal{B}} \perp \mathcal{C}$.*

Prova: Sejam $\mathcal{B}$ e $\mathcal{C}$ duas coleções tais que $\mathcal{B} \perp \mathcal{C}$. Suponhamos $A, B \in \mathcal{B}$ e $H \in \mathcal{C}$. Então $A \perp H$ e $B \perp H$. A Tabela 1 mostra que os conjuntos $A \cap B$, $A \cup B$ quando $A \cap B \neq \emptyset$ e $A \setminus B$ quando $B \not\subseteq A$ são todos ortogonais a H, qualquer que seja o caso pelo qual A e B são ortogonais a H. Já que os triviais são sempre ortogonais a qualquer conjunto, podemos concluir que $\overline{\mathcal{B}} \perp \mathcal{C}$. $\square$

$\diamondsuit$ **Teorema 32 (Meidanis e Munuera)** *Para qualquer coleção $\mathcal{C}$,*

$$\mathcal{C}^\perp = \overline{\mathcal{C}^\perp} = (\overline{\mathcal{C}})^\perp$$

$\cap$	$B \cap H = \emptyset$	$B \subseteq H$	$H \subseteq B$
$A \cap H = \emptyset$	$(A \cap B) \cap H = \emptyset$	$(A \cap B) \cap H = \emptyset$	$(A \cap B) \cap H = \emptyset$
$A \subseteq H$	$(A \cap B) \cap H = \emptyset$	$(A \cap B) \subseteq H$	$A \cap B = A$
$H \subseteq A$	$(A \cap B) \cap H = \emptyset$	$A \cap B = B$	$(A \cap B) \supseteq H$

$\cup$	$B \cap H = \emptyset$	$B \subseteq H$	$H \subseteq B$
$A \cap H = \emptyset$	$(A \cup B) \cap H = \emptyset$	$A \cap B = \emptyset$	$(A \cup B) \supseteq H$
$A \subseteq H$	$A \cap B = \emptyset$	$(A \cup B) \subseteq H$	$(A \cup B) \supseteq H$
$H \subseteq A$	$(A \cup B) \supseteq H$	$(A \cup B) \supseteq H$	$(A \cup B) \supseteq H$

$\setminus$	$B \cap H = \emptyset$	$B \subseteq H$	$H \subseteq B$
$A \cap H = \emptyset$	$(A \setminus B) \cap H = \emptyset$	$A \setminus B = A$	$(A \setminus B) \cap H = \emptyset$
$A \subseteq H$	$A \setminus B = A$	$A \setminus B \subseteq H$	$A \setminus B = \emptyset$
$H \subseteq A$	$A \setminus B \supseteq H$	$B \subseteq A$	$(A \setminus B) \cap H = \emptyset$

Tabela 1: Resultados dos operadores interseção, união não disjunta e diferença não contida.

Prova: Aplicando o Lema 31 e a Equação 2 sobre a relação $\mathcal{C} \perp \mathcal{C}^\perp$, podemos derivar:

$$\mathcal{C} \perp \overline{\mathcal{C}^\perp} \Longrightarrow \overline{\mathcal{C}^\perp} \subseteq \mathcal{C}^\perp$$
$$\overline{\mathcal{C}} \perp \mathcal{C}^\perp \Longrightarrow \mathcal{C}^\perp \subseteq (\overline{\mathcal{C}})^\perp$$

Como uma coleção está sempre contida em seu fecho, $\mathcal{C}^\perp \subseteq \overline{\mathcal{C}^\perp}$, o que implica na primeira igualdade enunciada. Usando a Equação 1, vemos que $(\overline{\mathcal{C}})^\perp \subseteq \mathcal{C}^\perp$. Então $\mathcal{C}^\perp = (\overline{\mathcal{C}})^\perp$. □

Lema 33 *Sejam $\mathcal{A}, \mathcal{B} \subseteq \mathcal{P}(U)$. Se $\mathcal{A} \perp \mathcal{B}$, então $\overline{\mathcal{A}} \cup \overline{\mathcal{B}}$ é completa.*

Prova: Se existir um conjunto em $\overline{\mathcal{A}} \cup \overline{\mathcal{B}}$ que pode ser gerado pelas operações de fecho, ele deve ser gerado envolvendo um conjunto em $\mathcal{A}$ e outro em $\mathcal{B}$, pois $\overline{\mathcal{A}}$ e $\overline{\mathcal{B}}$ já são completas. A Tabela 2 reúne todos os casos considerando que $A \perp B$. Em todas as operações, o conjunto gerado é um dos originais ou trivial. Então $\overline{\mathcal{A}} \cup \overline{\mathcal{B}}$ é completa. □

3.2 Relações entre a P1C e as Árvores PQR

Nesta seção nos dedicamos a mostrar que para todo par $(U, \mathcal{C})$ é possível construir uma árvore PQR e que se esta árvore não tem nós do tipo R, seu conjunto de permutações

	$A \cap B$	$A \cup B$	$A \setminus B$	$B \setminus A$
$A \subseteq B$	A	B	$\emptyset$	—
$B \subseteq A$	B	A	—	$\emptyset$
$A \cap B = \emptyset$	$\emptyset$	—	A	B

Tabela 2: Todos os casos possíveis para mostrar que $\overline{\mathcal{A}} \cup \overline{\mathcal{B}}$ é completa se $\mathcal{A} \perp \mathcal{B}$.

compatíveis é igual ao conjunto de permutações válidas de U para $\mathcal{C}$. Mostramos também que toda árvore PQR representa um par $(U, \mathcal{C})$.

Podemos construir uma árvore PQR indutivamente, das folhas para a raiz, tirando proveito das características das coleções $\overline{\mathcal{C}}$ e $\mathcal{C}^{\perp}$, que servem para caracterizar subárvores. Para tanto, construímos subinstâncias de $(U, \mathcal{C})$ aplicando as operações definidas abaixo:

$\diamondsuit$ **Definição 34** *Seja H um subconjunto não vazio de U e A um subconjunto de U com $H \subseteq A$ ou $H \cap A = \emptyset$. Definimos*

$$A/H = \begin{cases} (A \setminus H) \cup \{H\} & se\ H \subseteq A \\ A & se\ H \cap A = \emptyset \end{cases}$$

e sua operação inversa para $H \in \mathcal{P}(U)$ e $B \in (U \cup \{H\})$, com $B \cap H = \emptyset$,

$$B \bowtie H = \begin{cases} (B \setminus \{H\}) \cup H & se\ H \in B \\ B & se\ H \notin B \end{cases}$$

O conjunto A/H é o conjunto formado substituindo-se no conjunto A os elementos do conjunto H por um único elemento, chamado H por simplicidade.

Tomando um conjunto não vazio $H \in \overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$, podemos obter duas subinstâncias de uma instância $(U, \mathcal{C})$ para a qual queiramos construir uma árvore PQR:

1. A primeira subinstância é dada pelo par $(H, \mathcal{C} \wedge H)$, onde $\mathcal{C} \wedge H = \{A : A \in \mathcal{C}, A \subset H\}$. Esta subinstância é notada $(U, \mathcal{C}) \wedge H$.

2. A segunda subinstância é construída tirando vantagem do fato de H ser ortogonal a $\mathcal{C}$: $(U/H, \mathcal{C}/H)$, onde $\mathcal{C}/H = \{A/H : A \in \mathcal{C}$ e $(H \subset A$ ou $H \cap A = \emptyset)\}$. Esta subinstância é notada $(U, \mathcal{C})/H$.

Exemplo 35 *Sejam $U = abcdef$, $\mathcal{C} = \{acef, bcdef, abd, ef, ce\}$ e $H = cef$. Então,*

$$(U, \mathcal{C}) \wedge H = (cef, \{ef, ce\})$$
$$(U, \mathcal{C})/H = (abdH, \{aH, bdH, abd\})$$

Algumas instâncias não devem ser decompostas, o que acontece quando não temos um conjunto H adequado. Este é o caso em que $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp} = \mathcal{T}(U)$, pois a divisão irá resultar em subinstâncias equivalentes. Tais instâncias, bem como as coleções que as formam, são chamadas **primas**.

Exemplo 36 *Seja $U = abcdef$ e $\mathcal{C} = \{ab, bc, cd, de, ef\}$. Vemos que $\mathcal{C}^{\perp} = \mathcal{T}(U)$, logo, $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp} = \mathcal{T}(U)$. Se fizermos $H = a$, teremos*

$$(U, \mathcal{C}) \wedge H = (a, \emptyset)$$
$$(U, \mathcal{C})/H = (Hbcdef, \{Hb, bc, cd, de, ef\})$$

Fazendo $H = U$, teremos

$$(U, \mathcal{C}) \wedge H = (U, \mathcal{C})$$
$$(U, \mathcal{C})/H = (\{H\}, \emptyset)$$

Neste ponto é oportuno estudar as permutações válidas para as subinstâncias. Para tanto notaremos por $\beta[H/\gamma]$, quando β é uma permutação sobre U/H e γ uma permutação sobre H, a permutação obtida pela substituição de H por γ em β. Se H puder ser omitido notaremos $\beta \star \gamma$ ao invés de $\beta[H/\gamma]$. Assim, se X e Y são conjuntos de permutações, respectivamente, de U e H, teremos: $X \star Y = \{\beta \star \gamma : \beta \in X, \gamma \in Y\}$.

Exemplo 37 *Se $\beta = aHeb$ e $\gamma = dcf$, então $\beta[H/\gamma] = \beta \star \gamma = adcfeb$.*

O resultado que valida a divisão em subinstâncias em relação às permutações válidas é apresentado a seguir.

$\diamondsuit$ **Teorema 38 (Meidanis e Munuera)** *Dada uma coleção $\mathcal{C} \subseteq \mathcal{P}(U)$ e um conjunto $H \in (\overline{\mathcal{C}} \cap \mathcal{C}^{\perp})$, $H \neq \emptyset$, temos,*

$$\mathit{Valid}(U, \mathcal{C}) = \mathit{Valid}(U/H, \mathcal{C}/H) \star \mathit{Valid}(H, \mathcal{C} \wedge H).$$

Prova: Dada uma permutação $\alpha \in \mathit{Valid}(U, \mathcal{C})$, os elementos de H são consecutivos em α, já que $H \in \overline{\mathcal{C}}$. Então α pode ser escrita como $\alpha = \beta[H/\gamma]$, onde β é uma permutação de U/H e γ é uma permutação de H.

Vamos mostrar primeiro que $\gamma \in \mathit{Valid}(H, \mathcal{C} \wedge H)$. Uma vez que $H \in \mathcal{C}^{\perp}$, $A \cap H$ ou é vazio ou igual a A ou a H, para todo $A \in \mathcal{C}$. Consideremos então um conjunto $A \in \mathcal{C}$ qualquer. Se $A \cap H = \emptyset$ ou $A \cap H = H$, γ é certamente válida para $A \cap H$. Se $A \cap H = A$, γ é válida para $A \cap H$ porque α é válida para A.

Com a finalidade de mostrar que $\beta \in \mathit{Valid}(U/H, \mathcal{C}/H)$, consideremos A/H um conjunto de $\mathcal{C}/H$. Como α é válida para A, se $H \subseteq A$ com certeza β será válida para

A/H, já que o conjunto H foi substituído pelo elemento H tanto em A quanto em α. Se $H \cap A = \emptyset$ então os elementos de A em α não são alterados e se mantêm consecutivos em β. Em ambos os casos β é válida para A/H, o que mostra que $\beta \in \mathit{Valid}(U/H, \mathcal{C}/H)$. Analogamente, podemos ver que se $\beta \in \mathit{Valid}(U/H, \mathcal{C}/H)$ e $\gamma \in \mathit{Valid}(H, \mathcal{C} \wedge H)$, então $\beta[H/\gamma] \in \mathit{Valid}(U, \mathcal{C})$. □

Definimos agora o grafo de binários de $\overline{\mathcal{C}}$ e apresentamos alguns resultados envolvendo coleções primas apresentados por Munuera [Mun96].

♢ **Definição 39** *O grafo de* **binários** *de* $\overline{\mathcal{C}}$*,* $B(U, \mathcal{C})$*, tem um vértice para cada elemento de* U*. Há aresta entre dois vértices* a, b *de* $B(U, \mathcal{C})$ *se e somente se o conjunto* $\{a, b\}$*, formado pelos elementos representados pelos vértices* a *e* b *pertencer a* $\overline{\mathcal{C}}$*. Quando* U *puder ser omitido, notaremos* $B(U, \mathcal{C})$ *por* $B(\mathcal{C})$*.*

♢ **Lema 40 (Munuera)** *Para qualquer coleção* $\mathcal{C}$*, cada componente conexa de* $B(\mathcal{C})$ *é ou caminho ou subgrafo completo.*

Prova: A afirmação é claramente verdadeira para componentes de até três vértices. Vamos tomar então uma componente X que não é caminho tal que $|X| \geq 4$. Se X não é um caminho, então X possui um ciclo ou vértice de grau pelo menos 3.

Se X possui um ciclo envolvendo, por exemplo, a, b e c como elementos consecutivos, então, por união não disjunta teremos em $\overline{\mathcal{C}}$ o conjunto abc e um outro conjunto A que contenha a e c mas não b, conseguido por uniões sucessivas de conjuntos ao redor do ciclo (Figura 9). A interseção $abc \cap A$ é ac e então há um vértice de grau pelo menos 3 em qualquer caso.

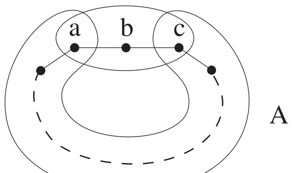

Figura 9: Conjuntos abc e A, ao redor do ciclo.

Afirmamos que a vizinhança de um vértice de grau 3 ou mais é um clique. De fato, se existem as arestas ax, bx e cx então $ab = (ax \cup bx) \setminus cx$ pertence a $\overline{\mathcal{C}}$. Por processo análogo, podemos mostrar que ac e bc também estão em $\overline{\mathcal{C}}$. Fazendo o mesmo com os vértices restantes em X, i.e., processando sucessivamente cada um dos vértices adjacentes ao clique já obtido, aumentando-o, podemos perceber que X é um subgrafo completo, o que confirma nossa afirmação e conclui nossa prova. □

Para avançar em nossos resultados, precisamos do lema a seguir e da seguinte definição: um conjunto $A \subseteq U$ é **conexo** (em relação a $\mathcal{C}$) quando A induz um subgrafo conexo de $B(\mathcal{C})$.

$\diamondsuit$ **Lema 41 (Munuera)** *Se $\mathcal{C}$ é prima e $A \in \overline{\mathcal{C}}$ não é trivial, então A é conexo.*

Prova: Vamos usar indução em $|A|$. O resultado é imediato se $|A| = 2$, o que constitui nosso caso base. Se $|A| \geq 3$, A não pode ser ortogonal a $\mathcal{C}$ porque $\mathcal{C}$ é prima. Então existe um conjunto $B \in \mathcal{C}$ tal que $A \not\perp B$. Neste caso tanto $A \cap B$ quanto $A \setminus B$ estão em $\overline{\mathcal{C}}$ (Lemas 18 e 20), são estritamente menores que A e por hipótese de indução, são ambos conexos. Ainda, já que $|A| \geq 3$, pelo menos um deles não é trivial. Mas A não pode ser formado pela união não-disjunta de $A \cap B$ e $A \setminus B$. Então vamos construir um conjunto X que se sobrepõe estritamente tanto a $A \cap B$ quanto $A \setminus B$ e está contido em A. Usando então o conjunto X como uma ponte, vamos mostrar que A pode ser formado por união não disjunta dos conjuntos conexos $A \cap B$, X e $A \setminus B$, e portanto é conexo.

Façamos então $D = A \cap B$ se $A \cap B$ não é trivial ou $D = A \setminus B$, caso contrário. Como em qualquer caso D não é trivial e $\mathcal{C}$ é prima, D não é ortogonal a $\mathcal{C}$ e existe um conjunto $E \in \mathcal{C}$ tal que $D \not\perp E$.

Se $E \cap (A \setminus D) \neq \emptyset$ então $(A \cap E) \cap D \neq \emptyset$ e $(A \cap E) \cap (A \setminus D) \neq \emptyset$. Façamos $X = A \cap E$, como na Figura 10(a). Uma vez que $X \in \overline{\mathcal{C}}$ e é conexo por indução, podemos ver que existe um caminho em X entre os elementos de D e elementos de $A \setminus D$, ambos em $\overline{\mathcal{C}}$ e conexos por indução, o que permite concluir que A é conexo.

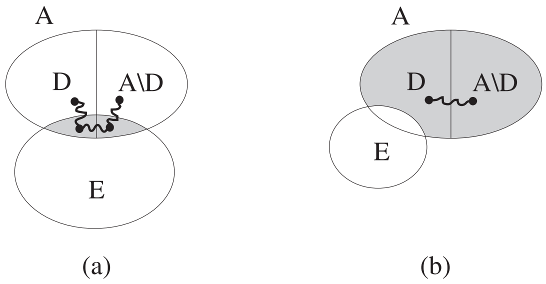

Figura 10: O conjunto X, sombreado, como ponte entre D e $A \setminus D$.

Se $E \cap (A \setminus D) = \emptyset$, E deve possuir algum elemento que não pertence a A. Mas nesse caso $(A \setminus E) \cap D \neq \emptyset$ e $(A \setminus E) \cap (A \setminus D) \neq \emptyset$. Usando a mesma argumentação anterior para $X = A \setminus E$, podemos concluir que A também é conexo neste caso (Figura 10(b)). $\square$

Como conseqüência deste resultado, apresentamos o seguinte lema:

$\diamondsuit$ **Lema 42 (Munuera)** *Se $\mathcal{C}$ é prima e $\overline{\mathcal{C}}$ não é trivial, então $B(\mathcal{C})$ é um grafo conexo.*

Prova: Seja A um conjunto maximal dentre todos os não triviais de $\mathcal{C}$. Pelo Lema 41, A é conexo. Como $\mathcal{C}$ é prima, $A \notin \mathcal{C}^{\perp}$ e então existe um conjunto $B \in \mathcal{C}$ tal que $B \not\perp A$. Certamente B não é trivial e então, novamente pelo Lema 41, B é conexo. Como A é maximal, $A \cup B$ deve ser igual a U, pois caso contrário teríamos em $\overline{\mathcal{C}}$ um conjunto maior que A, contrariando a maximalidade de A. Então, como $A \cap B \neq \emptyset$, A e B estão conectados, e já que ambos são conexos e sua união é igual a U, $B(\mathcal{C})$ é conexo. □

Quando temos uma coleção sobre um conjunto U e $|U| < 3$, todo conjunto é trivial e toda permutação é válida em relação a todo conjunto. O resultado que segue caracteriza instâncias primas maiores.

♢ **Teorema 43 (Meidanis e Munuera)** *Seja $\mathcal{C} \in \mathcal{P}(U)$ uma coleção prima com $|U| \geq 3$. Então, ou*

1. *$\overline{\mathcal{C}} = \mathcal{T}(U)$, ou*
2. *$\overline{\mathcal{C}} = Consec(\alpha)$ para alguma permutação α de U, ou então*
3. *$\overline{\mathcal{C}} = \mathcal{P}(U)$.*

Prova: Se $\mathcal{C} \subseteq \mathcal{T}(U)$ então $\overline{\mathcal{C}} = \mathcal{T}(U)$, a primeira afirmação.

Se $\mathcal{C}$ não é trivial, então, pelo Lema 42 teremos $B(\mathcal{C})$ conexo. Além disso, pelo Lema 40, $B(\mathcal{C})$ é ou caminho ou completo. Se $B(\mathcal{C})$ é caminho, este caminho define duas permutações dos elementos de U, uma delas sendo o reverso da outra (percorrendo-se o caminho de um extremo a outro). Seja α uma dessas permutações. Qualquer conjunto consecutivo em α pode ser construído a partir dos binários por união não disjunta e então $Consec(\alpha) \subseteq \overline{\mathcal{C}}$. Se um conjunto não consecutivo em α estivesse em $\overline{\mathcal{C}}$ então seríamos capazes de construir um conjunto binário não consecutivo em α usando interseção e $B(\mathcal{C})$ não seria um caminho. A segunda afirmação é, então, válida. Finalmente, se $B(\mathcal{C})$ é completo, todos os binários pertencem a $\overline{\mathcal{C}}$ e então todos os subconjuntos de U pertencem a $\overline{\mathcal{C}}$, o que valida a terceira afirmação e conclui a prova. □

Vamos agora definir a coleção $Compl(T)$ para uma árvore PQR T e a, partir de algumas de suas propriedades, obter os resultados a que nos propusemos no início desta seção.

♢ **Definição 44** *O conjunto de todas as folhas que são descendentes de um nó v em uma árvore PQR T é notado $Desc_T(v)$:*

$$Desc_T(v) = \{a \in U \ : \ v \text{ é ancestral de } a \text{ em } T\}.$$

Esta notação é estendida a conjuntos S de nós:

$$Desc_T(S) = \bigcup_{v \in S} Desc_T(v).$$

Devemos observar que $Dom(T) = Desc_T(v)$ se v é a raiz de T.

◇ **Definição 45** *A coleção $Compl(T)$ é assim construída:*

1. *$\mathcal{T}(U)$ está contida em $Compl(T)$.*
2. *Se S é o conjunto de todos os filhos de um nó P de T, então $Desc_T(S) \in Compl(T)$.*
3. *Se S é um conjunto de filhos consecutivos de um nó Q de T, então $Desc_T(S) \in Compl(T)$.*
4. *Se S é um conjunto qualquer de filhos de um nó R de T, então $Desc_T(S) \in Compl(T)$.*
5. *Nenhum outro conjunto está em $Compl(T)$.*

Exemplo 46 *A coleção $Compl(T)$ para a árvore da Figura 6 (b) é composta dos conjuntos:*

$\mathcal{T}(U)$
$\{ghi\}, \{hi\}$ *(filhos dos nós P)*
$\{abcd, de, ef, fghi, abcde, def, efghi, abcdef, defghi, abc, ghi\}$ *(filhos do nó Q)*
$\{abc, ab, bc, ac\}$ *(filhos do nó R)*

A coleção $Compl(T)$ construída a partir de uma árvore T é completa, o que será demonstrado a seguir. Esse fato é importante para estabelecer a relação entre árvores PQR e coleções de conjuntos.

Lema 47 *Para qualquer árvore PQR T, a coleção $Compl(T)$ é completa.*

Prova: Os subconjuntos triviais estão em $Compl(T)$ por definição. É necessário mostrar que $Compl(T)$ é fechada por interseção, união não disjunta e diferença não-contida. Para tanto é suficiente considerar dois conjuntos, $Desc_T(S)$ e $Desc_T(S')$, sendo S e S' conjuntos de filhos do mesmo nó interno v, pois, caso contrário os conjuntos seriam ortogonais e as operações resultariam em um dos conjuntos originais ou em um trivial. Consideremos então o tipo do nó v. Se v é um nó P, então $S = S'$ e os resultados das operações são triviais. Se v é um nó Q, qualquer das operações aplicadas sobre S e S' vai resultar em algum conjunto de filhos consecutivos de v ou em algum conjunto trivial. Finalmente, se v é um nó R, como qualquer subconjunto de seus filhos é válido, aplicando as operações de fecho sempre obteremos um conjunto em $Compl(T)$. Concluímos então que $Compl(T)$ é completa. □

O resultado abaixo mostra que é sempre possível construir uma árvore PQR para uma coleção $\mathcal{C}$. Esse fato é usado pelo algoritmo recursivo para a construção das árvores PQR apresentado no Capítulo 4.

$\diamondsuit$ **Teorema 48** *Para toda coleção $\mathcal{C}$ sobre um conjunto universo U, existe uma árvore PQR T tal que*

$$Compl(T) = \overline{\mathcal{C}}.$$

Prova: Vamos usar indução no número de elementos de U. Se $|U| = 2$, uma árvore que consista apenas de uma raiz P e dois filhos satisfaz os requisitos para qualquer coleção $\mathcal{C}$ e nos serve de base. Investigando os casos em que $|U| \geq 3$, consideremos primeiro a situação em que $\mathcal{C}$ é prima. O Teorema 43 descreve $\overline{\mathcal{C}}$, e em cada um dos três casos apresentados, uma árvore T pode ser construída tal que $Compl(T) = \overline{\mathcal{C}}$, como vamos ver a seguir.

- No caso 1 temos um fecho trivial. Então, T pode ser um nó P com todos os elementos de U como filhos e teremos $Compl(T) = \mathcal{T}(U)$.

- No caso 2 apenas duas permutações são válidas, α e sua reversa, porque $\overline{\mathcal{C}} = Consec(\alpha)$. Façamos T uma árvore que tenha um nó Q com todos os elementos de U como filhos, na ordem dada por α. É claro que $Compl(T) = Consec(\alpha)$.

- No caso 3, todos os conjuntos estão no fecho. Então tomemos T como um nó R com todos os elementos de U como filhos. Por definição, $Compl(T) = \mathcal{P}(U)$.

Investigando as instâncias não primas, tomemos um conjunto não trivial $H \in \overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$. Construindo subinstâncias $\mathcal{C}/H$ e $\mathcal{C} \wedge H$, pela hipótese de indução existirão árvores PQR para $\mathcal{C}/H$ e $\mathcal{C} \wedge H$, respectivamente T_1 e T_2, tais que,

$$\begin{aligned} Compl(T_1) &= \overline{\mathcal{C}/H} \\ Compl(T_2) &= \overline{\mathcal{C} \wedge H}. \end{aligned}$$

Substituindo a folha H em T_1 por T_2, obtemos uma árvore PQR T. Para mostrar que $Compl(T) = \overline{\mathcal{C}}$, aplicamos a hipótese de indução e obtemos

$$Compl(T) = (Compl(T_1) \bowtie H) \cup (Compl(T_2)) = ((\overline{\mathcal{C}/H} \bowtie H) \cup (\overline{\mathcal{C} \wedge H}).$$

Aplicando a decomposição sobre $\overline{\mathcal{C}}$,

$$\overline{\mathcal{C}} = (\overline{\mathcal{C}}/H \bowtie H) \cup (\overline{\mathcal{C}} \wedge H).$$

Como $H \in \mathcal{C}^{\perp}$ e conseqüentemente $H \in (\overline{\mathcal{C}})^{\perp}$ (Teorema 32), obtemos os termos

$$\begin{aligned} \overline{\mathcal{C}/H} &= \overline{\mathcal{C}}/H \\ \overline{\mathcal{C} \wedge H} &= \overline{\mathcal{C}} \wedge H. \end{aligned}$$

Concluímos então que $Compl(T) = (\overline{\mathcal{C}}/H \bowtie H) \cup (\overline{\mathcal{C}} \wedge H) = \overline{\mathcal{C}}$. $\square$

Apresentamos agora uma seqüência de resultados que vão ser usados para relacionar as árvores PQR e a P1C.

Teorema 49 *Se uma árvore PQR T não possui nós R, então*

$$Compl(T) = Consec(Compat(T)).$$

Prova: A partir da definição podemos ver que todo conjunto em $Compl(T)$ é consecutivo na fronteira de T. É imediato também o fato de que duas árvores equivalentes fornecem a mesma coleção completa. Algebricamente,

$$T \equiv T' \Longrightarrow Compl(T) = Compl(T').$$

Podemos concluir que $Compl(T) \subseteq Consec(Compat(T))$, com base nas duas últimas observações.

Para mostrar que $Consec(Compat(T)) \subseteq Compl(T)$, seja A um conjunto qualquer em $Consec(Compat(T))$. Marquemos todas as folhas em T que pertencem a A e executemos o seguinte procedimento:

Enquanto existir um nó v com todos os filhos marcados **faça**
Desmarque os filhos de v e marque v.

Vamos denominar M o conjunto de nós marcados após o término do procedimento. Podemos, então, fazer

$$A = \bigcup_{v \in M} Desc_T(v).$$

Se M é um conjunto unitário, A pertence a $Compl(T)$ por definição. Se M tem pelo menos dois elementos, vamos mostrar que eles devem ser irmãos. Para começar, se houvesse entre eles relação "pai-filho", o filho não poderia ter permanecido marcado. A mesma observação vale para relações entre ancestrais: "avo-neto" etc. Seja então v o ancestral comum de menor altura de dois nós $x, y \in M$. Se existir um nó z no caminho de x a v, então z não está marcado e há uma folha a que descende de z mas não pertence a A (Figura 11). Independentemente do tipo do nó z alguma transformação de equivalência vai deixar a entre os filhos de x e y na fronteira, o que contradiz o fato de A ser consecutivo na fronteira de toda árvore equivalente a T. Então não há nós intermediários entre x e v e argumentos semelhantes valem para outros nós de M. Como conseqüência, M é um conjunto de irmãos. Seja então v o nó pai. Se v é um nó P, M deve conter todos os filhos de v, ou caso contrário poderíamos encontrar uma permutação dos filhos de M que resultasse em uma árvore equivalente em cuja fronteira A não fosse consecutivo. Se v é um nó Q, M deve conter filhos consecutivos, pela mesma razão. Nos dois casos, A se enquadra na descrição de um conjunto em $Compl(T)$ e como não há nós R na árvore, podemos concluir que $Consec(Compat(T)) \subseteq Compl(T)$, o que completa a prova. $\square$

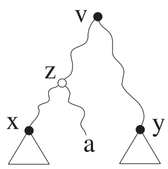

Figura 11: Filhos do nó v de T.

Teorema 50 *Se uma árvore PQR T não tem nós R, então*

$$Valid(Compl(T)) = Compat(T).$$

Prova: Vamos provar por indução no número de nós internos de T. Tomemos como base o caso em que T possua apenas um nó interno. Há apenas duas possibilidades para o tipo deste nó raiz: P ou Q. Se a raiz é um nó P então $Compl(T) = \mathcal{T}(U)$ e $Valid(Compl(T)) = Perm(U)$. Como $Compat(T) = Perm(U)$, vale a igualdade. Se a raiz é um nó Q, então $Compat(T) = \{\alpha, \overline{\alpha}\}$, para alguma permutação α ($\overline{\alpha}$ é a reversão de α). Do Teorema 49 vem que

$$Valid(Compl(T)) = Valid(Consec(\{\alpha, \overline{\alpha}\}) = \{\alpha, \overline{\alpha}\} = Compat(T).$$

Continuando com a indução, vamos considerar uma árvore T com mais de um nó interno e escolher um nó v de T diferente da raiz. O conjunto $H = Desc_T(v)$ satisfaz a hipótese do Teorema 38 e então temos:

$$Valid(Compl(T)) = Valid(Compl(T)/H) \star Valid(Compl(T) \wedge H).$$

Podemos dizer que $Compl(T)/H = Compl(T_1)$, onde T_1 é a árvore obtida de T pela substituição da subárvore enraizada em v por um único nó rotulado H. Da mesma forma, $Compl(T) \wedge H = Compl(T_2)$ onde T_2 é a subárvore enraizada em v. Por hipótese de indução, temos:

$$\begin{aligned} &Valid(Compl(T_1)) = Compat(T_1) \\ &Valid(Compl(T_2)) = Compat(T_2). \end{aligned}$$

Note que é possível usar a hipótese de indução, pois H não é trivial, o que faz com que tanto T_1 como T_2 tenham menos nós internos que T.

Também sabemos verdadeiro que

$$Compat(T) = Compat(T_1) \star Compat(T_2),$$

uma vez que T resulta da substituição da folha H de T_1 por T_2. Das três últimas igualdades, concluímos a prova. □

Teorema 51 *Dada uma árvore PQR T, a coleção $Compl(T)$ tem a P1C se e somente se T não tem nós R.*

Prova: Pelo Teorema 50, se T não tem nós R, então $Compl(T)$ possui permutações válidas: qualquer uma em $Compat(T)$. Por outro lado, tomemos qualquer permutação α das folhas de T e um nó v do tipo R em T. Consideremos ainda três filhos distintos, x, y e z, de v e rotulemos como a qualquer folha descendente de x, como b qualquer folha descendente de y e como c qualquer folha descendente de z (Figura 12).

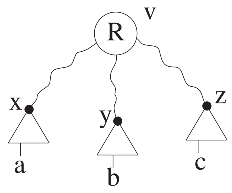

Figura 12: Filhos do nó v de T.

Indiferentemente à ordem em que a, b e c aparecem em α, um dos três conjuntos de $Compl(T)$ abaixo não vai ser consecutivo e então nenhuma permutação será válida:

$$Desc_T(x) \cup Desc_T(y)$$
$$Desc_T(y) \cup Desc_T(z)$$
$$Desc_T(z) \cup Desc_T(x)$$

□

O resultado abaixo estabelece a relação entre as permutações válidas para uma coleção $\mathcal{C}$ e as fronteiras de uma árvore PQR.

Teorema 52 *Se $\mathcal{C}$ tem a P1C e a árvore T é tal que $Compl(T) = \overline{\mathcal{C}}$ então,*

$$Valid(\mathcal{C}) = Compat(T).$$

Prova: Pelos Teoremas 26 e 50 temos

$$Valid(\mathcal{C}) = Valid(\overline{\mathcal{C}}) = Valid(Compl(T)) = Compat(T).$$

Podemos usar o Teorema 50 porque o Teorema 51 garante que T não tem nós R. □

Os Teoremas 48, 50 e 51 mostram que para qualquer coleção $\mathcal{C}$ é possível construir uma árvore PQR T que não terá nós R se e somente se $\mathcal{C}$ possui a P1C. Além disso, se $\mathcal{C}$ possui a P1C, as permutações válidas para $\mathcal{C}$ são exatamente as permutações compatíveis de T. Uma tal árvore será chamada de **árvore PQR para $\mathcal{C}$**.

3.3 Mais sobre Árvores, Coleções e Permutações

Apresentamos, deste ponto em diante, resultados que relacionam árvores, coleções e permutações. Alguns deles serão usados para resolver problemas apresentados no Capítulo 5 e outros complementam resultados apresentados na seção anterior.

Lema 53 *Se* $Valid(\mathcal{C}) \neq \emptyset$ *então* $\overline{\mathcal{C}} = Consec(Valid(\mathcal{C}))$.

Prova: Pelo Teorema 48 existe uma árvore PQR T com $\overline{\mathcal{C}} = Compl(T)$. O Teorema 51 garante que T não tem nós R e então podemos escrever

$$Compl(T) = Consec(Compat(T)) = Consec(Valid(\overline{\mathcal{C}})),$$

com base, respectivamente, nos Teoremas 49 e 50. O resultado segue então diretamente do Teorema 26. $\square$

A coleção $Orto(T)$, definida abaixo, será usada para resolver alguns problemas colocados no Capítulo 5. Na seqüência apresentamos algumas características dessa coleção.

Definição 54 *A coleção* $Orto(T)$ *é assim construída:*

1. $\mathcal{T}(U)$ *está contida em* $Orto(T)$.
2. *Se* S *é um conjunto qualquer de filhos de um nó* P *de* T, *então* $Desc_T(S) \in Orto(T)$.
3. *Se* S *é o conjunto de todos os filhos de um nó* Q *de* T, *então* $Desc_T(S) \in Orto(T)$.
4. *Se* S *é o conjunto de todos os filhos de um nó* R *de* T, *então* $Desc_T(S) \in Orto(T)$.
5. *Nenhum outro conjunto está em* $Orto(T)$.

Exemplo 55 *A coleção* $Orto(T)$ *para a árvore da Figura 6 (b) é formada pelos conjuntos*

$\mathcal{T}(U)$
$\{ghi, hi\}, \{hi\}$ *(filhos dos nós P)*
$\{abcdefghi\}$ *(filhos do nó Q)*
$\{abc\}$ *(filhos do nó R).*

Lema 56 *Para uma árvore PQR* T,

$$Compl(T) \perp Orto(T).$$

Prova: Observamos primeiramente que é suficiente considerar um único nó v de T ao analisar $Compl(T)$ e $Orto(T)$, pois se considerarmos nós distintos, os conjuntos gerados a partir deles serão sempre ortogonais, independentemente dos tipos dos nós. Tomemos um nó v de T qualquer e consideremos os possíveis tipos de v:

- O nó v é do tipo P: Neste caso, em $Compl(T)$ teremos um conjunto formado por todas as folhas que dele descendem, chamemo-lo A, e em $Orto(T)$ conjuntos B_i formados por quaisquer folhas descendendo de v. Então está claro que $B_i \subseteq A$ para todo B_i em $Orto(T)$ e estes conjuntos são ortogonais.

- O nó v é do tipo Q ou R: Neste caso, teremos em $Orto(T)$ um conjunto B formado por todos os filhos de v. Em $Compl(T)$ teremos conjuntos A_i de filhos de v e é claro que $A_i \subseteq B$ para todo conjunto A_i em $Compl(T)$. Estes conjuntos são claramente ortogonais.

Finalmente, como $\mathcal{T}(U)$ é sempre ortogonal a qualquer conjunto, $Compl(T) \perp Orto(T)$. □

Lema 57 *Dados $(U,\mathcal{C})$ e uma árvore PQR T tal que $Compl(T) = \overline{\mathcal{C}}$,*

$Orto(T) = \mathcal{C}^\perp$.

Prova: Pelo Teorema 48 e pelo Lema 56, $\overline{\mathcal{C}} \perp Orto(T)$. Então $Orto(T) \subseteq (\overline{\mathcal{C}})^\perp$. Pelo Teorema 32, $Orto(T) \subseteq \mathcal{C}^\perp$.

Para mostrar que $\mathcal{C}^\perp \subseteq Orto(T)$, suponhamos $A \in \mathcal{C}^\perp$ e marquemos os elementos de A na fronteira de T. Vamos executar o seguinte procedimento e denomiar M o conjunto de nós marcados após seu término.

Enquanto existir um nó v com todos os filhos marcados **faça**
Desmarque os filhos de v e marque v.

Podemos fazer $A = \bigcup_{v \in M} Desc_T(v)$.

Se M é um conjunto unitário, A pertence a $Orto(T)$ por definição. Se M tem pelo menos dois elementos, vamos mostrar que eles devem ser irmãos. Para começar, se houvesse entre eles relação "pai-filho", o filho não poderia ter permanecido marcado. A mesma observação vale para relações entre ancestrais: "avo-neto" etc. Seja então v o ancestral comum de menor altura de dois nós $x, y \in M$. Se existir um nó z no caminho de x a v, então z não está marcado e há uma folha a que descende de z mas não pertence a A. Independentemente do tipo do nó z, $A \not\perp Desc_T(z)$, uma contradição a $A \in \mathcal{C}^\perp \Longrightarrow A \in (\overline{\mathcal{C}})^\perp \Longrightarrow (A \in Compl(T)^\perp)$. Então não há nós intermediários entre

x e v e argumentos semelhantes valem para outros nós de M. Como conseqüência, M é um conjunto de irmãos. Seja então v o nó pai. Se v é um nó P, $A \subseteq Desc_T(v)$ e pertence a $Orto(T)$ por definição. Se v é um nó Q ou R e nem todos os filhos de v foram marcados, existe $B \in Compl(T)$ tal que $A \not\perp B$. Logo, todos os seus filhos foram marcados e $A \in Orto(T)$. □

Lema 58 *Para qualquer árvore PQR T e qualquer nó v de T, temos que*

$$Desc_T(v) \in Compl(T) \cap Orto(T).$$

Ainda, qualquer conjunto em $Compl(T) \cap Orto(T)$ é da forma $Desc_T(v)$ para algum nó v de T.

Prova: Vamos mostrar que $Desc_T(v) \in Compl(T)$. Se v é uma folha, então $Desc_T(v)$ é trivial e a relação é válida. Se v é um nó interno, podemos tomar um conjunto S de todos os filhos de v e por definição, independentemente do tipo de v, $Desc_T(S) \in Compl(T)$ e ainda, neste caso, $Desc_T(S) = Desc_T(v)$. Analogamente, como um conjunto de todos os filhos de um nó v pertence sempre a $Orto(T)$, $Desc_T(v) \in Orto(T)$.

Suponhamos agora que um conjunto $H \in Compl(T)$ também pertence a $Orto(T)$. Se H é trivial (conjunto de filhos da raiz ou de uma folha), H está na forma enunciada. Se $H = Desc_T(S)$ para algum conjunto S de filhos de um nó v, independentemente do tipo de v, S deve incluir todos os seus filhos por definição, ora de $Compl(T)$, ora de $Orto(T)$. Então H sempre está na forma $Desc_T(v)$ para algum nó v de T. □

Agora que estabelecemos as principais relações entre $\overline{\mathcal{C}}$, $\mathcal{C}^\perp$ e as árvores PQR, vamos demonstrar propriedades das árvores equivalentes.

Lema 59 *Dadas duas árvores PQR T e T' sobre o mesmo conjunto U, temos,*

$$Compl(T) = Compl(T') \iff T \equiv T'.$$

Prova: Uma vez que as regras para formar a coleção $Compl(T)$ são invariantes sob as transformações de equivalência, duas árvores equivalentes resultam na mesma coleção completa, o que demonstra um lado da relação.

Para mostrar o outro lado, $Compl(T) = Compl(T') \Longrightarrow T \equiv T'$, vamos usar indução no número de elementos de U. Para a base da indução tomemos $|U| = 2$, caso em que a árvore possui um nó P como raiz de suas duas folhas e em que as árvores são certamente equivalentes.

Se $|U| \geq 3$, consideremos inicialmente o caso em que a coleção $Compl(T)$ é prima. Pela prova do Teorema 48, as árvores T e T' têm apenas um nó interno, sua raiz, e aquele teorema descreve todas as possibilidades para $Compl(T)$. É claro que as raízes de T e T' devem ser do mesmo tipo. Se o tipo é P ou R, as árvores são equivalentes com certeza. Se as raízes são nós Q, para gerar a mesma coleção $Compl(T)$ é necessário que tenhamos os filhos em T' na mesma ordem na qual aparecem em T ou na sua reversa e novamente as árvores serão equivalentes.

Consideremos $H \in Compl(T) \cap Orto(T)$ um conjunto não trivial para analisarmos o caso em que $Compl(T)$ não é prima. Pelo Lema 58 existem nós v em T e v' em T' tais que

$$H = Desc_T(v) = Desc_T(v').$$

Sejam T/v a árvore PQR obtida de T substituindo-se a subárvore enraizada em v por uma nova folha $H = Desc_T(v)$ e $T \wedge v$ a subárvore enraizada em v. As igualdades a seguir são imediatas:

$$Compl(T/v) = Compl(T)/Desc_T(v)$$
$$Compl(T \wedge v) = Compl(T) \wedge Desc_T(v).$$

Já que $Compl(T) = Compl(T')$ e que $Desc_T(v) = Desc_T(v')$, obtemos,

$$Compl(T/v) = Compl(T'/v')$$
$$Compl(T \wedge v) = Compl(T' \wedge v').$$

Por hipótese de indução obtemos as equivalências:

$$T/v \equiv T'/v'$$
$$T \wedge v \equiv T' \wedge v'.$$

Estas relações implicam na equivalência de T e T'. □

Corolário 60 *A menos de transformações de equivalência, uma árvore PQR T tal que $Compl(T) = \overline{\mathcal{C}}$ é única.*

Lema 61 *Se T e T' são árvores PQR sem nós R, então,*

$$Compat(T) = Compat(T') \Longleftrightarrow T \equiv T'.$$

Prova: O sentido $\Longleftarrow$ é claro pelas definições. Do Teorema 49, derivamos que

$$Compat(T) = Compat(T') \Longrightarrow Compl(T) = Compl(T'),$$

porque nenhuma das árvores possui nós R. O resultado do sentido $\Longrightarrow$ vem, então, imediatamente, do Lema 59. □

A Figura 13 mostra que a restrição sobre os nós R é necessária ao teorema anterior.

Depois de investigar as coleções $(\overline{\mathcal{C}})^{\perp}, \overline{\mathcal{C}^{\perp}}$ e $\overline{\overline{\mathcal{C}}}$, surge o questionamento a respeito de $(\mathcal{C}^{\perp})^{\perp}$. O lema abaixo mostra o relacionamento entre $\overline{\mathcal{C}}$ e $(\mathcal{C}^{\perp})^{\perp}$.

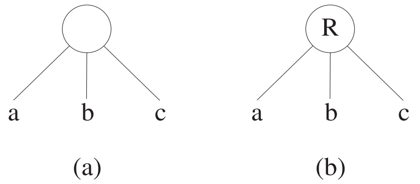

Figura 13: Árvores PQR com raízes do tipo (a) P e (b) R.

Lema 62 *Seja* $\mathcal{C} \subseteq \mathcal{P}(U)$*. Então*

$$\overline{\mathcal{C}} = (\mathcal{C}^{\perp})^{\perp}$$

se a árvore PQR para $\mathcal{C}$ *não possui nós Q.*

Prova: Seja T a árvore PQR tal que $\overline{\mathcal{C}} = Compl(T)$. Por definição,

$$\overline{\mathcal{C}} = Compl(T) = \begin{cases} \text{Conjuntos de todos os filhos de nós P.} \\ \text{Conjuntos de filhos consecutivos de nós Q.} \\ \text{Conjuntos de quaisquer filhos de nós R.} \end{cases}$$

$$\mathcal{C}^{\perp} = Orto(T) = \begin{cases} \text{Conjuntos de quaisquer filhos de nós P.} \\ \text{Conjuntos de todos os filhos de nós Q.} \\ \text{Conjuntos de todos os filhos de nós R.} \end{cases}$$

Podemos perceber que, pela construção de $\mathcal{C}^{\perp}$,

$$(\mathcal{C}^{\perp})^{\perp} \text{ em } T = \begin{cases} \text{Conjuntos de todos os filhos de nós P.} \\ \text{Conjuntos de quaisquer filhos de nós Q.} \\ \text{Conjuntos de quaisquer filhos de nós R.} \end{cases}$$

Uma comparação permite concluir que $\overline{\mathcal{C}} = (\mathcal{C}^{\perp})^{\perp}$ se T não possui nós Q. □

Definimos agora a coleção canônica para uma árvore PQR.

Definição 63 *A* **coleção canônica** *para uma árvore PQR* T*,* $Can(T)$*, é assim construída:*

1. *Se* S *é o conjunto de todos os filhos de um nó P,* $Desc_T(S)$ *está em* $Can(T)$*.*

2. *Se* S *é um conjunto de um par de filhos consecutivos de nó Q,* $Desc_T(S)$ *está em* $Can(T)$*.*

3. *Fixada uma ordem qualquer dos filhos de um nó v do tipo R, se S é um conjunto de um par de filhos consecutivos de v, ou o par formado pelo primeiro e pelo último filhos de v, na ordem, $Desc_T(S)$ está em $Can(T)$.*

4. *Nenhum outro conjunto está em $Can(T)$.*

A coleção canônica é uma coleção com poucos conjuntos que representa $\mathcal{C}$, pois $\overline{Can(T)} = Compl(T)$, o que é demonstrado a seguir. A coleção $Can(T)$ não é a menor possível para representar a árvore, como mostra o Exemplo 65, mas $|Can(T)| \leq |Compl(T)|$, o que torna esta coleção mais atraente para a operação de união de árvores na Seção 5.6.4.

Lema 64 *Para uma árvore PQR T,*

$$\overline{Can(T)} = Compl(T).$$

Prova: Para mostrar que $Compl(T) \subseteq \overline{Can(T)}$, seja S um conjunto de filhos de um nó v de T tal que $Desc_T(S) \in Compl(T)$. Se v é um nó P, $Desc_T(S) \in Can(T)$ por definição e então $Desc_T(S) \in \overline{Can(T)}$. Se v é um nó Q e S não é um par de filhos consecutivos de v, $Desc_T(S)$ pode ser obtido pela união não disjunta de conjuntos em $Can(T)$, pois S é um conjunto consecutivo de filhos. Finalmente, se v é um nó R, como $Can(T)$ contém descendentes de pares circulares de filhos consecutivos de v, qualquer conjunto em $Compl(T)$ pode ser obtido aplicando as operações de fecho ao redor do ciclo.

Para mostrar que $\overline{Can(T)} \subseteq Compl(T)$, seja S um conjunto de filhos de um nó v de T tal que $Desc_T(S) \in \overline{Can(T)}$. Independentemente do tipo de v, $Desc_T(S)$ se enquadra na definição de um conjunto de $Compl(T)$.

Podemos concluir que $\overline{Can(T)} = Compl(T)$. □

Exemplo 65 *Seja $\mathcal{C} = \{abc, cd\}$. A árvore PQR para $\mathcal{C}$ aparece na Figura 14. A coleção $Can(T) = \{ab, abc, cd\}$ é maior que $\mathcal{C}$.*

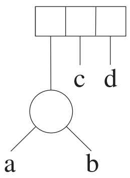

Figura 14: Árvore PQR para a coleção $\mathcal{C} = \{abc, cd\}$.

Capítulo 4

Construção das Árvores PQR

Neste capítulo apresentamos dois algoritmos para a construção das árvores PQR. O primeiro é o algoritmo recursivo de Meidanis e Munuera e o segundo, um algoritmo não recursivo. Neste capítulo analisamos ainda o custo de um algoritmo incremental para as árvores PQ e PQR.

4.1 O Algoritmo de Meidanis e Munuera

O algoritmo de Meidanis e Munuera [MM96] baseia-se no Teorema 48, construindo a árvore PQR T para uma coleção $\mathcal{C}$ de tal forma que $Compl(T) = \overline{\mathcal{C}}$. Ao contrário do algoritmo incremental de Booth e Lueker para as árvores PQ, que processa os conjuntos em $\mathcal{C}$ um a um, considerando-os uma restrição à fronteira da árvore que está sendo construída, o algoritmo recursivo de Meidanis e Munuera, apresentado na Figura 15, processa todos os conjuntos ao mesmo tempo. A correção do algoritmo é garantida pelos resultados apresentados no Capítulo 3. Antes de analisar a complexidade do algoritmo, vamos detalhar cada uma das tarefas desempenhadas por ele. São elas: encontrar conjuntos não triviais em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$; dividir a coleção $\mathcal{C}$, substituindo os elementos de H em todo conjunto que contém H e em U e verificar a P1C para instâncias primas.

Para encontrar conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$, Meidanis e Munuera sugerem dois métodos que, se aplicados corretamente, são suficientes para obter todos os conjuntos dessa interseção. São eles: encontrar uniões de componentes conexas do grafo de sobreposições estritas e encontrar classes de equivalência da relação de gêmeos em $\mathcal{C}$. A seguir formalizamos as relações entre esses métodos e os conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$, apresentando as definições e demonstrações necessárias.

Definição 66 *Dois conjuntos A e B se* **sobrepõem estritamente** *se $A \cap B \neq \emptyset$ e $A \not\subseteq B$ e $B \not\subseteq A$.*

Árvore PQR **Função** $PQR(U, \mathcal{C})$
  Se existe um conjunto não trivial H em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$ **então**
    $T_1 \leftarrow PQR(U/H, \mathcal{C}/H)$;
    $T_2 \leftarrow PQR(H, \mathcal{C} \wedge H)$;
    Retorne(T_1 com a folha H substituída por T_2);
  Senão
    {$\mathcal{C}$ é prima}
    Escolha
      $\mathcal{C}$ é trivial: **Retorne**(*Nó-P*(U));
      Existe permutação α válida em $\mathcal{C}$: **Retorne**(*Nó-Q*(α));
      Senão: **Retorne**(*Nó-R*(U));

Figura 15: Algoritmo recursivo para a construção de árvores PQR.

Definição 67 *O* **grafo de sobreposições estritas**, *apresentado originalmente por Fulkerson e Gross [FG65], $G(\mathcal{C})$, tem um vértice para cada conjunto em $\mathcal{C}$ e uma aresta entre dois vértices se e somente se os conjuntos a que correspondem se sobrepõem estritamente.*

Para simplificar, quando nos referirmos aos vértices de $G(\mathcal{C})$, indistintamente usaremos o termo "conjunto" significando "o conjunto que corresponde ao vértice". Da mesma forma, diremos "união de componente" com o significado de "união dos conjuntos correspondentes aos vértices da componente".

Antes de mostrar que as uniões de componentes de $G(\mathcal{C})$ são conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$, vamos apresentar um resultado necessário a algumas provas nesta seção.

Lema 68 *Se A, B e H são subconjuntos de U sendo que $A \subseteq H$, $B \perp H$ e $A \not\perp B$, então $B \subseteq H$.*

Prova: O fato de que $B \perp H$ nos dá três possibilidades. Se $H \subseteq B$ então $A \subseteq B$, uma contradição. Se $H \cap B = \emptyset$, então A seria ortogonal a B, outra contradição. A única possibilidade que nos restou, $B \subseteq H$, é verdadeira. □

Teorema 69 *Seja X uma componente conexa de $G(\mathcal{C})$. Então*

$$H = \bigcup_{A \in X} A$$

é um conjunto em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$.

Prova: O conjunto H pode ser obtido por uniões não disjuntas de todos os conjuntos de X, pela própria definição de $G(\mathcal{C})$, e sendo assim, $H \in \overline{\mathcal{C}}$. Por construção, é claro que $A \subseteq H$ e então $H \perp A$, para todo $A \in X$. Se o grafo tem apenas uma componente, $H \in \mathcal{C}^{\perp}$. Se o grafo possui mais de uma componente, tomemos um conjunto B de outra componente qualquer. Então temos três possibilidades:

1. $A \cap B = \emptyset$ para todo $A \in X$: Neste caso $B \cap H = \emptyset$ e $H \perp B$.

2. $B \subset A$, para algum $A \in X$: Então $B \subseteq H$ e $H \perp B$.

3. $A \subseteq B$, para algum $A \in X$: Neste caso afirmamos que todo conjunto em X deve estar contido em B. Seja $C \in X$ tal que $A \not\perp C$. Como B e C estão em componentes distintas de $G(\mathcal{C})$, $B \perp C$. Pelo Lema 68, temos que $C \subseteq B$. A aplicação desse lema a todos os vértices adjacentes a A, depois aos adjacentes àqueles e assim por diante permite concluir que $A \subseteq B$ para todo $A \in X$.

Como não há outra possibilidade, concluímos que $H \in \mathcal{C}^\perp$. Segue diretamente que $H \in \overline{\mathcal{C}} \cap \mathcal{C}^\perp$. □

Apresentamos agora uma característica do grafo de sobreposições estritas que será usada em seguida.

Lema 70 *Se todas as componentes de $G(\mathcal{C})$ têm união trivial, então todos os conjuntos não triviais em $\mathcal{C}$ pertencem à mesma componente conexa de $G(\mathcal{C})$.*

Prova: Suponhamos que existem duas componentes X e Y de $G(\mathcal{C})$ com união trivial contendo conjuntos não triviais. Sabemos que

$$\bigcup_{A \in X} A = \bigcup_{B \in Y} B = U.$$

Sejam $A \in X$ e $B \in Y$ tais que $A \cap B \neq \emptyset$. Como $A \perp B$, pois estão em componentes distintas, suponhamos, sem perda de generalidade, que $A \subseteq B$. Seja $C \in X$ tal que $A \not\perp C$. Então $C \perp B$, pois estão em componentes distintas. Pelo Lema 68, $C \subseteq B$. Se estendermos esta verificação a todos os vértices de X, considerando primeiro os adjacentes a A, depois os adjacentes àqueles e assim sucessivamente, poderemos concluir que todos os conjuntos de X estão contidos em B. Mas isso não é possível, pois $B \neq U$. □

A outra forma de encontrar conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^\perp$ é através da relação de gêmeos. Esta é uma relação de equivalência cujas classes de equivalência são conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^\perp$ desde que todas as uniões de componentes em $G(\mathcal{C})$ sejam triviais, como demonstramos a seguir.

Definição 71 *Dois elementos, $a, b \in U$ são* **gêmeos** *em relação a $\mathcal{C}$ se e somente se o conjunto binário $\{a, b\}$ pertence a $\mathcal{C}^\perp$.*

Lema 72 *A relação $\delta \subseteq U \times U$ tal que, dados $a, b \in U$, a e b são gêmeos em relação a $\mathcal{C}$ se $ab \in \mathcal{C}^\perp$, é de equivalência.*

Prova:

- Reflexiva: Para todo $a \in U$, $aa = a \in \mathcal{T}(U) \subseteq \mathcal{C}^{\perp}$.
- Simétrica: Para todo $a, b \in U$, se $ab \in \mathcal{C}^{\perp}$ então $ba \in \mathcal{C}^{\perp}$, já que $ab = ba$.
- Transitiva: Sejam $a, b, c \in U$ e $ab, bc \in \mathcal{C}^{\perp}$. A Tabela 3 descreve o que ocorre com o conjunto ac em função dos conjuntos ab e bc e de um conjunto $C \in \mathcal{C}$ qualquer e nos permite concluir que $ac \in \mathcal{C}^{\perp}$ em todas as situações válidas. Os casos marcados com $*$ são contradições ao fato de ab e bc pertencerem simultaneamente a $\mathcal{C}^{\perp}$.

$\square$

	$ab \cap C = \emptyset$	$ab \subseteq C$	$C \subseteq ab$
$bc \cap C = \emptyset$	$ac \cap C = \emptyset$	$*$	$C = \{a\}$ ou $\emptyset : C \subset ac$
$bc \subseteq C$	$*$	$ac \subseteq C$	$*$
$C \subseteq bc$	$C = \{c\}$ ou $\emptyset : C \subset ac$	$*$	$C = \{b\}$ ou $\emptyset : ac \cap C = \emptyset$

Tabela 3: Casos possíveis para demonstrar que a relação de gêmeos é transitiva.

Lema 73 *As classes de equivalência da relação de gêmeos δ descrita no Lema 72 são conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$ desde que todas as uniões de componentes em $G(\mathcal{C})$ sejam triviais.*

Prova: Seja H uma classe de equivalência de δ. Como a relação de gêmeos é de equivalência e cada par em uma classe de equivalência H está contido em $\mathcal{C}^{\perp}$, $H \in \mathcal{C}^{\perp}$.

Para mostrar que $H \in \overline{\mathcal{C}}$ vamos supor H não trivial. Vamos particionar os conjuntos não triviais de $\mathcal{C}$ em $A_1, A_2, \ldots, A_k$ tais que $H \subseteq A_i$, $1 \leq i \leq k$, e em $B_1, B_2, \ldots, B_l$, tais que $H \cap B_j = \emptyset$, $1 \leq j \leq l$. Note que não há não triviais $C \in \mathcal{C}$ tais que $C \subset H$.

Sabemos que $k \geq 1$, ou seja, ao menos um conjunto não trivial em $\mathcal{C}$ contém H, pois H não é trivial, todas as componentes conexas de $G(\mathcal{C})$ têm união trivial e todos os conjuntos não triviais em $\mathcal{C}$ estão em uma única componente conexa de $G(\mathcal{C})$ (Lema 70). Vamos supor também que $B_1, B_2, \ldots, B_l$ foram ordenados por uma busca em largura iniciada em A_1.

Vamos mostrar por indução em j que, para todo j entre 0 e l,

$$H_j = (\ldots(A_1 \cap A_2 \cap \ldots \cap A_k) \setminus B_1) \setminus B_2) \setminus \ldots) \setminus B_j) \in \overline{\mathcal{C}}.$$

Como um caso base tomemos $j = 0$. Nesse caso $H_0 = (A_1 \cap A_2 \cap \ldots \cap A_k) \in \overline{\mathcal{C}}$. Suponhamos que, para $j > 0$,

$$(\ldots(A_1 \cap A_2 \cap \ldots \cap A_k) \setminus B_1) \setminus B_2) \setminus \ldots) \setminus B_{j-1}) \in \overline{\mathcal{C}}.$$

Já que B_j não é trivial e pertence à mesma componente conexa que os outros conjuntos, existe um conjunto A_i ou B_i, $i < j$, adjacente a ele. Consideremos então os dois casos:

1. Se $B_j \not\perp A_i$ então $B_j \nsubseteq A_i$ e assim

$$B_j \not\subset (\ldots(A_1 \cap A_2 \cap \ldots \cap A_k) \setminus B_1) \setminus B_2) \setminus \ldots) \setminus B_{j-1}),$$

o que implica que

$$(\ldots(A_1 \cap A_2 \cap \ldots \cap A_k) \setminus B_1) \setminus B_2) \setminus \ldots) \setminus B_j) \in \overline{\mathcal{C}}$$

por diferença não contida.

2. Se $B_j \not\perp B_i$ então $B_j \cap B_i \neq \emptyset$ e assim

$$B_j \not\subset (\ldots(A_1 \cap A_2 \cap \ldots \cap A_k) \setminus B_1) \setminus B_2) \setminus \ldots) \setminus B_{j-1}),$$

o que implica que

$$(\ldots(A_1 \cap A_2 \cap \ldots \cap A_k) \setminus B_1) \setminus B_2) \setminus \ldots) \setminus B_j) \in \overline{\mathcal{C}}$$

também por diferença não contida.

O resultado segue observando que $H = H_l$.

□

O teorema abaixo mostra que os dois métodos enunciados são suficientes para encontrar todos os conjuntos não triviais em $\overline{\mathcal{C}} \cap \mathcal{C}^\perp$.

Teorema 74 *Se $\mathcal{C}$ é uma coleção tal que*

- *todas as uniões de componentes em $G(\mathcal{C})$ são triviais e*
- *$\mathcal{C}$ não possui gêmeos,*

então $\mathcal{C}$ é prima.

Prova: O resultado é óbvio se $\mathcal{C} \subseteq \mathcal{T}(U)$. Então vamos supor que $\mathcal{C}$ tem pelo menos um conjunto não trivial, que todas as uniões de $G(\mathcal{C})$ são triviais e que $\mathcal{C}$ não possui gêmeos. Vamos provar o resultado por contradição. Suponhamos que H é um conjunto não trivial em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$. Uma vez que H não é trivial, há em U pelo menos dois elementos distintos a e b que pertencem a H e pelo menos um elemento c que não pertence a H. Por hipótese, a e b não são gêmeos e então há um conjunto $A \in \mathcal{C}$ que separa a e b, ou seja, $ab \not\perp A$. Mas como $A \perp H$, necessariamente $A \subseteq H$. Como todas as componentes de $G(\mathcal{C})$ têm união trivial, pelo Lema 70 sabemos que existe um conjunto não trivial $C \in \mathcal{C}$ tal que $c \in C$, na mesma componente conexa de A em $G(\mathcal{C})$. Há, então, um caminho

$$A \not\perp A_1 \not\perp \ldots \not\perp A_k \not\perp C$$

em $G(\mathcal{C})$ que leva de A a C. Pelo Lema 68 (e lembrando que $H \in \mathcal{C}^{\perp}$), $A_1 \subseteq H$ e a aplicação repetida desse lema eventualmente resulta que $C \subseteq H$, mas isso contradiz o fato de que $c \notin H$. □

A seguir definimos a partição mais grossa de um conjunto U em relação a uma coleção $\mathcal{C}$ e mostramos que os blocos dessa partição são iguais às classes de equivalência da relação de gêmeos. Esse resultado nos dá um método algorítmico para encontrar estas classes.

Deste ponto em diante, os resultados apresentados referem-se todos a **coleções sem conjuntos triviais**. Já que tais conjuntos são sempre consecutivos em qualquer permutação de U, sua eliminação não enfraquece os resultados.

Definição 75 *Uma* **partição** *de um conjunto U em relação a uma coleção $\mathcal{C}$, é a divisão de U em conjuntos (blocos) não vazios, disjuntos dois a dois de tal forma que:*

- *A união de todos os blocos é igual a U.*
- *Todo conjunto em $\mathcal{C}$ pode ser formado pela união de blocos da partição.*

Dadas duas partições X e Y de U em relação a $\mathcal{C}$, dizemos que X **refina** *Y se, para todo bloco $A \in X$ e $B \in Y$, $A \subseteq B$.*

A **partição mais grossa** *de um conjunto U em relação a uma coleção $\mathcal{C}$ é aquela que não refina nenhuma outra diferente dela.*

Podemos perceber, pela definição, que se $\bigcup_{A \in \mathcal{C}} A \neq U$ então um bloco da partição mais grossa de U conterá elementos que nunca aparecem em conjuntos de $\mathcal{C}$. Este bloco será chamado **bloco externo**. Podemos ver ainda que se as uniões de componentes do grafo $G(\mathcal{C})$ são todas triviais, então o bloco externo da partição será vazio ou igual a U, este último caso só acontecendo quando $\mathcal{C}$ for vazia. Quando for claro pelo contexto, nos referiremos à partição mais grossa de U em relação a $\mathcal{C}$ apenas como partição mais grossa.

Apresentamos agora o resultado que relaciona os blocos da partição mais grossa às classes de equivalência da relação de gêmeos.

Lema 76 *Seja $\mathcal{C}$ uma coleção sem conjuntos unitários e tal que todas as uniões de $G(\mathcal{C})$ são triviais. Então os blocos não triviais na partição mais grossa de U em relação a $\mathcal{C}$ são iguais a classes de equivalência de gêmeos da relação de gêmeos δ definida no Lema 72.*

Prova: Seja H uma classe de equivalência de δ. Pelo Lema 73, $H \in \overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$. Suponhamos que H não está contido em um único bloco da partição mais grossa, mas dividido entre os blocos A e B. Por construção da partição mais grossa e como o bloco externo é vazio, A aparece em pelo menos um conjunto de $\mathcal{C}$ que não contém B, sem perda de generalidade. Mas isto é uma contradição ao fato de que $H \in \mathcal{C}^{\perp}$ e então H deve estar contido em um único bloco da partição mais grossa.

Por outro lado, seja H um bloco da partição mais grossa de U em relação a $\mathcal{C}$. Suponhamos que H não está contido em uma única classe de equivalência de δ mas dividido entre pelo menos duas classes A e B distintas. Neste caso, existe pelo menos um conjunto $S \in \mathcal{C}$ tal que $A \subseteq S$ e $B \not\subseteq S$ ou $A \not\subseteq S$ e $B \subseteq S$. Evidentemente, S deve refinar H, que deixa de ser um bloco da partição mais grossa. Desta contradição segue a validade da prova. $\square$

Vamos investigar agora os algoritmos para implementar esses métodos, analisando suas complexidades. Na Figura 16 apresentamos um algoritmo para encontrar as uniões de componentes do grafo de sobreposições estritas para uma coleção $\mathcal{C}$. O Lema 77 mostra que o algoritmo é correto. Sua complexidade é analisada no Lema 78.

Na primeira parte do algoritmo, os conjuntos de $\mathcal{C}$ são pré-processados. Para cada conjunto S de $\mathcal{C}$ é construído o vetor S^{-1}. Cada posição i de S^{-1} indica se o elemento i pertence ou não a S. Por exemplo, se $U = \{1,2,3,4,5,6\}$ e $S = \{2,4,5\}$, então $S^{-1}[1] = 0, S^{-1}[2] = 1, S^{-1}[3] = 0, S^{-1}[4] = 1, S^{-1}[5] = 1$ e $S^{-1}[6] = 0$. A posição 0 de S^{-1}, $S^{-1}[0]$, é usada como um contador para calcular o tamanho da interseção entre conjuntos. O grafo é construído calculando o tamanho da interseção entre um conjunto S, sendo processado, e os conjuntos A já colocados no grafo, marcando os elementos de S em A^{-1}.

Na última parte do algoritmo uma busca em largura é usada para percorrer cada componente construindo o conjunto união Z. Para construir Z, o vetor γ é usado para marcar os elementos que já estão em Z e a variável l nos dá o tamanho de Z.

Lema 77 *O algoritmo na Figura 16 retorna exatamente as uniões de componentes conexas de $G(\mathcal{C})$.*

```
Lista de Conjuntos Função UniComp(n, C)
    inteiro Z[n];            {Conjunto união de uma componente.}
    inteiro γ[n] ← 0;        {Vetor para marcação de elementos em Z.}
    inteiro l;               {Tamanho de Z.}
    inteiro i;
    Grafo G(C) ← ∅;
    Lista de Conjuntos Uniões ← ∅;

    Para cada conjunto S ∈ C faça
        Para i ← 0 até n faça
            S^{-1}[i] ← 0;
        Para i ← 1 até |S| faça
            S^{-1}[S[i]] ← 1;

    Para cada conjunto S ∈ C faça
        Para cada elemento i ∈ S faça
            Para cada conjunto A ∈ G(C) faça
                Se A^{-1}[i] ≠ 0 então
                    A^{-1}[0] ← A^{-1}[0] + 1;
                    Marque A;
        Adicione S a G(C);
        Para cada conjunto A ∈ G(C) marcado faça
            Se A^{-1}[0] > 0 e A^{-1}[0] ≠ |A| e A^{-1}[0] ≠ |S| então
                Adicione (S, A) a G(C);
            A^{-1}[0] ← 0;
            Desmarque A;

    Para cada vértice S não marcado em G(C) faça
        l ← 0;
        Para cada conjunto A encontrado por uma busca em largura
        iniciada em S (inicialmente S = A) faça
            Marque A;
            Para cada elemento i ∈ A faça
                Se γ[i] = 0 então
                    γ[i] ← 1;
                    l ← l + 1;
                    Z[l] ← i;

        Adicione Z ao fim da lista Uniões;
        Para i ← 1 até l faça
            γ[Z[i]] ← 0;
    Retorne(Uniões);
```

Figura 16: Algoritmo para obter as uniões de componentes conexas de $G(\mathcal{C})$.

Prova: A prova de que o grafo de sobreposições estritas é construído na primeira parte do algoritmo é obtida por uma indução simples no número de conjuntos de $\mathcal{C}$. Na última parte, cada uma das componentes de $G(\mathcal{C})$ é percorrida por uma busca em largura, o que garante que todos os conjuntos na componente serão considerados. A união de cada componente é obtida à medida que cada conjunto é alcançado pela busca e, depois que toda a componente é considerada, o conjunto união é adicionado à lista que é retornada pelo algoritmo. □

Lema 78 *O custo de construção do grafo de sobreposições estritas para uma coleção $\mathcal{C}$ é $O((n+m+r)m)$.*

Prova: Para construir os vetores S^{-1} serão necessárias $O(mn)$ operações. A parte do algoritmo que constrói o grafo tem três iterações aninhadas. A mais externa processa todos os m conjuntos. A mais interna processa no máximo m conjuntos. A intermediária é composta de duas outras, uma que processa cada elemento de cada conjunto e outra que processa os conjuntos que têm interseção não vazia com o conjunto sendo processado. A primeira iteração intermediária e a mais interna executam $O(r)$ operações. A segunda intermediária, $O(m)$. A mais externa realiza então $O(rm+m^2)$ operações.

Podemos supor que nosso grafo será representado como uma matriz de adjacências $m \times m$, de implementação bastante simples. Para as inicializações serão necessárias $O(m^2 + n)$ operações. O custo total para aplicar as buscas em largura em $G(\mathcal{C})$ será $O(m^2)$, inspecionando as colunas da matriz e usando um vetor de marcações para as colunas já visitadas. As uniões de conjuntos serão obtidas em tempo $O(r)$, gerando uma lista de no máximo $O(m)$ conjuntos. O custo total do algoritmo será então $O((n+m+r)m)$. □

Hsu [Hsu92, Hsu97] apresenta uma solução alternativa para esta construção. Ao invés de construir o grafo $G(\mathcal{C})$, Hsu constrói um subgrafo G' de $G(\mathcal{C})$ que tem as mesmas uniões de componentes de $G(\mathcal{C})$, em tempo $O(n+m+r)$. No entanto, não é claro como o autor obtém este limite para a complexidade de seu algoritmo, por isso optamos por propor um algoritmo menos elaborado e de maior custo.

Se todas as uniões de componentes de $G(\mathcal{C})$ são triviais, devemos procurar por blocos não triviais da partição mais grossa de U em relação a $\mathcal{C}$. Faremos isto com um algoritmo que, além de retornar os blocos da partição mais grossa, verifica se a componente possui a P1C, pois esse teste é necessário quando a coleção é prima e não trivial, caso em que todos os conjuntos não triviais de $\mathcal{C}$ estão na mesma componente conexa de $G(\mathcal{C})$.

Para testar a P1C de componentes de $G(\mathcal{C})$, propomos um algoritmo baseado na idéia de Hsu [Hsu92] e que tira vantagem das sobreposições entre os conjuntos de uma mesma componente conexa de $G(\mathcal{C})$. Podemos modificar ligeiramente o algoritmo da Figura 16 (sem causar mudança na sua complexidade assintótica) para que ele construa, além da lista das uniões de componentes, uma lista dos conjuntos de cada componente na mesma ordem dada pela busca em largura. Esta ordem, que chamaremos Ψ, será tal que todo conjunto A_i, $i > 1$, na lista, se sobrepõe estritamente a algum A_j, $j < i$. Esta lista, juntamente com a cardinalidade do conjunto união, será a entrada para o algoritmo que testa a P1C.

O algoritmo constrói um grafo caminho G_{P1C}. Cada vértice em G_{P1C} é um subconjunto de elementos de U. A construção de G_{P1C} se baseia no fato de que, tomando dois conjuntos $A, B \in \mathcal{C}$, a união desses conjuntos pode ser particionada em $A \setminus B$, $A \cap B$ e $B \setminus A$. Essa partição determina uma ordem única (a menos de sua reversa) de blocos de elementos de $A \cup B$ em uma permutação válida. Os demais conjuntos em $\mathcal{C}$ podem refinar esses blocos mas sua ordem deve ser mantida para que A e B continuem consecutivos. O grafo G_{P1C} é aumentado à medida que seus vértices são refinados por conjuntos de $\mathcal{C}$. Se G_{P1C} não puder ser mantido um grafo caminho que tem $\mathcal{C}$ consecutiva, $\mathcal{C}$ não possui a P1C. Os vértices de G_{P1C} são refinados por cada conjunto de $\mathcal{C}$, da mesma forma que os blocos da partição mais grossa.

O algoritmo inicializa o grafo G_{P1C} construindo conjuntos para a interseção e diferença dos dois primeiros conjuntos de $\mathcal{C}$. Cada conjunto em $\mathcal{C}$, ao ser processado, marca seus elementos em G_{P1C}. As condições para que $\mathcal{C}$ tenha a P1C são verificadas. Os vértices são divididos para que sejam iguais a blocos de elementos de U em uma permutação e aos blocos da partição mais grossa. Se $\mathcal{C}$ não possuir a P1C, uma variável de controle passa a ter valor falso, que será retornado pelo algoritmo. Uma lista de conjuntos é retornada por referência e contém os blocos da partição mais grossa. Se $\mathcal{C}$ tem a P1C, os blocos na lista podem ser concatenados para obter uma permutação da união da componente válida para os conjuntos da componente.

O algoritmo aparece na Figura 17. Usaremos o termo "vértice totalmente marcado" quando todos os elementos do conjunto do vértice foram marcados pelo algoritmo e o termo "vértice parcialmente marcado" quando pelo menos um, mas não todos os elementos foram marcados. Genericamente, diremos "vértice marcado" quando o vértice foi parcial ou totalmente marcado. Finalmente, usaremos o termo "vértice não marcado" para um vértice sem nenhum elemento marcado em seu conjunto. Nos Lemas 79 e 80 analisamos a correção do algoritmo. No Lema 81 analisamos a complexidade do algoritmo e as estruturas de dados necessárias para obtê-la.

Lema 79 *O algoritmo na Figura 17 retorna uma permutação válida dos elementos de U se e somente se a coleção dos conjuntos $\mathcal{C}$, ordenada segundo Ψ, possui a P1C.*

Prova: Por indução no número de conjuntos processados pelo algoritmo. Como base, tomemos $|\mathcal{C}| = 2$. A coleção possui a P1C, pois seus dois conjuntos, A e B, se sobrepõem estritamente. Uma permutação de seus elementos dada pela concatenação de blocos

Booleano **Função** *TestaP1C*($U, \mathcal{C}$, *ListaConjuntos*)
    Grafo G_{P1C};
    Booleano *TemP1C* $\leftarrow$ verdadeiro;

    Retire os dois primeiros conjuntos, A e B, de $\mathcal{C}$;
    Inicialize G_{P1C} com os vértices $A \setminus B, A \cap B, B \setminus A$ e
    arestas $(A \setminus B, A \cap B)$ e $(A \cap B, B \setminus A)$;
    Para cada conjunto S em $\mathcal{C}$, na ordem Ψ, **faça**
        Marque os elementos de S em G_{P1C};
        $w \leftarrow S \setminus$ Vértices(G_{P1C});
        Se há pelo menos um vértice parcialmente marcado ou não marcado entre
        dois marcados **então**
            TemP1C $\leftarrow$ falso;
        Se $w \neq \emptyset$ e não há vértice de grau 1 marcado totalmente **então**
            TemP1C $\leftarrow$ falso;
        Para todo vértice v parcialmente marcado **faça**
            $v' \leftarrow S \cap v$;
            $v'' \leftarrow v \setminus S$, se houver;
            Torne v' adjacente a v'';
            Se *TemP1C* **então**
                Torne o vértice adjacente a v marcado adjacente a v';
                Torne o vértice adjacente a v não marcado adjacente a v'', se houver;
            Senão
                Se v tem grau 2 **então**
                    Torne um dos vértices adjacentes a v adjacente a v' e o outro a v'';
                Senão
                    Torne o vértice adjacente a v adjacente a v';
            Remova v e as arestas incidentes a v de G_{P1C};
        Se $w \neq \emptyset$ **então**
            Se *TemP1C* **então**
                Acrescente w a G_{P1C} e conecte-o ao vértice de grau 1 marcado totalmente;
            Senão
                Acrescente w a G_{P1C} e conecte-o a um vértice de grau 1;

    Para cada vértice v de G_{P1C}, começando em um de grau 1 e percorrendo o caminho **faça**
        Crie um conjunto A com os elementos de v;
        Acrescente A a *ListaConjuntos*;
    Retorne(*TemP1C*);

Figura 17: Algoritmo para testar a P1C para coleções ordenadas de acordo com Ψ.

$A \setminus B \cdot A \cap B \cdot B \setminus A$ é válida para A e B. Vamos supor que para uma coleção com $k-1$ conjuntos, $\mathcal{C}_{k-1}$, $k \geq 3$, a hipótese é válida e vamos provar para uma coleção $\mathcal{C}_k$ com k conjuntos.

Por um lado, vamos supor que $\mathcal{C}_k$ tem a P1C. Por hipótese de indução, a ordem dos vértices do grafo G_{P1C} para a coleção com $k-1$ conjuntos, $G_{P1C}(k-1)$, nos dá uma permutação válida para $\mathcal{C}_{k-1}$. Vamos considerar dois casos, analisando as possibilidades depois de marcar os elementos de k em $G_{P1C}(k-1)$:

1. Todos os elementos de $\mathcal{C}_k$ estão em $G_{P1C}(k-1)$.

    - O conjunto k marcou apenas um vértice ou todos os vértices de $G_{P1C}(k-1)$ foram totalmente marcados: Isto não é possível, pois, por Ψ, k se sobrepõe estritamente a algum conjunto adicionado antes de k ao grafo.
    - O conjunto k marcou um caminho onde todos os vértices internos foram totalmente marcados: Se um ou ambos vértices marcados extremos do caminho foram marcados parcialmente, os vértices serão divididos, tornando a parte marcada do vértice adjacente ao vizinho marcado e a parte não marcada adjacente ao vizinho não marcado. Os conjuntos em $\mathcal{C}_{k-1}$ continuam consecutivos na nova permutação dada pela ordem dos vértices, pois continham totalmente ou eram disjuntos dos vértices divididos. O conjunto k também é consecutivo nessa permutação, logo, ela é válida para $\mathcal{C}_k$.
    - Algum vértice no caminho entre dois outros marcados foi marcado parcialmente ou não foi marcado: Seja v um vértice parcialmente marcado entre dois outros, w e z, marcados em $G_{P1C}(k-1)$. Por hipótese de indução, os conjuntos em $\mathcal{C}_{k-1}$ são consecutivos na ordem dada pelos blocos. Se v for dividido em v' e v'' como no algoritmo, ou um dos conjuntos em $\mathcal{C}_{k-1}$ ou k não será consecutivo nas ordens dos blocos abaixo ou suas reversas:

    $$\begin{array}{cccccc} wv'zv'' & zv'wv'' & v'wzv'' & wv'v''z & wzv'v'' & zwv'v'' \\ wv''zv' & zv''wv' & v''wzv' & wv''v'z & wzv''v' & zwv''v' \end{array}$$

    Isso mostra que $\mathcal{C}(k)$ não possui a P1C, uma contradição à nossa hipótese inicial. Analogamente, se v não foi marcado, a mesma argumentação vale para as permutações wvz, vwz e wzv.

2. Nem todos os elementos de k estão em $G_{P1C}(k-1)$. Seja j o vértice que contém os elementos em $k \setminus \bigcup_{A \in \mathcal{C}_{k-1}} A$. Vamos considerar então que nenhum vértice não marcado ou parcialmente marcado aparece entre dois marcados em $G_{P1C}(k-1)$, pois esse caso é uma contradição ao fato de $\mathcal{C}_k$ possuir a P1C. Seja v um vértice marcado do extremo de um caminho em $G_{P1C}(k-1)$, adjacente a w, marcado. Analisemos alguns casos.

- Se v tem grau 1 e foi parcialmente marcado, v foi dividido em v' e v'', e nesse caso ou k ou $\mathcal{C}_{k-1}$ não será consecutivo nas ordens abaixo ou suas reversas, considerando w o vértice marcado adjacente a v, uma contradição à hipótese de que $\mathcal{C}_k$ tem a P1C:

$$\begin{array}{cccccc} wv'jv'' & jv'wv'' & v'wjv'' & wv'v''j & wjv'v'' & jwv'v'' \\ wv''jv' & jv''wv' & v''wjv' & wv''v'j & wjv''v' & jwv''v' \end{array}$$

- Se v tem grau 2 podemos usar o mesmo argumento do item anterior considerando os vértices w e z adjacentes a v.

- Se v foi totalmente marcado e tem grau 1, j pode ser feito adjacente a v e $\mathcal{C}_k$ é consecutivo na permutação dada pela ordem dos blocos, pois $\mathcal{C}_{k-1}$ não foi perturbada pela adição de k e k não marcou um caminho que destrói a propriedade em $G_{P1C}(k-1)$.

Por outro lado, suponhamos que uma ordem α dos blocos válida para U foi retornada pelo algoritmo. Por hipótese de indução, $\mathcal{C}_{k-1}$ é consecutiva em α. Pela argumentação anterior, nenhum vértice não marcado ou marcado parcialmente aparece entre dois marcados. Então, todos os elementos de k que pertencem a $\mathcal{C}_{k-1}$ foram feitos consecutivos em $G_{P1C}(k-1)$, dividindo eventualmente vértices extremos do caminho marcado. Mais ainda, se algum elemento em k não pertence a $G_{P1C}(k-1)$, então há um vértice de grau 1 totalmente marcado e o vértice que contém os elementos de $j = k \setminus \bigcup_{A \in \mathcal{C}_{k-1}} A$ é feito adjacente a v. Logo $\mathcal{C}_k$ possui a P1C. □

Lema 80 *O algoritmo na Figura 17 retorna os blocos da partição mais grossa da união de todos os conjuntos em $\mathcal{C}$ em relação aos conjuntos de $\mathcal{C}$.*

Prova: Esta prova pode ser obtida por indução no número de conjuntos processados pelo algoritmo, observando que a cada conjunto S processado os vértices marcados são divididos de acordo com as interseções com S. □

Como uma última observação, se a coleção processada pelo algoritmo for prima, e conseqüentemente sem gêmeos, os conjuntos em cada vértice serão unitários e uma permutação dos elementos de U pode ser obtida por uma concatenação simples.

Lema 81 *O algoritmo na Figura 17 tem custo $O(n+m+r)$.*

Prova: Para processar o grafo em tempo $O(r)$, vamos construi-lo como uma lista duplamente encadeada de vértices e os elementos de U como apontadores para vértices, organizados em um vetor. A medida que os elementos de U forem sendo adicionados ao grafo, eles apontarão para os vértices a que pertencem.

Cada vértice conterá uma variável para armazenar seu tamanho e um contador de marcações, além dos apontadores para seus vizinhos. Para marcar um conjunto S em G_{P1C}, os contadores de marcação dos vértices para os quais os elementos apontam são incrementados. Além disso, com base no número de elementos marcados em cada vértice é possível verificar as condições do caminho sendo marcado e decidir se a P1C se mantém ou não.

Para dividir os vértices parcialmente marcados, basta criar um novo vértice para cada vértice marcado e percorrer S novamente, transferindo os elementos marcados para estes novos vértices. Todo o controle pode ser feito adicionando algumas informações aos vértices. Para incluir um novo vértice é só direcionar a ele os apontadores dos elementos em S que ainda não estão em G_{P1C}, durante a marcação. O ajuste dos apontadores de cada vértice pode ser feito sem problemas depois da divisão.

Toda a manipulação do grafo pode ser feita então em tempo $O(r)$. Cada um dos m conjuntos processados demanda um número constante de operações de teste. A construção da lista de conjuntos pode ser feita em tempo $O(n)$, percorrendo o vetor de elementos. Como a construção gera um grafo de no máximo n vértices, o número total de operações é então $O(n + m + r)$. $\square$

Exemplo 82 *A Figura 18 ilustra a construção proposta no Lema 81 para* $|U| = 10$. *Na parte (a), o estado inicial do grafo é mostrado para* $\mathcal{C} = \{\{1, 2, 3, 4\}, \{3, 4, 5, 6, 7\}\}$. *Na parte (b) temos as estruturas após o processamento do conjunto* $A = \{6, 7, 8\}$.

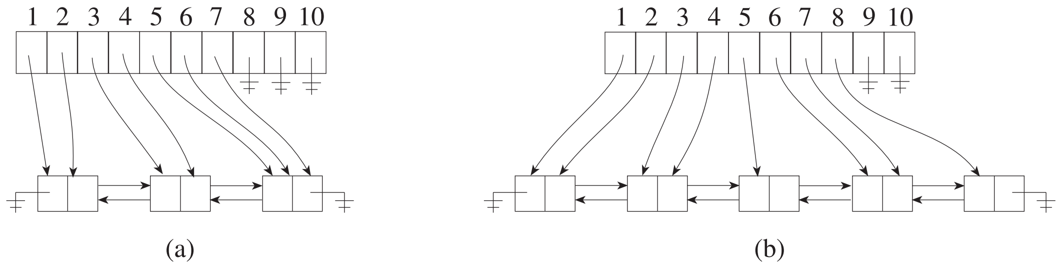

Figura 18: Construção proposta para G_{P1C}. Na parte (a), o estado inicial para $\mathcal{C}$ e na parte (b) após o processamento do conjunto A.

Antes de analisar a complexidade do algoritmo de Meidanis e Munuera vamos recapitular brevemente as tarefas desempenhadas por ele. Inicialmente, o algoritmo constrói o grafo de sobreposições estritas. Se existe uma componente com união não trivial em $G(\mathcal{C})$, essa união vai ser usada para dividir os conjuntos e fazer as chamadas recursivas. Caso contrário, o algoritmo que testa a P1C e constrói os blocos da partição mais grossa deve ser aplicado à componente que tem os conjuntos não triviais de $\mathcal{C}$. Se não existir essa componente, a coleção é trivial. Se existir, algum bloco não trivial da partição mais grossa é usado para dividir os conjuntos e fazer as chamadas recursivas. Se todos os blocos são triviais, o algoritmo que testa a P1C já retornou o resultado desse teste, o que permite escolher entre um dos casos para coleções primas.

Teorema 83 *O algoritmo de Meidanis e Munuera para a construção das árvores PQR, na Figura 15, tem custo* $O((m+n+r)^2 n^2)$.

Prova: O custo T do algoritmo recursivo para construir uma árvore PQR para uma instância I de tamanho $w = n + m + r$, onde $n = |U|$, $m = |\mathcal{C}|$ e $r = \sum_{A \in \mathcal{C}} |A|$ pode ser expresso por $T(w) = T(w_1) + T(w_2) + D(w)$, onde w_1 e w_2 são, respectivamente, os tamanhos das subinstâncias I_1 e I_2 do problema. O fator $D(w)$ é o custo para encontrar conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$, dividir e reagrupar soluções e testar a P1C para instâncias primas. Sabemos que $D(w) \leq cw^2 n$, para certo $c > 0$. Queremos mostrar que $T(w) = cw^2 n^2$.

O tamanhos das subinstâncias I_1 e I_2 da árvore PQR podem ser expressos por $w_1 = n_1 + m_1 + r_1$ e $w_2 = n_2 + m_2 + r_2$. O relacionamento entre os tamanhos das subinstâncias é dado pelas expressões:

- $n_1 + n_2 = n + 1$

  A subinstância I_1 terá $U_1 = U \setminus H \cup \{H\}$ e I_2 terá $U_2 = H$ o que faz com que a soma de seus tamanhos seja uma unidade maior que U.

- $m - 1 \leq m_1 + m_2 \leq m$

  Os conjuntos de $\mathcal{C}$ serão divididos da seguinte forma: em I_1 estarão os conjuntos que contém propriamente ou são disjuntos de H e em I_2 estarão os conjuntos contidos propriamente em H. Se $H \in \mathcal{C}$, teremos $m_1 + m_2 = m - 1$, senão, teremos $m_1 + m_2 = m$.

- $r_1 + r_2 \leq r$

  Os conjuntos que participarem da instância I_2 não serão alterados, mas os conjuntos que estiverem em I_1 e contiverem H terão alguns de seus elementos substituídos por apenas um e nesse caso $r_1 + r_2 < r$. No caso em que I_1 for formada apenas por conjuntos disjuntos de H teremos $r_1 + r_2 = r$.

Somando, temos $w_1 + w_2 \leq w + 1$.

Vamos provar a complexidade do algoritmo por indução forte. Para toda instância onde $w < 6$ o algoritmo consome tempo $T(w) \leq D(w) \leq cw^2 n \leq cw^2 n^2$, já que todas as coleções geradas obedecendo este limite de tamanho serão triviais, nosso caso base. Vamos supor verdadeiro que

$$T(w_1) \leq cw_1^2 n_1^2$$
$$T(w_2) \leq cw_2^2 n_2^2$$

Para mostrar que $T(w) = O(w^2 n^2)$, vamos aplicar a hipótese de indução e usar as seguintes relações:

1. $w_1, w_2 > 0$.
2. $w_1 + w_2 \leq w + 1 \Longrightarrow w_1 \leq w$ e $w_2 \leq w$.
3. $n_1^2 + n_2^2 = (n_1 + n_2)^2 - 2n_1 n_2 = (n+1)^2 - 2n_1 n_2 = n^2 + 2n + 1 - 2n_1 n_2$

$$\begin{aligned}
T(w) &= T(w_1) + T(w_2) + D(w) \\
&= cw_1^2 n_1^2 + cw_2^2 n_2^2 + cw^2 n \\
&= c(w_1^2 n_1^2 + w_2^2 n_2^2) + cw^2 n \\
&\leq c(w^2 n_1^2 + w^2 n_2^2) + cw^2 n \\
&= cw^2(n_1^2 + n_2^2) + cw^2 n \\
&= cw^2(n^2 + 2n + 1 - 2n_1 n_2) + cw^2 n \\
&= cw^2 n^2 + 2cw^2 n + cw^2 - 2cw^2 n_1 n_2 + cw^2 n \\
&\leq cw^2 n^2 + 3cw^2 n + cw^2 - 2cw^2(n(n+2)) \\
&= cw^2 n^2 + 3cw^2 n + cw^2 - 2cw^2(n^2 + 2n) \\
&= cw^2 n^2 + 3cw^2 n + cw^2 - 2cw^2 n^2 - 4cw^2 n \\
&\leq cw^2 n^2
\end{aligned}$$

Então $T(w) = O(w^2 n^2) = O((n + m + r)^2 n^2)$, o que devíamos demonstrar. □

Exemplo 84 *Vamos construir uma árvore PQR com o algoritmo de Meidanis e Munuera para*

$$U = abcdefghi$$
$$\mathcal{C} = \{ab, bc, ac, gh, def, ghi, abcd, efghi\}.$$

O conjunto $H = abc$ é a união não trivial de uma das componentes do grafo $G(\mathcal{C})$, que aparece na Figura 19 (a). Decompondo, obtemos então as seguintes subinstâncias:

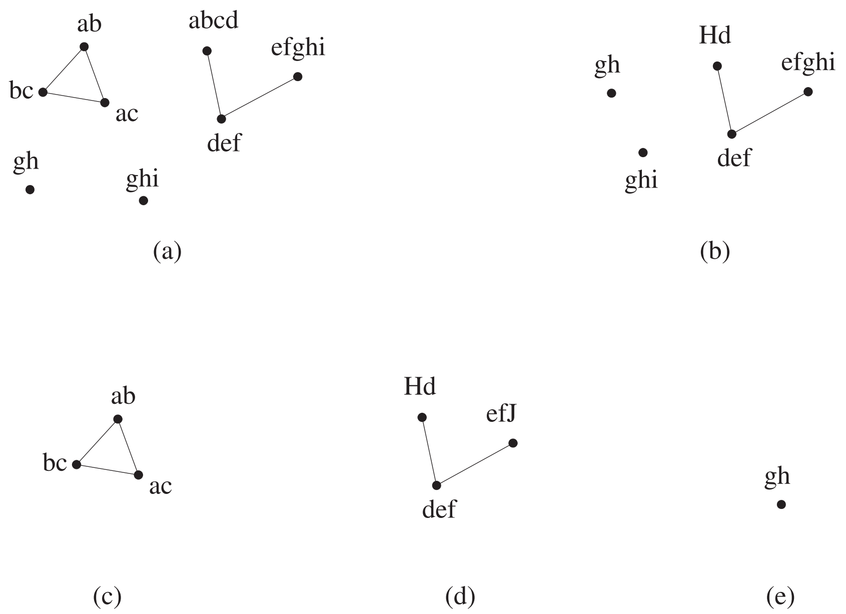

Figura 19: Grafos de sobreposição estrita das instâncias (a) $(U,\mathcal{C})$, (b) $(U_1,\mathcal{C}_1)$, (c) $(U_2,\mathcal{C}_2)$, (d) $(U_{11},\mathcal{C}_{11})$ e (e)$(U_{12},\mathcal{C}_{12})$.

$$(U,\mathcal{C})/H = \left\{ \begin{array}{l} U_1 = Hdefghi \\ \mathcal{C}_1 = \{gh, def, ghi, Hd, efghi\} \end{array} \right. \qquad (U,\mathcal{C}) \wedge H = \left\{ \begin{array}{l} U_2 = abc \\ \mathcal{C}_2 = \{ab, bc, ac\} \end{array} \right.$$

Podemos observar que todas as uniões de componentes do grafo $G(\mathcal{C}_2)$, para a instância $(U,\mathcal{C}) \wedge H$ (Figura 19 (c)) são triviais e a partição mais grossa de U_2 em relação a $\mathcal{C}_2$ tem apenas blocos triviais, o que permite concluir que a instância é prima. A árvore T_2 aparece na Figura 20 (a).

Para a instância $(U,\mathcal{C})/H$, obtemos a união não trivial de componente $J = ghi$ do

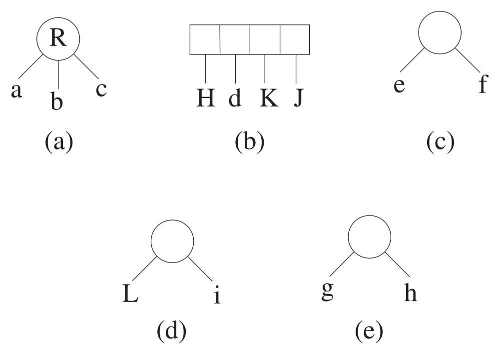

Figura 20: Árvores PQR (a) T_2, (b) T_{111}, (c) T_{112}, (d) T_{121} e (e) T_{122}.

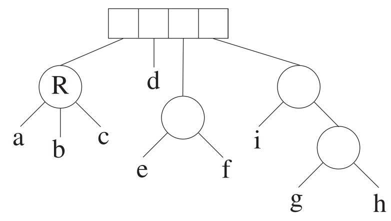

Figura 21: Árvore PQR para $(U,\mathcal{C})$.

grafo $G(\mathcal{C}_1)$ (Figura 19 (b)). Obtemos então as subinstâncias:

$$(U_1,\mathcal{C}_1)/J = \begin{cases} U_{11} = HdefJ \\ \mathcal{C}_{11} = \{def, Hd, efJ\} \end{cases} \qquad (U_1,\mathcal{C}_1) \wedge J = \begin{cases} U_{12} = ghi \\ \mathcal{C}_{12} = \{gh\} \end{cases}$$

A instância $(U_1,\mathcal{C}_1)/J$ ainda pode ser decomposta, pois $K = ef$ é um bloco da partição mais grossa de U_{11} em relação a $\mathcal{C}_{11}$, que é $\{H, d, ef, J\}$ (O grafo $G(\mathcal{C}_{11})$ tem todas as uniões de componentes triviais, como podemos ver na Figura 19 (d).) Decompondo $(U_{11},\mathcal{C}_{11})$, obtemos,

$$(U_{11},\mathcal{C}_{11})/K = \begin{cases} U_{111} = HdKJ \\ \mathcal{C}_{111} = \{dK, Hd, KJ\} \end{cases} \qquad (U_{11},\mathcal{C}_{11}) \wedge K = \begin{cases} U_{112} = ef \\ \mathcal{C}_{112} = \{\} \end{cases}$$

Estas duas instâncias são primas e as árvores T_{111} e T_{112} aparecem nas Figuras 20 (b) e (c), respectivamente.

No caso da instância $(U_1,\mathcal{C}_1) \wedge J$, $L = gh$ é união não trivial de componente do grafo $G(\mathcal{C}_{12})$, na Figura 19 (e). Decompondo $(U_{12},\mathcal{C}_{12}) \wedge L$, obtemos,

$$(U_{12},\mathcal{C}_{12})/L = \begin{cases} U_{121} = Li \\ \mathcal{C}_{121} = \{L\} \end{cases} \qquad (U_{12},\mathcal{C}_{12}) \wedge L = \begin{cases} U_{122} = gh \\ \mathcal{C}_{122} = \{\} \end{cases}$$

Todas essas instâncias são primas e as árvores T_{121} e T_{122} aparecem respectivamente nas Figuras 20 (d) e (e).

Recompondo as árvores, obtemos a árvore PQR na Figura 21.

4.2 Algoritmo Não Recursivo

Ao construir um algoritmo não recursivo, queremos evitar, além da própria recursão, a construção do grafo de sobreposições estritas e a obtenção de todas as suas uniões a cada recursão. A idéia é encontrar todos os nós internos de uma única vez e então construir a árvore. Vamos considerar então três tarefas distintas a serem desempenhadas pelo algoritmo. São elas:

1. Encontrar os nós internos da árvore.
2. Identificar o tipo dos nós e, para os nós Q, encontrar a ordem de seus filhos.
3. Construir a árvore.

A partir de agora vamos analisar cada uma das tarefas e considerar seu custo.

Encontrar os nós internos da árvore

O trabalho de Meidanis e Munuera [MM96] apresenta dois métodos que são suficientes para encontrar os nós internos de uma árvore PQR: uniões de componentes do grafo de sobreposições estritas de $\mathcal{C}$ e classes de equivalência da relação de gêmeos de $\mathcal{C}$ desde que todas as uniões de $G(\mathcal{C})$ sejam triviais. Estes dois métodos, no entanto, não permitem que encontremos todos os conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$ de uma única vez, como mostra o Exemplo 85.

Exemplo 85 *Seja $U = abcdefghijk$ e $\mathcal{C} = \{abc, cdefg, ghi\}$. O grafo $G(\mathcal{C})$ tem uma única componente cuja união é $abcdefghi$, não trivial, pois j e k não aparecem em nenhum conjunto de $\mathcal{C}$. As classes de gêmeos são $\{ab, c, def, g, hi, jk\}$. Como as uniões de componentes de $G(\mathcal{C})$ não são triviais, não podemos tomar as classes como conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$, pois $jk \notin \overline{\mathcal{C}}$, mas def e hi são fronteiras de subárvores da árvore PQR para $(U, \mathcal{C})$.*

Por conta dessa limitação, vamos modificar essa estratégia ligeiramente. Ao invés de encontrar os blocos da partição mais grossa para $\mathcal{C}$ (lembremo-nos da igualdade com as classes de equivalência da relação de gêmeos), vamos encontrá-los para cada componente de $G(\mathcal{C})$. Encararemos então os conjuntos de uma componente conexa X de $G(\mathcal{C})$ como uma coleção $\mathcal{X}$ e encontraremos os blocos da partição mais grossa de $\bigcup_{A \in \mathcal{X}} A$ em relação a $\mathcal{X}$. O Lema a seguir mostra que essa nova estratégia é suficiente.

Lema 86 *Os conjuntos formados a partir de*

- *união de componentes conexas de $G(\mathcal{C})$ e*
- *blocos da partição mais grossa de cada uma das componentes conexas de $G(\mathcal{C})$*

são precisamente os conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$.

Prova: Vamos supor que H é a união não disjunta dos conjuntos que formam uma componente X de $G(\mathcal{C})$. Então $H \in \overline{\mathcal{C}}$. Seja $\mathcal{X}$ a coleção formada pelos conjuntos de X. Certamente H contém todos os conjuntos de X e então $H \perp \mathcal{X}$. Como nenhum conjunto $A \in X$ se sobrepõe a algum conjunto B em uma componente $Y \neq X$ de $G(\mathcal{C})$, $H \in \mathcal{C}^\perp$. Suponhamos agora que H é um bloco da partição mais grossa de $\bigcup_{A \in \mathcal{X}} A$ em relação a $\mathcal{X}$. Como H não contém nenhum conjunto de X propriamente e os conjuntos em X e aqueles em todas as outras componentes de $G(\mathcal{C})$ diferentes de X são ortogonais, $H \in \mathcal{C}^\perp$. Pelos Lemas 73 e 76, $H \in \overline{\mathcal{X}} \cap \mathcal{X}^\perp$. Pelo Teorema 24, $\mathcal{X} \subseteq \mathcal{C} \Longrightarrow \overline{\mathcal{X}} \subseteq \overline{\mathcal{C}}$ e então $H \in \overline{\mathcal{C}}$.

Por outro lado, seja $H \in \overline{\mathcal{C}} \cap \mathcal{C}^\perp$. Se $H \in \mathcal{C}$, H é uma união de componente de $G(\mathcal{C})$, pois nenhum $A \in \mathcal{C}$ se sobrepõe estritamente a H. Se $H \notin \mathcal{C}$, suponhamos então que H não é união de componente de $G(\mathcal{C})$. Como $H \in \overline{\mathcal{C}}$, pelo menos um conjunto $A \in \mathcal{C}$ contém H propriamente, pois H não pode ter sido obtido pela união não disjunta de conjuntos contidos propriamente em H, porque nesse caso H não pertenceria a $\mathcal{C}^\perp$ ou seria união de componente de $G(\mathcal{C})$. Então H é ou a interseção ou a diferença não contida de dois conjuntos em $\overline{\mathcal{C}}$. Também sabemos que nenhum conjunto contido propriamente em H está na mesma componente conexa que um que contém H. Podemos concluir então que H é um bloco da partição mais grossa de $\bigcup_{A \in \mathcal{X}} A$, pois nenhum conjunto de $\mathcal{C}$ refina H e nenhum elemento $a \notin H$ aparece no mesmo bloco que H, porque $H \in \overline{\mathcal{X}}$. □

Lema 87 *O custo para encontrar todos os conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^\perp$ é $O((n+m+r)m)$.*

Prova: Os conjuntos em cada componente, bem como suas uniões, podem ser obtidos pelo algoritmo da Figura 16 a custo $O((n+m+r)m)$. Para construir todas as partições mais grossas, podemos usar o algoritmo da Figura 17, que vai encontrá-las em tempo $O(n+m+r)$. O algoritmo será aplicado no máximo n vezes, marcando os r elementos dos m conjuntos. O custo total será, então, $O((n+m+r)m)$. □

Identificar o tipo dos nós

Nossa primeira observação é que se um nó é uma união de componente de $G(\mathcal{C})$ formada por apenas um conjunto, este nó é P. Caso contrário o nó será ou Q ou R, e para decidir entre um e outro, o algoritmo da Figura 17 foi usado para verificar se a componente possui a P1C. Se uma componente X possui a propriedade, o tipo do nó que é sua união é Q e caso contrário, R. No caso em que o nó é do tipo Q, aquele algoritmo nos dá ainda uma permutação dos blocos dessa união, e, a partir desta permutação poderemos derivar uma ordem para os filhos deste nó, com cuidado adicional, pois nesse caso, como a coleção não é necessariamente prima, os blocos construídos pelo algoritmo podem não ser unitários. Finalmente observamos que os conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^\perp$ obtidos como blocos da partição mais grossa de alguma das componentes é um nó P. Os resultados a seguir confirmam nossas observações.

Lema 88 *Os nós que são formados como uma componente conexa de $G(\mathcal{C})$ com apenas um conjunto de $\mathcal{C}$ são do tipo P.*

Prova: Seja um conjunto $H \in \mathcal{C}$ e X a componente conexa de $G(\mathcal{C})$ formada apenas por H. O conjunto H é um nó interno da árvore sendo construída. Suponha que H é um nó R. Nesse caso, essa componente não tem a P1C, mas isso não é possível, já que uma componente com apenas um conjunto possui sempre a propriedade. Resta a possibilidade dessa componente ser um nó Q, mas isto também não é possível, já que qualquer permutação de H deixa os conjuntos de X consecutivos.

Por outro lado, se X é uma componente formada por dois ou mais conjuntos que se sobrepõem estritamente, não mais todas as permutações de sua união, H, tem todos os conjuntos em X consecutivos, o que impede que H seja do tipo P. □

Lema 89 *Os nós que são obtidos como blocos da partição mais grossa de cada uma das componentes de $G(\mathcal{C})$ são do tipo P.*

Prova: Suponhamos, para efeito de contradição que um nó H obtido como bloco da partição mais grossa é do tipo Q ou R. Em qualquer dos casos deve haver em $\mathcal{C}$ pelo menos um conjunto que se sobrepõe estritamente a H, uma contradição, pois $H \in \overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$. □

Observamos finalmente que certos nós podem ser obtidos tanto como uniões de componentes quanto como blocos da partição mais grossa. Nesse caso, os tipos Q e R devem ter precedência sobre o tipo P. Podemos observar que nunca ocorrem conflitos entre as classificações Q e R, porque um mesmo nó não pode ser união de duas componentes distintas, cada uma delas com mais de um conjunto.

Pelo que foi mostrado na descrição do passo anterior, uma simples marcação vai permitir que identifiquemos o tipo dos nós ao mesmo tempo que os encontramos.

Construir a árvore

Depois de obter os nós internos, o algoritmo para a filogenia binária de Meidanis e Munuera [MM95] fornece uma maneira simples, de custo $O(n + m + r)$, para construir a árvore PQR. Vamos fazer algumas definições e mostrar o relacionamento entre uma árvore filogenética e uma árvore PQR.

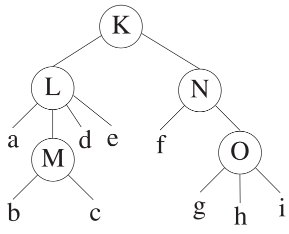

Figura 22: Exemplo de árvore filogenética.

Definição 90 *Dizemos que uma matriz binária M admite uma filogenia quando é possível construir uma árvore enraizada T tal que as linhas de M estejam em correspondência um para um com as folhas de T e as colunas de M estejam em correspondência um para um com os nós internos de T. Além disso, para toda coluna A de M, as linhas que têm o um na interseção com A devem corresponder exatamente às folhas da subárvore enraizada no nó correspondente a A.*

Pela definição, podemos perceber que esse modelo é adequado à árvore PQR, já que os conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$ são exatamente as fronteiras de todas as subárvores de uma árvore PQR. Para que uma árvore filogenética possa ser construída, as restrições impostas pelo teorema abaixo [Gus91] devem ser satisfeitas.

Teorema 91 *Uma matriz binária M tem uma árvore filogenética se e somente se para cada par de colunas A e B de M, ou $A \subseteq B$, ou $B \subseteq A$ ou então $A \cap B = \emptyset$.*

Exemplo 92 *Para a matriz abaixo podemos construir uma árvore filogenética, pois suas colunas respeitam as restrições. A árvore aparece na Figura 22.*

$$
\begin{array}{c}
\\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i
\end{array}
\begin{array}{c}
\begin{array}{ccccc} K & L & M & N & O \end{array} \\
\begin{pmatrix}
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1
\end{pmatrix}
\end{array}
$$

O próximo resultado mostra que os nós calculados nos passos anteriores respeitam a restrição, e, como conseqüência, sempre poderemos construir a árvore PQR usando o algoritmo para a árvore filogenética.

Lema 93 *Para todo par A, B de nós obtidos pelos passos anteriores de nosso algoritmo, ou $A \subseteq B$ ou $B \subseteq A$ ou então $A \cap B = \emptyset$.*

Prova: Sejam A e B conjuntos em $\overline{\mathcal{C}} \cap \mathcal{C}^{\perp}$. Então $A \in \overline{\mathcal{C}}$ e $B \in \mathcal{C}^{\perp}$. Pelo Teorema 32, $\mathcal{C}^{\perp} = (\overline{\mathcal{C}})^{\perp}$ e então $B \in (\overline{\mathcal{C}})^{\perp}$. Podemos concluir então que A e B satisfazem às restrições enunciadas. □

O algoritmo de Meidanis e Munuera aceita como entrada um grafo bipartido cujos vértices são divididos em dois conjuntos X e Y, disjuntos. O conjunto X tem um vértice para cada elemento de U e Y tem um vértice para cada conjunto de $\mathcal{C}$. Há uma aresta entre dois vértices $x \in X$ e $y \in Y$ se e somente se o elemento x pertence ao conjunto y. Esta é uma outra representação possível para uma instância $(U, \mathcal{C})$. Podemos construir o grafo diretamente, a partir da coleção $\mathcal{C}$, em tempo $O(n+m+r)$, processando os conjuntos um a um.

Uma pequena modificação no algoritmo permite que as informações relativas ao tipo do nó sejam consideradas durante a construção da árvore sem afetar sua complexidade. Basta que adicionemos a cada vértice em Y informação sobre o tipo do nó e sobre a ordem dos filhos de nós Q. A ordem dos filhos de nós Q pode ser determinada percorrendo a lista de conjuntos retornada pelo algoritmo para testar a P1C. Tal informação pode ser facilmente transportada para a árvore, à medida que seus nós são adicionados, a um custo $O(mn)$.

Podemos perceber que, ao final dos três passos, teremos construído uma árvore PQR para um par $(U, \mathcal{C})$ fazendo $O((n + m + r)m)$ operações. A correção da construção é garantida pelos resultados apresentados aqui e no trabalho de Meidanis e Munuera [MM95].

Exemplo 94 *Vamos construir uma árvore PQR com o algoritmo não recursivo para*

$$U = abcdefghi$$
$$\mathcal{C} = \{ab, bc, ac, gh, def, ghi, abcd, efghi\}.$$

O grafo $G(\mathcal{C})$ aparece na Figura 19(a). Suas uniões de componentes e respectivas coleções são:

$$H_1 = abc \qquad H_2 = gh$$
$$\mathcal{X}_1 = \{ab, bc, ac\} \qquad \mathcal{X}_2 = \{gh\}$$
$$PMG(H_1, \mathcal{X}_1) = \{a, b, c\}$$

$$H_3 = ghi \qquad H_4 = abcdefghi$$
$$\mathcal{X}_3 = \{ghi\} \qquad \mathcal{X}_4 = \{abcd, def, efghi\}$$
$$PMG(H_4, \mathcal{X}_4) = \{abc, d, ef, ghi\}$$

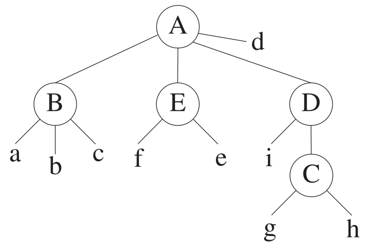

Figura 23: Árvore filogenética construída com os nós da árvore PQR para $\mathcal{C}$.

O par $(H_1, \mathcal{X}_1)$ não possui a P1C, então abc é o domínio de um nó R. O par $(H_4, \mathcal{X}_4)$ possui a P1C e então abcdefghi é o domínio de um nó Q com seus filhos na ordem $\alpha = abc \cdot d \cdot ef \cdot ghi$. Os conjuntos gh, ghi e ef são domínios de nós do tipo P. Construindo o grafo e aplicando o algoritmo de Meidanis e Munuera obtemos a árvore filogenética na Figura 23. Transportando a informação sobre os tipos dos nós e a ordem dos filhos de nós Q, obtemos a árvore PQR na Figura 21.

4.3 O Problema Incremental

Alguns algoritmos, como o de Ferreira e Song [FS92], utilizam bactracking sobre as árvores PQ para eliminar conjuntos que poderiam torná-la nula. Ghosh [Gho72] caracteriza o problema da recuperação consecutiva, que poderia beneficiar-se da existência de um algoritmo de tempo real para as árvores PQ. O resultado que segue é uma consideração sobre o custo desta abordagem.

Dadas uma coleção $\mathcal{C}$, uma árvore PQR para esta coleção e um conjunto A, ambos sobre um conjunto universo $U = a_1 a_2 \ldots a_n$, queremos saber se é possível encontrar uma árvore PQR para $(\mathcal{C} \cup A)$ em tempo $O(|A|)$. Infelizmente, isso não é sempre possível. Para construir um contra-exemplo, suponhamos a família de coleções $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ definida como se segue:

$$
\begin{aligned}
&\mathcal{C}_1 = \{a_1\} \\
&\mathcal{C}_2 = \mathcal{C}_1 \cup \{a_1 a_2\} \\
&\mathcal{C}_3 = \mathcal{C}_2 \cup \{a_1 a_2 a_3\} \\
&\vdots \\
&\mathcal{C}_n = \mathcal{C}_{n-1} \cup \{a_1 a_2 \ldots a_n\}
\end{aligned}
$$

Associamos uma árvore PQR a cada coleção $\mathcal{C}_i$, das quais podemos ver alguns exemplos na Figura 24.

Suponhamos agora que tomamos a árvore PQR T para a coleção $\mathcal{C}_n$ e a $\mathcal{C}_n$ adicionamos o conjunto $a_1 a_n$, i.e., o conjunto formado pelas folhas de menor e maior altura na

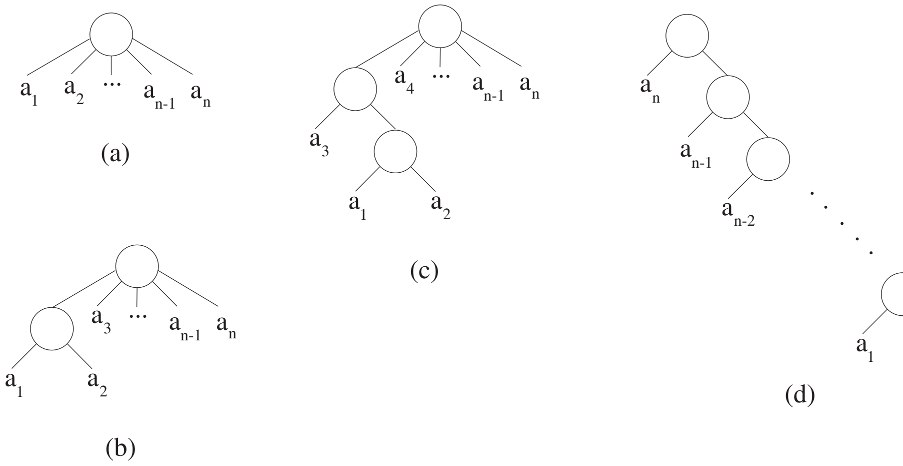

Figura 24: Árvores geradas a partir da família de coleções, (a) $\mathcal{C}_1$, (b) $\mathcal{C}_2$, (c) $\mathcal{C}_3$ e (d) $\mathcal{C}_n$.

árvore. Teremos que ajustar a árvore de forma que não haja nenhum elemento entre a_1 e a_n em qualquer das árvores equivalentes a T.

Mas os conjuntos que já faziam parte de $\mathcal{C}_n$ impunham as seguintes restrições:

a_1 e a_2 devem ser consecutivos
a_1, a_2 e a_3 devem ser consecutivos
$\vdots$
$a_1, a_2, \ldots, a_n$ devem ser consecutivos

Desta forma, a_n, a_1 e a_2 devem ser filhos de um mesmo nó Q pois não podem ser permutados. Logo, o elemento a_3 deve ser colocado como vizinho de a_2, pois se for colocado como vizinho de a_n, a_1, a_2 e a_3 deixarão de ser consecutivos. Indutivamente podemos perceber que todo elemento a_i, $i < n$ considerado nesse ajuste deverá ser posicionado somente como vizinho do $(i-1)$-ésimo elemento, sob pena de contrariar as restrições impostas pela coleção. Então a árvore será reduzida a uma outra com um único nó Q como raiz, como na Figura 25.

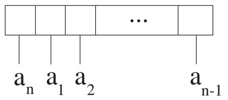

Figura 25: Árvore reduzida pelas restrições à coleção $\mathcal{C}_n$.

Este ajuste nos fez reduzir uma árvore com n nós a uma árvore com apenas 1 nó interno, com a inclusão de um conjunto de cardinalidade 2. Concluímos então que uma solução para o problema incremental não será $O(|A|)$, onde A é o conjunto incluído.

Capítulo 5

Problemas e Aplicações

Neste capítulo propomos um modelo para representar as árvores PQR e resolvemos os problemas apresentados no Capítulo 2 com base nessas árvores e na teoria desenvolvida no Capítulo 3. Propomos, ainda, alguns problemas envolvendo as coleções $\overline{\mathcal{C}}$ e $\mathcal{C}^{\perp}$. A correção das soluções apresentadas aqui vêm, em sua maior parte, da correção da teoria apresentada quando da definição das árvore PQR. Analisamos também algumas aplicações das árvores PQ, considerando a possibilidade de usar as árvores PQR.

5.1 Representação das Árvores PQR

No Capítulo 4, quando tratamos de algoritmos para a construção das árvores PQR, não nos preocupamos com a representação da árvore propriamente. Vamos propor, então, um modelo orientado a objetos para a árvore. Um modelo orientado a objetos também foi proposto para as árvores PQ [Lei97]. A Figura 26 apresenta a estrutura de classes proposta, seguindo a OMT [RBP+91]. Os métodos e atributos no modelo foram usados para solucionar os problemas de obter uma permutação válida de U e obter todas as permutações válidas de U.

5.2 Encontrar Uma Permutação Válida

Para obter uma permutação válida de U para $\mathcal{C}$ podemos inicialmente construir uma árvore PQR T para $(U,\mathcal{C})$. Se T não possui nós R, sua fronteira é uma permutação em $\mathit{Valid}(\mathcal{C})$, como demonstramos no Capítulo 3. Se T possui nós R, nenhuma permutação é válida.

Para obter a fronteira de uma árvore PQR, basta que iniciemos uma busca em pré-ordem a partir da raiz da árvore. Esta busca deve percorrer primeiramente os filhos mais

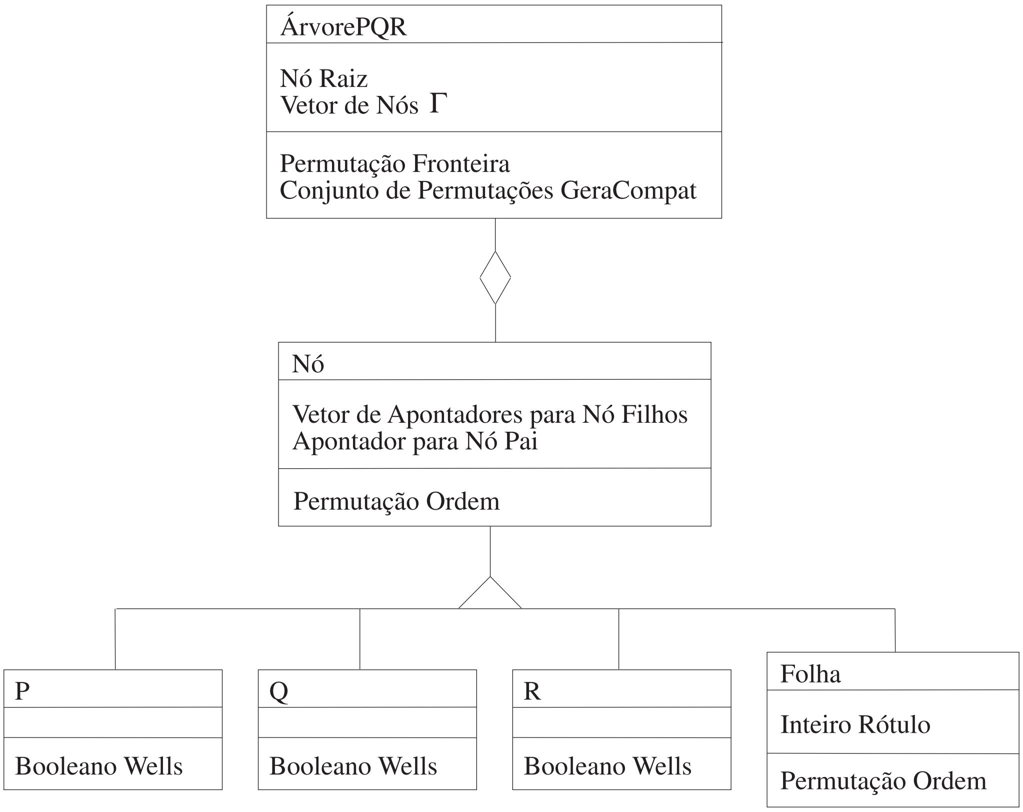

Figura 26: Modelo proposto para a árvore PQR.

à esquerda e cada folha alcançada deve retornar seu rótulo ao nó que enviou a mensagem (ou impimi-lo ou ainda armazená-lo). Os métodos *Fronteira*, que retorna uma permutação de U, na classe *ÁrvorePQR*, e *Ordem* na classe *Nó* (sobrecarregado na classe *Folha*) são usados para obter a fronteira da árvore. O método *Fronteira* faz parte da interface de *ÁrvorePQR* e inicia uma busca a partir da raiz. O método *Ordem* retorna o rótulo de uma folha ou a concatenação de rótulos de subárvores filhas de nós internos. O número de mensagens e operações para executar esse procedimento é limitado pelo número de nós internos da árvore, que por sua vez é limitado pelo número de folhas, e é, portanto, $O(n)$.

5.3 Encontrar Todas as Permutações Válidas

Para obter todas as fronteiras de uma árvore PQR sem nós R, vamos aplicar todas as transformações de equivalência possíveis e a cada uma, gerar a fronteira da árvore. Para isso vamos executar o algoritmo *GeraCompat* (Figura 27) incluído na interface da classe *ÁrvorePQR*.

Para gerar todas as permutações de filhos de um nó P ou R de forma sistemática podemos usar o algoritmo de Wells [Wel61], que gera todas as $l!$ permutações de l índices em tempo $O(l!)$. Para gerar a $(k+1)$-ésima permutação, aquele algoritmo permuta apenas duas posições da k-ésima permutação. Cada uma das classes derivadas de *Nó* sobrecarrega o método *Wells*. Este método é inicializado com a primeira permutação possível dos seus filhos. A cada chamada, este método gera uma nova permutação, retornando verdadeiro quando a permutação original for gerada novamente e falso, caso contrário. No caso dos nós Q, o método *Wells* vai gerar apenas duas permutações, uma a reversa da outra.

A correção do algoritmo é demonstrada no Lema 95 e sua complexidade é analisada no Lema 96.

```
Lista de Permutações Função GeraCompat()
    Conjunto A ← ∅;
    Inteiro i;

    Armazene em Γ apontadores para os l nós internos de T;
    Repita
        A ← A ∪ {Fronteira()};
        i ← 0;
        Enquanto i ≤ l e Γ(i).Wells() faça
            i ← i + 1;
    até i > l;
    Retorne(A);
```

Figura 27: Algoritmo para gerar todas as fronteiras de uma árvore PQR.

Lema 95 *O algoritmo na Figura 27 gera todas as fronteiras de uma árvore PQR T.*

Prova: Vamos usar indução no número l de nós internos de T. Se $l = 1$, as transformações de equivalência de seu único nó vão ser geradas pelo método *Wells* e adicionadas a A. Quando a permutação inicial das folhas for gerada, o método *Wells* retornará verdadeiro e o algoritmo retornará o conjunto de todas as fronteiras de T. Vamos supor que o algoritmo retorne todas as fronteiras de árvores com $l = k - 1$ nós internos, $l \geq 1$. Como o algoritmo pára apenas depois que todas as permutações de k forem geradas e, por hipótese de indução, todas as permutações dos $k - 1$ nós são geradas e mais, são geradas para cada permutação do nó k, podemos concluir que todas as fronteiras de T são adicionadas a A, que é retornado pelo algoritmo. □

Lema 96 *O custo do algoritmo na Figura 27 é $O(|Compat(T)|.n)$.*

Prova: A árvore tem no máximo $n-1$ nós internos. Para cada permutação de um nó, uma fronteira é obtida a custo $O(n)$. Para propagar as mensagens pelos $n-1$ nós, o custo será no máximo n para cada fronteira em $Compat(T)$. Logo o custo total é $O(|Compat(T)|.n)$. $\square$

5.4 Problemas Relacionados às Coleções $\overline{\mathcal{C}}$ e $\mathcal{C}^{\perp}$

Nesta seção nos dedicamos a solucionar algumas questões que surgem após a definição das coleções $\overline{\mathcal{C}}$ e $\mathcal{C}^{\perp}$. Para tanto, vamos supor que, dada a coleção $\mathcal{C}$, construímos uma árvore PQR T para $\mathcal{C}$ tal que $Compl(T) = \overline{\mathcal{C}}$ e vamos nos apoiar nos resultados do Capítulo 3. As questões são as seguintes:

1. Como verificar se um conjunto $A \subseteq U$ pertence a $\overline{\mathcal{C}}$.
2. Como gerar um conjunto não trivial em $\overline{\mathcal{C}}$, se houver.
3. Como gerar todos os conjuntos não triviais em $\overline{\mathcal{C}}$.
4. Como verificar se um conjunto $A \subseteq U$ pertence a $\mathcal{C}^{\perp}$.
5. Como gerar um conjunto não trivial em $\mathcal{C}^{\perp}$, se houver.
6. Como gerar todos os conjuntos não triviais em $\mathcal{C}^{\perp}$.

A solução para cada um desses problemas é a seguinte:

1. Para verificar se um conjunto $A \subseteq U$ pertence a $\overline{\mathcal{C}}$, primeiramente os elementos de A devem ser marcados em T, fazendo uma busca em pré-ordem. Depois, um procedimento similar ao da prova do Teorema 48 deve ser executado, propagando as marcações de baixo para cima. Ao fim do procedimento (lembremo-nos que $Compl(T) = \overline{\mathcal{C}}$), se os nós que permaneceram marcados formam um conjunto de todos os filhos de nó P, um conjunto de filhos consecutivos de nó Q ou um conjunto qualquer de filhos de nó R, $A \in \overline{\mathcal{C}}$. Caso contrário, $A \notin \overline{\mathcal{C}}$.
2. Para gerar um conjunto não trivial em $\overline{\mathcal{C}}$, basta escolher um nó interno qualquer v de T e selecionar um subconjunto de filhos de v que respeite as restrições da Definição 45, de acordo com o tipo de v. Construindo $\bigcup_{x\in A} Desc_T(x)$, obteremos um conjunto não trivial em $\overline{\mathcal{C}}$, a não ser que v seja a raiz da árvore e A contenha todos os seus filhos.

3. Para gerar todos os conjuntos não triviais em $\overline{\mathcal{C}}$, é necessário tomar todos os nós internos v de T e encontrar todos os conjuntos A de filhos de v que respeitem as restrições da Definição 45, de acordo com o tipo de v excetuando o caso em que A é formado por todos os filhos da raiz.

4. Para verificar se um conjunto pertence a $\mathcal{C}^{\perp}$, podemos usar um procedimento análogo ao usado para verificar um conjunto em $\overline{\mathcal{C}}$, lembrando, dessa vez que $\mathcal{C}^{\perp} = Orto(T)$.

5. Para gerar um conjunto não trivial em $\mathcal{C}^{\perp}$, podemos usar um procedimento análogo ao usado para gerar um conjunto não trivial em $\overline{\mathcal{C}}$, neste caso de acordo com a Definição 54.

6. Para gerar todos os conjuntos não triviais em $\mathcal{C}^{\perp}$, podemos usar um procedimento análogo ao usado para gerar todos os conjuntos não triviais em $\overline{\mathcal{C}}$, neste caso de acordo com a Definição 54. Este problema também foi resolvido por Novick [Nov89], usando as árvores gPQ.

As soluções para os Problemas 1, 2, 4 e 5 têm custo $O(n)$. As soluções para os problemas 3 e 6 têm, respectivamente, custo $O(n.|\overline{\mathcal{C}}|)$ e $O(n.|\mathcal{C}^{\perp}|)$.

5.5 Nós R

Em algumas situações, uma subárvore de uma árvore PQR enraizada em um nó R tem como domínio um subconjunto relativamente pequeno de elementos de U, como na Figura 28. Algumas aplicações onde a árvore PQ seria nula nesses casos podem, potencialmente, tirar vantagem dessa informação.

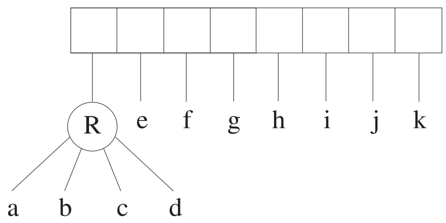

Figura 28: Árvore com nó R que tem o domínio com poucos elementos de U.

Esse parece ser o caso da construção de mapas físicos de DNA, em que a matriz pode conter erros introduzidos pelos experimentos de hibridização. Uma árvore que possua uma subárvore enraizada em um nó R com pequeno domínio pode indicar um subconjunto dos experimentos como condidato a possuir um erro. Pode ser o caso em que os experimentos nesse pequeno grupo devam ser repetidos. Porto [Por97] sugere ainda a possibilidade de relacionar os nós R às matrizes que não são totalmente unimodulares, possibilitando aproximar soluções para problemas de programação linear inteira. O algoritmo de Ferreira e Song [FS92] faz backtracking sobre as árvores PQ sempre que elas se tornam nulas, para maximizar o número de conjuntos processados, usando heurísticas para ordená-los. Pode ser o caso em que os nós R ajudem a fornecer, senão uma solução ótima, pelo menos uma garantia ao algoritmo daqueles autores.

Em outros casos, uma subárvore enraizada em um nó R tem como domínio um subconjunto com muitos elementos de U, como é o caso da árvore na Figura 29.

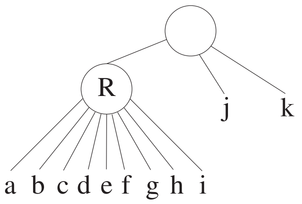

Figura 29: Árvore com nó R que tem o domínio com muitos elementos de U.

Para os mapas físicos parece não haver vantagem explícita em conhecer um conjunto tão grande de experimentos que potencialmente possuem erros, pois apenas um pode causar a transformação de um nó Q em um nó R. Este parece ser o caso também para outras aplicações, isto é, as informações que o nó R fornecem não parecem úteis.

5.6 Aplicações de Árvores PQ

Nesta seção apresentamos algumas aplicações das árvores PQ que não são derivadas diretamente da propriedade dos uns consecutivos. Analisamos brevemente a possibilidade de empregarmos as árvores PQR para essas aplicações.

5.6.1 Teste de planaridade em grafos

Um grafo $G = (V, E)$ é planar se ele puder ser representado no plano sem arestas que se cruzam. Uma das aplicações das árvores PQ é o teste de planaridade de grafos. Booth e Lueker [BL76] propõem um algoritmo que, usando árvores PQ, simula o algoritmo de Lempel e colaboradores [LEC66]. Este último algoritmo considera a vizinhança de cada um dos n vértices do grafo (numerados segundo uma ordem st), representando-as como fórmulas. Permutando, substituindo e reduzindo os fatores dessas fórmulas, o algoritmo conclui que o grafo é planar se uma fórmula puder ser obtida depois de processar $n-1$ vizinhanças.

Construindo uma árvore PQ com as arestas do grafo como folhas e tirando vantagem do algoritmo incremental para construi-la, as reduções sobre as fórmulas são simuladas. A árvore PQ é inicializada com a vizinhança do vértice 1 e, entre cada redução para uma vizinhança, algumas folhas da árvore são substituídas. Se uma árvore PQ puder ser reduzida por todas as vizinhanças, o grafo é planar. Chiba e colaboradores [CNAO85] apresentam um algoritmo para desenhar o grafo no plano que também é baseado no algoritmo de Lempel e colaboradores e usa as árvores PQ.

Para usar árvores PQR, que não são construídas por um algoritmo incremental, haverá a necessidade de construir $n-1$ árvores. Além disso, haverá a necessidade de manipular os conjuntos em $Compl(T)$ para simular as substituições do algoritmo original, entre a construção de uma árvore e outra, de maneira bem menos natural que as substituições entre uma redução e outra das árvores PQ. Por outro lado, nada impede a obtenção de um algoritmo incremental para construir árvores PQR.

5.6.2 Isomorfismo de grafos intervalo

O trabalho de Booth e Lueker [BL79] descreve um método para comparar duas árvores PQ T e T', construídas para grafos intervalo G e G' respectivamente, e verificar se G e G' são isomorfos. Os nós das árvores são rotulados e colocados em uma ordem específica, definida pelos autores. Se T e T' são isomorfas e os rótulos de seus vértices equivalentes são iguais, os resultados apresentados pelos autores garantem que G e G' são isomorfos.

Este teste também pode ser aplicado às árvores PQR, pois estas são isomorfas a árvores PQ se a coleção possui a P1C.

5.6.3 Subgrafo maximal planar

Um grafo $G=(V,E)$ é planar se ele puder ser representado no plano sem arestas que se cruzam. Se G não é planar, um subgrafo maximal planar de G é um subgrafo planar G' tal que toda aresta em $G \setminus G'$ destroi a planaridade se acrescentada a G'.

Apesar do algoritmo de Kant [Kan92], que usa as árvores PQ para encontrar um subgrafo maximal planar (e corrige problemas com algoritmos anteriores), Jünger, Leipert e Mutzel [JLM97] sugerem que as árvores PQ não devem ser usadas para resolver esse problema, pois uma invariante do algoritmo de Lempel e colaboradores [LEC66] não é mantida durante o algoritmo de planarização. Afirmam, ainda, que essa invariante não é mantida nem mesmo em uma versão corrigida do algoritmo, supondo um caso ideal. No mesmo trabalho, Jünger, Leipert e Mutzel mostram que um algoritmo para o reconhecimento de grafos planares nivelados de Heath e Pammaraju [HP95], baseado em árvores PQ não funciona corretamente.

5.6.4 Gerador de módulos

As árvore PQ são usadas por Heeb e Fichtner [HF92] como parte do algoritmo para a construção de circuitos lógicos. Cada módulo de um circuito é um bloco combinacional ou seqüencial que implementa uma função lógica. O problema é encontrar um posicionamento dos módulos que permita que eles sejam alimentados por seus sinais de entrada com um mínimo de linhas adicionais possível e maximizando a ocupação do espaço físico. A entrada para o algoritmo é uma lista de expressões lógicas otimizadas.

Primeiro o algoritmo divide as equações em módulos. Depois, usa uma heurística para posicionar os módulos em linhas, cuidando para que as linhas sejam preenchidas igualmente e que módulos com muitos sinais de entrada comuns sejam colocados juntos. Depois o algoritmo calcula a ordem dos sinais de entrada que permite alimentar todos os módulos, usando as árvores PQ.

Cada módulo é representado por uma árvore PQ. As folhas da árvore são os sinais de entrada, saída e pseudo-entradas do módulo (pseudo-entradas são usadas para representar eventuais falhas nas linhas). Cada linha também é representada por uma árvore PQ, tornando as árvores de cada módulo descendentes de um nó P.

Para encontrar a árvore PQ de todas as linhas e conseqüentemente uma ordem dos sinais de entrada, as árvores são unidas duas a duas, formando uma pirâmide. Na base estão todas as árvores na mesma ordem que as linhas de módulos. No topo, a árvore resultante. Cada nível da pirâmide é obtido unindo duas árvores vizinhas no nível anterior. Unir duas árvores T e T' é obter uma terceira, T'', que tenha os conjuntos em T e os conjuntos em T' consecutivos em sua fronteira. Esta operação é feita reduzindo a fronteira de T em relação aos domínios de subárvores de T'.

Se alguma união falha, isso indica que o algoritmo deverá incluir linhas adicionais. Neste caso, todas as árvores que falharam são agrupadas e um algoritmo de programação dinâmica encontra o menor conjunto de linhas a ser adicionado.

Se usarmos árvores PQR, durante a união das árvores T e T', a árvore T'' pode ser construída a partir da coleção $Can(T) \cup Can(T')$. Esta nova árvore satisfaz às restrições das duas anteriores se não possuir nós R. Se a árvore união possui nós R, as árvores que a geraram devem ser consideradas pelo algoritmo para a adição de linhas. Pode ser o caso, também, que o nó R permita a obtenção de um algoritmo diferente para a adição de linhas.

Capítulo 6

Conclusões

Neste trabalho formalizamos a teoria e apresentamos dois algoritmos para a construção de árvores PQR. Os dois algoritmos evitam a comparação e substituição de padrões de Booth e Lueker para as árvores PQ, de implementação complexa. Nossos algoritmos usam estruturas de dados razoavelmente simples, embora de custo mais alto. Acreditamos, no entanto, que a grande contribuição dessa nova teoria é possibilitar o avanço das investigações a respeito da propriedade dos uns consecutivos, ressaltando que são três, e não apenas duas, as possibilidades para coleções primas.

A complexidade dos algoritmos apresentados é um estimulo à pesquisa de outros mais eficientes, que possibilitarão não só encontrar soluções do problema dos uns consecutivos, mas que poderão fornecer informações interessantes para aproximar a solução de alguns outros, com base nos nós R. Outra possibilidade é pesquisar um algoritmo incremental para as árvores PQR, que processe os conjuntos um a um como o de Booth e Lueker, para que elas possam ser aplicadas a problemas de tempo real.

Finalmente, apontamos a possibilidade de explorar ainda mais a teoria apresentada aqui, obtendo novos e interessantes resultados.

Referências

[BL76] Kellogg S. Booth e George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[BL79] Kellogg S. Booth e George S. Lueker. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM*, 26(2):183–195, 1979.

[BM76] J. A. Bondy e U. S. R. Murty. *Graph theory with applications*. Macmillan Press, Londres, 1976.

[CLR89] Thomas H. Cormen, Charles E. Leiserson, e Ronald L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, 1989.

[CNAO85] Norishige Chiba, Takao Nishizeki, Shigenobu Abe, e Takao Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30:54–76, 1985.

[CY91] Lin Chen e Yaacov Yesha. Parallel recognition of the consecutive ones property with applications. *Journal of Algorithms*, 12:375–392, 1991.

[Esw75] Kapali P. Eswaran. Faithful representation of a family of sets by a set of intervals. *SIAM Journal On Computing*, 4(1):56–68, 1975.

[FG65] D. R. Fulkerson e O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[FS92] Afonso G. Ferreira e Siang W. Song. Achieving optimality for gate matrix layout and PLA folding: a graph theoretic approach. In I. Simon, editor, *Latin'92*, volume 583 de *LNCS*, páginas 139–153, São Paulo, Brasil, 1992. Springer-Verlag.

[Gho72] Sakti P. Ghosh. File organization: the consecutive retrieval property. *Communications of the ACM*, 15(9):802–808, 1972.

[GI95] David S. Greenberg e Sorin Istrail. Physical mapping by STS hybridization: algorithmic strategies and the challenge of software evaluation. *Journal of Computational Biology*, 2(2):219–274, Summer 1995.

[Gus91] Dan Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

[HF92] Hansruedi Heeb e Wolfgang Fichtner. A module generator based on the PQ-tree algorithm. *IEEE Transactions on Computer-Aided Design*, 11(7):876–884, 1992.

[HP95] L. Heath e S. Pemmaraju. Recognizing leveled-planar dags in linear time. In F. J. Brandenburg, editor, *Graph Drawing'95*, volume 1027 de *LNCS*, páginas 300–311. Springer-Verlag, 1995.

[Hsu92] Wen-Lian Hsu. A simple test for the consecutive ones property. In T. Ibaraki, Y. Inagaki, K. Iwama, et al., editores, *ISAAC'92*, volume 650 de *LNCS*, páginas 459–468, Nagoya, Japão, 1992. Springer-Verlag.

[Hsu97] Wen-Lian Hsu. On physical mapping algorithms: an error-tolerant test for the consecutive ones property. Disponível para download em 15 de outubro de 1997; http://www.iis.sinica.edu.tw/LPDA/member/hsu/eindex.html, 1997.

[JLM97] Michael Jünger, Sebastian Leipert, e Petra Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. Relatório Técnico 97.278, Universidade de Köln – Instituto de Informática, Alemanha, 1997.

[Kan92] Goos Kant. An $O(n^2)$ maximal planarization algorithm based on PQ-trees. Relatório Técnico RUU-CS-92-03, Universidade de Utrecht – Departmento de Ciência da Computação, Holanda, 1992.

[Ken69] D. G. Kendall. Incidence matrices, interval graphs and seriation in archaeology. *Pacific Journal of Mathematics*, 28(3):565–570, 1969.

[LEC66] A. Lempel, S. Even, e I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs International Symposium*, páginas 215–232, Paris, 1966. Gordon and Brench.

[Lei97] Sebastian Leipert. PQ-trees, an implementation as template class in C++. Relatório Técnico 97.259, Universidade de Köln – Instituto de Informática, Alemanha, 1997.

[MM95] João Meidanis e Erasmo G. Munuera. A simple linear time algorithm for binary phylogeny. In N. Ziviani, J. Piquer, B. Ribeiro, e R. Baeza-Yates, editores, *Proc. of the XV Intern. Conf. of the Chilean Computer Science Society*, páginas 275–283, Arica, Chile, 1995.

[MM96] João Meidanis e Erasmo G. Munuera. A theory for the consecutive ones property. In N. Ziviani, R. Baeza-Yates, e K. Guimarães, editores, *Proc. of the III South American Workshop on String Processing*, páginas 194–202, Recife, Brasil, 1996.

[MS97] João Meidanis e João C. Setubal. *Introduction to computational molecular biology*. PWS Publishing Co., 1997.

[Mun96] Erasmo G. Munuera. Propriedade dos uns consecutivos e reconhecimento de grafos intervalo. Mestrado em Ciência da Computação, Unicamp, Brasil, 1996.

[Nov89] Mark B. Novick. Generalized PQ-trees. Relatório Técnico 89-1074, Universidade de Cornell – Departmento de Ciência da Computação, 1989.

[OFD85] L. Oubiña e C. Martinez Favini-Dubost. Formules sur un unsemble et hypergraphes d'intervalles. In A. Batbedat, editor, *Cahiers de DEA*, volume S, capítulo 14, páginas 63–84. Universidade de Montpellier, França, 1985.

[Por97] Oscar Porto. Recognizing interval matrices through PQR trees. Apresentado no *ISPM'97 – International Symposium on Mathematical Programming*, Lousanne, Suíça, Agosto 1997.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, et al. *Object-oriented modeling and design*. Prentice-Hall, New Jersey, 1991.

[Rys69] H. J. Ryser. Combinatorial configurations. *SIAM Journal on Applied Mathematics*, 17(3):593–602, 1969.

[Tuc71] Alan Tucker. Matrix characterizations of circular-arc graphs. *Pacific Journal of Mathematics*, 39(2):535–545, 1971.

[Tuc72] Alan Tucker. A structure theorem for the consecutive 1's property. *Journal of Combinatorial Theory*, 12(B):153–162, 1972.

[Wel61] Mark B. Wells. Generation of permutations by transposition. *Mathematics of Computation*, 15:192–195, 1961.