kMST é FPT: color-coding e hashing perfeito

Paulo Feofiloff

IME–USP, 7/12/2006

Resumo

Dado um número inteiro positivo k e um grafo G com pesos não-negativos nas arestas, encontrar uma sub-árvore de G de peso mínimo dentre as que têm k vértices. Esse é o problema kMST. É claro que o problema pode ser resolvido em tempo $O(n^k)$, sendo n o número de vértices de G. Betzler, baseada no trabalho de Scott, Ideker, Karp, Sharan e no de Alon, Yuster e Zwick, mostrou que existe um algoritmo para o kMST que consome tempo $2^{O(k)}\, n^2 \lg n$. A existência de um tal algoritmo prova que o problema kMST pertence à classe de complexidade *FPT* definida por Downey e Fellows. O algoritmo de Betzler envolve a combinação de três idéias muito diferentes: color-coding, geração de árvores e hashing perfeito. O presente texto descreve um algoritmo mais simples, que envolve apenas color-coding e hashing perfeito. O algoritmo consome tempo $O(2^{13k} n^3)$.

1 Introdução

O seguinte problema surge naturalmente em várias aplicações da otimização combinatória, em particular nas aplicações à biologia computacional:

> **Problema** kMST (G, w, k): Dado um grafo G, uma função w que associa um peso não-negativo a cada aresta de G e um número inteiro positivo k, encontrar uma sub-árvore H de G que minimize $w(H)$ sob a restrição $|V_H| = k$.

Sabe-se [11] que o problema kMST *NP*-difícil. Existe um algoritmo de aproximação [8, 2] para o problema (a razão de aproximação é $2+\varepsilon$), mas estamos interessados aqui apenas em algoritmos exatos (e portanto exponenciais).

O problema tem dois parâmetros: k e $n := |V_G|$. Quando $k = n$, o problema kMST se reduz ao problema da árvore geradora de peso mínimo, que pode ser resolvido em tempo $O(n^2)$. Isso sugere imediatamente o seguinte algoritmo "força bruta" para o problema kMST:

Algoritmo FORÇA-BRUTA: Para cada K em $\mathsf{C}_k^{V_G}$, calcule uma árvore geradora de peso mínimo no subgrafo de G induzido por K. Dentre todas essas árvores, escolha uma de peso mínimo.

(A expressão C_k^V denota o conjunto de todos os subconjuntos K de V tais que $|K| = k$.) O algoritmo examina C_k^n subconjuntos de V_G. Como $\mathsf{C}_k^n\, k^2 \leq (n^k/k!)\, k^2 \leq 2n^k$, o algoritmo consome

$$O(n^k)$$

unidades de tempo.

Estamos particularmente interessados no caso em que k é constante.[1] Nesse caso, o algoritmo FORÇA-BRUTA é polinomial, mas extremamente lento para valores elevados de k. Na prática, o algoritmo é tão lento que não permite resolver o problema nem mesmo para valores modestos de n e k.

Seria interessante dispor de um algoritmo em que os efeitos de n e k sobre o consumo de tempo estivessem separados, ou seja, um algoritmo em que "n^k" fosse substituído por algo como "$2^k\, n^3$". (Para $k = 10$, por exemplo, um tal algoritmo teria "$2^{10} n^3$" no lugar de "n^{10}". Para $k = \lg n$, um tal algoritmo teria "n^4" no lugar de "$n^{\lg n}$".) É claro que algo como "$2^{4k} n^5$" também seria um substituto aceitável para "n^k". Problemas que admitem algoritmos desse tipo são conhecidos como *FPT* (= *fixed-parameter tractable*) [6, 9]. Gostaríamos, portanto, de ter uma resposta para a seguinte pergunta:

O problema kMST é *FPT*?

Recentemente, Betzler [4] e Scott, Ideker, Karp, Sharan [13] deram uma resposta afirmativa a essa pergunta. A resposta é conseqüência de um célebre artigo de Alon, Yuster e Zwick [1][3, p.551] publicado há 10 anos. O algoritmo combina três ingredientes muito diferentes e interessantes: color-coding, geração de árvores e hashing perfeito. A complexidade do algoritmo envolve um fator da forma 2^{13k}, ou algo semelhante. Isso pode colocar em dúvida o valor prático do algoritmo. Mas Scott *et al.* dão conta da aplicação bem sucedida de algoritmos semelhantes a um problema em biologia computacional.

O presente artigo descreve um algoritmo para o problema kMST que é um pouco mais simples que o de Betzler, um algoritmo que envolve apenas

- color-coding e
- hashing perfeito.

[1] Ou seja, estamos interessados em famílias infinitas de instâncias do problema, todas com um mesmo valor de k.

2 Esboço do algoritmo FPT para o kMST

Nosso algoritmo para o problema kMST é uma versão simplificada do algoritmo Betzler [4] e Scott–Ideker–Karp–Sharan [13].[2] O primeiro passo do algoritmo usa uma técnica introduzida por Alon–Yuster–Zwick [1][3, p.551] para reduzir o kMST a uma "versão colorida" do problema. Antes de enunciar essa versão, é preciso definir os conceitos de coloração de vértices e subgrafo multicolorido.

Uma k-**coloração** de V_G é qualquer função que leva V_G em $\{1, \ldots, k\}$. (A coloração ignora as arestas do grafo.) Dada uma k-coloração de V_G, diremos que um subgrafo H de G é **multicolorido** se vértices distintos de H tiverem cores distintas.

> **Problema** COLORED-kMST (G, w, k, c): Dado um grafo G, uma função-peso $w \geq 0$ sobre as arestas de G, e uma k-coloração c de V_G, encontrar uma subárvore multicolorida H de G, com k vértices, para a qual $w(H)$ seja mínimo.[3]

(Não é essencial que o número de cores seja igual a k. A versão em que temos mais que k cores também faz sentido. Como veremos adiante, pode ser conveniente trabalhar com $6k$ cores.)

Podemos agora esboçar o algoritmo para o problema kMST. Para simplificar, vamos nos comportar como se os problemas kMST e COLORED-kMST exigissem como resposta apenas o peso $w(H)$ de uma solução H. Assim, o algoritmo esboçado abaixo recebe G, w e k e devolve o peso de uma solução de kMST(G, w, k). (Se essa instância do problema não tiver solução, devolve ∞.)

Esboço do Algoritmo COLOR-CODING (G, w, k)

```
1  min := ∞
2  para cada k-coloração c de V_G faça
3      h := COLORED-kMST (G, w, k, c)
4      se h < min então min := h
5  devolva min
```

Se implementarmos a linha 2 literalmente, o consumo de tempo do algoritmo será inaceitável, uma vez que o número de k-colorações de V_G é k^n. Veremos nas seções 5 a 7 que é possível implementar a linha 2 de modo a examinar apenas $2^{O(k)}\, n$ colorações.

[2] É preciso dizer que o algoritmo resolve um problema mais geral que kMST, pois não exige que os pesos das arestas sejam não-negativos.

[3] Dado que os pesos das arestas são não-negativos, a condição "multicolorida, com k vértices" poderia ser substituída pela condição "$c(V_H) = \{1, \ldots, k\}$". A segunda condição é equivalente à primeira. A equivalência fica particularmente evidente se todos os pesos forem estritamente positivos.

O subalgoritmo na linha 3 resolve o problema COLORED-kMST. Ele será discutido na seção 3. O problema COLORED-kMST é mais fácil que o problema kMST. De fato, enquanto o segundo envolve (implicitamente) a enumeração de n^k subconjuntos de V_G, o primeiro envolve apenas a enumeração das 3^k tripartições do conjunto de cores. Assim, a redução do problema kMST ao problema COLORED-kMST tem o efeito de trocar o fator n^k por um fator 3^k, que é menor que $2^{1.6k}$, na complexidade computacional do problema.

3 Sub-árvores multicoloridas com k vértices

Esta seção descreve nossa solução do subproblema COLORED-kMST. Ela foi inspirada no algoritmo de Betzler [4] (que por sua vez baseou-se nos algoritmos de Alon–Yuster–Zwick [1] e Scott–Ideker–Karp–Sharan [13]).

Seja c uma coloração do grafo G e sejam H e H' duas sub-árvores multicoloridas de G. Suponha que os conjuntos de cores das duas sub-árvores são disjuntos, ou seja, que $c(H) \cap c(H') = \emptyset$ (donde $V_H \cap V_{H'} = \emptyset$). Se uma aresta vv' de G tem uma ponta em H e outra em H' então o subgrafo $H + vv' + H'$ é uma árvore multicolorida. Graças a essa propriedade recursiva, o problema COLORED-kMST pode ser resolvido por programação dinâmica.

Seja A o conjunto $\{1, \ldots, k\}$ de todas as cores. O algoritmo constrói uma matriz W indexada por

$$V_G \times 2^A .$$

No fim da execução do algoritmo, para cada v em V_G e cada parte B de A,

> $W[v, B]$ é o peso mínimo de uma sub-árvore multicolorida H de G tal que $v \in V_H$ e $c(V_H) = B$,

Eis o algoritmo que resolve o problema COLORED-kMST:

Algoritmo COLORED-kMST (G, w, k, c)

```
01  A := {1, ..., k}   ▷ (conjunto de cores)
02  para cada B ⊆ A faça
03      para cada v em V_G faça
04          W[v, B] := ∞
05  para cada v em V_G faça
06      W[v, {c(v)}] := 0
07  para j := 2, ..., k faça
08      para cada B ⊆ A tal que |B| = j faça
09          para cada v em V_G faça
10              para cada vizinho v' de v faça
```

```
11              para cada bipartição ordenada (C, C') de B faça
12                  x := W[v, C] + w(vv') + W[v', C']
13                  se x < W[v, B] então W[v, B] := x
14      minw := min_{v∈V_G} W[v, A]
15      devolva minw
```

(Na linha 11, supõe-se que C e C' não são vazios.) No começo de cada iteração do bloco de linhas 08–13 (ou seja, para cada valor de j), vale a seguinte propriedade: para cada vértice v e cada parte B de A tal que $|B| < j$, $W[v, B]$ é o peso mínimo de uma árvore multicolorida H tal que $v \in V_H$ e $c(H) = B$.

As linhas 12–13 explicam o papel da multicoloração: digamos que H é uma sub-árvore multicolorida tal que $c(H) = C$, $v \in V_H$ e $w(H) = W[v, C]$; defina H' analogamente para v' e C'; como $C \cap C' = \emptyset$, as sub-árvores H e H' não têm vértices em comum, e portanto podem ser interligadas pela aresta vv' para produzir uma nova árvore.

Teorema: O consumo de tempo do algoritmo COLORED-kMST é

$$O(3^k\, n^2)\,,$$

sendo $n := |V_G|$.

Esboço da prova: A inicialização de W (linhas 01–06) consome $O(2^k n)$ unidades de tempo. Agora considere o processo iterativo principal, que ocupa as linhas 07–13. As linhas 09–10 fazem com que cada aresta de G seja visitada exatamente duas vezes, e assim contribuem um fator $n(n-1)$ para o consumo de tempo. A linha 08 faz com que cada subconjunto de tamanho j de A seja examinado uma vez, e assim contribui um fator C_j^k para o consumo de tempo. A linha 11 contribui um fator 2^j. A contribuição combinada das linhas 07, 08 e 11 é menor que $\sum_{j=1}^{k} \mathsf{C}_j^k\, 2^j = 3^k$. Portanto, o algoritmo consome tempo $O(3^k\, n^2)$. □

4 À procura de uma coloração ótima

Seja H um subgrafo de G com k vértices, por exemplo uma solução do kMST(G, w, k). Se colorirmos os vértices de G aleatoriamente com k cores, a probabilidade de que H resulte multicolorido é

$$\frac{k!}{k^k}\,,$$

pois há k^k diferentes k-colorações de H e $k!$ delas tornam H multicolorido. Observe que

$$\frac{k!}{k^k} > \frac{1}{\mathrm{e}^k} > \frac{1}{2.72^k} > \frac{1}{2^{1.5\,k}}\,.$$

Portanto, com grande probabilidade, uma em cada $2^{1.5k}$ colorações de V_G escolhidas aleatoriamente tornam H multicolorido. Para $k = 10$, por exemplo, pelo menos uma em cada 2^{15} colorações de G tornam H multicolorido.

Infelizmente, colorações aleatórias não são suficientes para mostrar que kMST é *FPT*. Precisamos de um algoritmo determinístico. Para isso é necessário, em tese, examinar todas as k-colorações de V_G.

5 Coloração de vértices versus hashing perfeito

Para revelar um candidato a solução do kMST, não é preciso examinar todas as colorações de V_G. Basta examinar uma família $c_1, \ldots, c_N$ de colorações que tenha a seguinte propriedade:

> para cada subconjunto K de V_G com k vértices, alguma coloração c_i torna K multicolorido.

Diz-se que uma tal família de colorações é C_k^V-perfeita. A presente seção descreve uma família C_k^V-perfeita. Essa primeira família é muito grande para nossas necessidades, mas prepara o terreno para uma segunda família, suficientemente pequena, a ser construída na próxima seção.

Por razões técnicas, é mais simples construir uma família perfeita se tivermos mais que k cores. Usaremos $3k$ cores nesta seção.

Hashing perfeito. Seja K uma parte do conjunto de vértices[4] $V := \{1, \ldots, n\}$. Se c é uma coloração de V e v é um elemento de V, diremos que $c(v)$ é a **cor** de v. Uma coloração c de V é **K-perfeita** se sua restrição a K for injetiva, ou seja, se $c(v) \neq c(v')$ para todo par (v, v') de elementos distintos de K. Em outras palavras, c é K-perfeita se não atribui uma mesma cor a dois diferentes elementos de K.

Em terminologia mais tradicional, uma coloração K-perfeita é conhecida como **hashing perfeito** ou **hashing sem colisões**. Mas essa terminologia não parece apropriada para o presente contexto.

Nesta subseção, nosso objetivo é construir um algoritmo que implemente uma $3k$-coloração K-perfeita, sendo $k := |K|$. Ao receber qualquer vértice v, o algoritmo deve devolver a cor de v.

Comecemos por discutir dois algoritmos óbvios mas insatisfatórios. Suponha que $v_1, \ldots, v_k$ são os elementos de K. Nosso primeiro algoritmo atribui cor i ao vértice v_i (para $i =$

[4] Na presente seção, a natureza dos elementos de V é irrelevante, uma vez que as arestas do grafo são ignoradas. Ainda assim, continuaremos a dizer que os elementos de V são *vértices*.

$1, \ldots, k$) e uma cor qualquer aos vértices em $V \setminus K$. Para implementar essa idéia, o algoritmo deve ter acesso a uma tabela $A[1\,..\,n]$ tal que $A[v_i] = i$ para todo i e $A[v]$ é arbitrário para v em $V \setminus K$. (O tempo necessário para a construção da tabela é irrelevante.) O algoritmo usa essa tabela para atribuir uma cor a qualquer vértice v em tempo constante. Infelizmente, a tabela ocupa muito espaço, o que torna o algoritmo inaceitável. Nosso segundo algoritmo supõe que $v_1 < \cdots < v_k$ e usa uma tabela $B[1\,..\,k]$ tal que $B[i] = v_i$. Ao receber v, o algoritmo executa uma busca binária na tabela para encontrar v_i que minimize $|v - v_i|$ e atribui cor i a v. Embora a tabela B ocupe pouco espaço, o algoritmo consome muito tempo: cerca de $\lg k$ operações aritméticas e comparações no pior caso. Isso torna nosso segundo algoritmo inaceitável.

Queremos um algoritmo cujas tabelas internas tenham $O(k)$ números razoavelmente pequenos (todos entre 1 e $2n$, digamos) e que seja capaz de calcular a cor de qualquer elemento de V com um número constante (ou seja, independente de k e de n) de operações aritméticas e comparações.

Para construir o algoritmo, usaremos uma função M definida da seguinte maneira: para quaisquer números naturais γ, p e k' e para qualquer elemento v de V,

$$M(\gamma, v, p, k') := (\gamma v \bmod p) \bmod k' \,. \tag{1}$$

Aqui, o operador mod é definido de maneira um pouco diferente da usual: se b divide a então $a \bmod b := b$ (e não 0, como é usual); senão, $a \bmod b$ é o resto da divisão de a por b. Assim, o valor de $M(\gamma, v, p, k')$ está em $\{1, \ldots, k'\}$ quaisquer que sejam γ e p. Em geral, p é um número primo maior que n e γ é um número em $\{1, \ldots, p-1\}$.

No que segue, usaremos a expressão verbal "a função $M(\gamma, *, p, k')$" para designar a função que leva qualquer elemento v de V no número $M(\gamma, v, p, k')$. No mesmo espírito, usaremos a expressão "o conjunto $M(\gamma, K, p, k')$" para designar o conjunto de todos os números da forma $M(\gamma, v, p, k')$ com v em K. A função $M(\gamma, *, p, k')$ nada mais é que uma k'-coloração de V. Quanto a $M(\gamma, K, p, k')$, esse é o conjunto de cores atribuídas aos elementos de K.

Se pudéssemos escolher γ e p de modo que função $M(\gamma, *, p, k)$ fosse uma coloração K-perfeita de V, nosso problema estaria resolvido. Infelizmente, tudo indica que tais γ e p não existem em geral. A existência de valores apropriados de γ e p pode ser garantida somente se o número de cores for muito maior que k. É o que mostraremos a seguir.

Diremos que um número γ é **bom** para um dado subconjunto B de V se a restrição da função $M(\gamma, *, p, |B|^2)$ a B for injetiva. Em outras palavras, γ é bom para B se a $|B|^2$-coloração $M(\gamma, *, p, |B|^2)$ é B-perfeita. Fredman *et al.* mostraram [7, 5] a existência de números bons:

> **Lema 5.1**: Para qualquer B de V e qualquer número primo $p > n$, algum γ em $\{1, \ldots, p-1\}$ é bom para B.

(Veja prova no apêndice 9.) O uso do lema com $B = K$ não dá bons resultados pois

k^2 é excessivamente grande. Será preciso aplicar a V, previamente, uma k-coloração que quebre K em pedaços suficientemente pequenos. O Lema 5.1 poderá então ser aplicado a cada um desses pedaços. É o que discutiremos a seguir.

Dados números β e p, denotaremos por B_j o conjunto dos elementos de K que recebem cor j na k-coloração $M(\beta, *, p, k)$ de V. Portanto,

$$B_j := \{v \in K : M(\beta, v, p, k) = j\} \,. \tag{2}$$

(Deveríamos escrever "$B_j^{\beta,p}$", mas vamos deixar "β, p" subentendido para simplificar a notação.) Cada B_j é conhecido na literatura como *collision bucket*. É claro então que

$$|B_1| + \cdots + |B_k| = k \,, \tag{3}$$

uma vez que $\{B_1, \ldots, B_k\}$ é uma partição de K. Fredman, Komlós e Szemerédi observaram [7, 5] que β e p podem ser escolhidos de modo que todos os B_j sejam pequenos:

> **Lema 5.2**: Para qualquer número primo $p > n$, existe β no conjunto $\{1, \ldots, p-1\}$ tal que
> $$|B_1|^2 + \cdots + |B_k|^2 < 3k \,,$$
> sendo $k := |K|$.

(Veja prova no apêndice 9.) Podemos agora descrever o algoritmo de $3k$-coloração K-perfeita proposto por Fredman, Komlós e Szemerédi [7, 5]. O primeiro passo do algoritmo faz uma k-coloração de V do tipo indicado no Lema 5.2. Essa coloração induz uma partição $B_1, \ldots, B_k$ de K em blocos monocromáticos. No segundo passo, para cada j, o algoritmo usa o Lema 5.1 para fazer uma $|B_j|^2$-coloração B_j-perfeita de V. O terceiro passo do algoritmo faz "translações" apropriadas de todas as colorações determinadas no segundo passo para obter uma $3k$-coloração K-perfeita de V.

Algoritmo FKS (v)
1 $j := M(\beta, v, p, k)$
2 $s_0 := 0$
3 devolva $s_{j-1} + M(\gamma_j, v, p, b_j^2)$

O algoritmo usa as constantes numéricas

$$p,\ \beta,\ b_1, \ldots, b_k,\ s_1, \ldots, s_k,\ \gamma_1, \ldots, \gamma_k, \tag{4}$$

sendo

- p é o primeiro número primo maior que n,
- β é um número em $\{1, \ldots, p-1\}$ que satisfaz o Lema 5.2,
- b_j é a cardinalidade do conjunto B_j definido em (2),

- s_j é o número $b_1^2 + \cdots + b_j^2$ (veja Lema 5.2),
- γ_j está em $\{1, \ldots, p-1\}$ e é bom para B_j (veja Lema 5.1).

O tempo necessário para calcular essas constantes é irrelevante; eles podem ter sido determinados por busca exaustiva. O algoritmo pode ser formalizado assim:

Uma família C_k^V-perfeita de $3k$-colorações. Trataremos agora de colorações que são K-perfeitas para muitos conjuntos K simultaneamente. Uma família $\{c_1, \ldots, c_N\}$ de $3k$-colorações do conjunto de vértices $V := \{1, \ldots, n\}$ é k-**perfeita** se, para cada conjunto K de k vértices, alguma coloração c_i é K-perfeita.

A existência de uma tal família com $N = \mathsf{C}_k^n$ membros segue trivialmente da subseção anterior. Mas é claro que gostaríamos de obter uma família bem menor.

O algoritmo abaixo examina todas as possíveis combinações das constantes (4) do algoritmo FKS e assim gera uma família C_k^V-perfeita de $3k$-colorações de V.

Algoritmo FAMÍLIA-PERFEITA-FKS (n, k)

```
01  seja p o primeiro número primo maior que n
02  para cada β em {1,...,p−1} faça
03      para cada vetor de bits Q[1..2k−1] de posto k−1 faça
04          (b_1,...,b_k) := DECOD(Q[1..2k−1])
05          se b_1^2 + ··· + b_k^2 < 3k então
06              para cada elemento (γ_1,...,γ_k) de {1,...,p−1}^k faça
07                  para v := 1,...,n faça
08                      j := M(β, v, p, k)
09                      c(v) := b_1^2 + ··· + b_{j−1}^2 + M(γ_j, v, p, b_j^2)
10                  imprima c(1),...,c(n)
```

Na linha 03, o **posto** de um vetor de bits é o número de ocorrências do bit 0 no vetor. Todo vetor de bits de posto $k-1$ tem a forma

$$1^{b_1}\,0\,1^{b_2}\,0\,\cdots\,0\,1^{b_k}\,.$$

A rotina DECOD converte um tal vetor na seqüência $(b_1, \ldots, b_k)$. (Por exemplo, ao receber o vetor de bits 11101001100, que tem posto 5, a rotina DECOD devolve a seqüência $(3, 1, 0, 2, 0, 0)$.) Na linha 04, DECOD é aplicada a um vetor de posto $k-1$ com $2k-1$ bits e portanto $b_1 + \cdots + b_k = k$.

Mostraremos a seguir que o algoritmo FAMÍLIA-PERFEITA-FKS de fato produz uma família C_k^V-perfeita. Seja $\dot{K}$ um conjunto de k vértices e sejam $\dot{\beta}, \dot{b_1}, \ldots, \dot{b_k}, \dot{\gamma_1}, \ldots, \dot{\gamma_k}$ os valores das constantes em (4) apropriados para $\dot{K}$. Queremos mostrar que em alguma das execuções do bloco de linhas 07–09 tem-se $\beta = \dot{\beta}$ bem como $b_j = \dot{b_j}$ e $\gamma_j = \dot{\gamma_j}$ para cada j.

Isso é evidente com relação a β e $\gamma_1, \ldots, \gamma_k$. Como $\dot{b_1} + \cdots + \dot{b_k} = k$ em virtude de (3), o vetor $1^{\dot{b_1}}\,0\,1^{\dot{b_2}}\,0 \cdots 0\,1^{\dot{b_k}}$ tem $2k-1$ bits e seu posto é $k-1$. Portanto, um dos vetores de bits examinados na linha 03 certamente coincide com $\dot{b_1}, \ldots, \dot{b_k}$. Em suma, pelo menos uma das colorações produzidas pelo algoritmo FAMÍLIA-PERFEITA-FKS é $\dot{K}$-perfeita.

Quantas colorações o algoritmo FAMÍLIA-PERFEITA-FKS imprime? Ou seja, quantas vezes o algoritmo executa a linha 10? Eis as estimativas:

- a linha 02 induz $p-1$ iterações,
- a linha 03 induz $\mathsf{C}_{k-1}^{2k-1} < 2^{2k-1}$ iterações e
- a linha 06 induz $(p-1)^k$ iterações.

Como se sabe, existe um número primo entre n e $2n$. Portanto $p < 2n$ e assim o número total de execuções da linha 10 é menor que $n^{k+1}\,2^{3k}$.

Infelizmente, esse número é muito grande, muito maior que C_k^n. Isso acontece porque muitas das combinações de constantes geradas pelo algoritmo não têm a forma especificada em (4). O principal responsável por esse fato é a linha 06 do algoritmo. Na seção seguinte, descreveremos uma variante de FKS que substitui os k números $\gamma_1, \ldots, \gamma_k$ por apenas $1 + \lfloor \lg k \rfloor$ números.

6 Coloração versus hashing perfeito: parte II

Para obter uma família C_k^V-perfeita suficientemente pequena, será preciso obter um algoritmo de hashing mais "poderoso" que FKS. Para facilitar a construção do algoritmo, será necessário aumentar o número de cores de $3k$ para $6k$.

Um hashing perfeito melhorado. Dado um subconjunto K de $V := \{1, \ldots, n\}$, queremos construir um algoritmo que produza uma $6k$-coloração K-perfeita de V, sendo $k := |K|$. O passo principal nessa construção é uma certa extensão do Lema 5.1, que será aplicada a um universo V' diferente de V.

Seja n' um número natural e p' um número primo maior que n'. Diremos que um número γ é **tolerado** por um dado subconjunto B de $V' := \{1, \ldots, n'\}$ se a função $M(\gamma, *, p', 2|B|^2)$ é uma injeção quando restrita a B. Ou seja, γ é tolerado por B se a $2|B|^2$-coloração $M(\gamma, *, p', 2|B|^2)$ de V' for B-perfeita. O Lema 5.1 admite a seguinte extensão [14]:

> **Lema 6.1**: Para qualquer coleção $B_1, \ldots, B_l$ de não mais que n' subconjuntos de V', algum γ em $\{1, \ldots, p'-1\}$ é tolerado por pelo menos metade dos elementos de $\{B_1, \ldots, B_l\}$.

(A prova do lema será omitida, embora seja elementar.) O lema tem a seguinte conseqüência [14]:

Corolário 6.2: Seja $L := 1 + \lfloor \lg k \rfloor$. Dados subconjuntos $B_1, \ldots, B_k$ de V', existem números $\gamma_1, \ldots, \gamma_L$ em $\{1, \ldots, p'-1\}$ e índices $f_1, \ldots, f_k$ com valores em $\{1, \ldots, L\}$ tais que $f_1 + \cdots + f_k < 2k$ e

$$B_j \text{ tolera } \gamma_{f_j}$$

para $j = 1, \ldots, k$.

Prova: Seja J_1 o conjunto $\{1, \ldots, k\}$ e repita a seguinte construção para $i = 1, 2, 3, \ldots$ Seja γ_i um número tolerado por pelo menos metade dos elementos de $\{B_j : j \in J_i\}$. (O Lema 6.1 garante a existência de um tal γ_i.) Seja J_{i+1} o conjunto dos j em J_i tais que B_j não tolera γ_i. Defina $f_j := i$ para todo j em $J_i \setminus J_{i+1}$.

Por hipótese, $|J_{i+1}| \leq \frac{1}{2}|J_i|$. Portanto, $|J_i| \leq k/2^{i-1}$ para todo $i \geq 1$. Segue-se que $J_i = \emptyset$ para todo $i > L$. Assim, podemos interromper a construção quando i ultrapassar L. Teremos definido $\gamma_1, \ldots, \gamma_L$ e $f_1, \ldots, f_k$. Como $f_j = i$ se e somente se $j \in J_i \setminus J_{i+1}$, temos

$$\begin{aligned} f_1 + \cdots + f_k &= \sum |J_i \setminus J_{i+1}|\, i \\ &\leq \sum i\, |J_i| \\ &\leq \sum i\, k/2^{i-1} \\ &= 2k \sum i/2^i \\ &< 2k\,, \end{aligned}$$

onde $\sum$ representa $\sum_{i=1}^{L}$. □

Slot e van Emde Boas [14] propuseram o seguinte algoritmo de $6k$-coloração K-perfeita. No primeiro passo, o algoritmo escolhe um número α que seja bom para K (o Lema 5.1 garante a existência de α) e aplica a k^2-coloração $M(\alpha, *, p, k^2)$ a V. A partir desse ponto, o algoritmo passa a tratar o conjunto de cores $\{1, \ldots, k^2\}$ como se fosse um novo conjunto de vértices. Assim, o conjunto de vértices original $V = \{1, \ldots, n\}$ é reduzido ao conjunto $V' := M(\alpha, V, p, k^2)$. Essa mesma "compressão" transforma K em

$$K' := M(\alpha, K, p, k^2)\,.$$

Como a coloração é K-perfeita, temos $|K'| = k$. O objetivo do algoritmo passa a ser a obtenção de uma $6k$-coloração K'-perfeita de V'. Os passos subseqüentes do algoritmo são semelhantes aos do algoritmo FAMÍLIA-PERFEITA-FKS examinado na seção 5. Assim, o segundo passo do algoritmo calcula uma k-coloração de V' do tipo indicado no Lema 5.2. Essa coloração induz uma partição $B_1, \ldots, B_k$ de K', sendo

$$B_j := \{v' \in K' : M(\beta, v', p', k) = j\} \tag{5}$$

para cada j. No terceiro passo, para cada cor j, o algoritmo calcula uma $2|B_j|^2$-coloração B_j-perfeita de V'. No quarto passo, o algoritmo faz "translações" apropriadas de todas essas colorações de modo a obter uma $6k$-coloração K'-perfeita de V'. Como se vê, o algoritmo depende das constantes

$$L,\ p,\ \alpha,\ p',\ \beta,\ b_1, \ldots, b_k,\ s_1, \ldots, s_k,\ \gamma_1, \ldots, \gamma_L,\ f_1, \ldots, f_k\,, \tag{6}$$

onde $k := |K|$. Essas constantes são assim definidos:

- L é o número $1 + \lfloor \lg k \rfloor$,
- p é o primeiro número primo maior que n,
- α é um número em $\{1, \ldots, p-1\}$ bom para K (veja Lema 5.1),
- p' é o primeiro número primo maior que k^2,
- β é um número em $\{1, \ldots, p'-1\}$ que satisfaz o Lema 5.2 com k^2, p', V' e K' nos papéis de n, p, V e K respectivamente,
- b_j é a cardinalidade do conjunto B_j definido em (5),
- s_j é o número $2b_1^2 + \cdots + 2b_j^2$ (veja Lema 5.2),
- $\gamma_1, \ldots, \gamma_L$ são números em $\{1, \ldots, p'-1\}$,
- f_j é tal que B_j tolera γ_{f_j} e
- $f_1 + \cdots + f_k < 2k$ (veja Corolário 6.2).

A descrição formal do algoritmo é simples:

Algoritmo SB (v)
0 $v' := M(\alpha, v, p, k^2)$
1 $j := M(\beta, v', p', k)$
2 $i := f_j$
3 $s_0 := 0$
4 devolva $s_{j-1} + M(\gamma_i, v', p', 2b_j^2)$

O algoritmo tem as propriedades desejadas: o número de operações aritméticas não depende de k nem de n e o número de constantes (veja 6) é $O(k)$.

Uma família C_k^V-perfeita de $6k$-colorações. Da mesma forma que construímos o algoritmo FAMÍLIA-PERFEITA-FKS a partir do algoritmo FKS, podemos construir um algoritmo FAMÍLIA-PERFEITA-SB a partir de SB.

Algoritmo FAMÍLIA-PERFEITA-SB (n, k)
01 seja p o primeiro número primo maior que n
02 seja p o primeiro número primo maior que k^2
03 $L := 1 + \lfloor \lg k \rfloor$
04 para cada α em $\{1, \ldots, p-1\}$ faça
05 para cada β em $\{1, \ldots, p'-1\}$ faça
06 para cada vetor de bits $Q[1..2k-1]$ de posto $k-1$ faça
07 $(b_1, \ldots, b_k) :=$ DECOD $(Q[1..2k-1])$
08 se $2b_1^2 + \cdots + 2b_k^2 < 6k$ então

```
09          para cada elemento (γ_1, ..., γ_L) de {1, ..., p'−1}^L faça
10            para cada vetor de bits F[1..3k] de posto k faça
11              (f_1, ..., f_k, f_{k+1}) := DECOD (F[1..3k])
12              se 1 ≤ f_j ≤ L para j := 1, ..., k então
13                para v := 1, ..., n faça
14                  v' := M(α, v, p, k^2)
15                  j := M(β, v', p', k)
16                  i := f_j
17                  c(v) := 2b_1^2 + ··· + 2b_{j−1}^2 + M(γ_i, v', p', 2b_j^2)
18                imprima c(1), ..., c(n)
```

Para mostrar que o algoritmo produz uma família C_k^V-perfeita, tome um conjunto $\dot{K}$ de k vértices e sejam $\dot{\beta}, \dot{b_1}, \ldots, \dot{b_k}, \dot{\gamma_1}, \ldots, \dot{\gamma_L}, \dot{f_1}, \ldots, \dot{f_k}$ os valores das constantes (6) apropriados para $\dot{K}$. É preciso mostrar que alguma das execuções do bloco de linhas 14–17 usa $\beta = \dot{\beta}$, $b_j = \dot{b_j}$, $\gamma_i = \dot{\gamma_i}$ e $f_j = \dot{f_j}$. Isto é óbvio com relação a β e $\gamma_1, \ldots, \gamma_L$ e já foi discutido com relação a $b_1, \ldots, b_k$ na seção 5. Resta discutir $f_1, \ldots, f_k$. Como $\dot{f_1} + \cdots + \dot{f_k} < 2k$, o vetor

$$1^{\dot{f_1}}\, 0\, 1^{\dot{f_2}}\, 0 \cdots 0\, 1^{\dot{f_k}}$$

tem menos que $3k-1$ bits e posto $k-1$. Portanto, ao examinar todas os vetores de bits de comprimento $3k$ e posto k nas linhas 10–11, o algoritmo fatalmente encontrará $\dot{f_1}, \ldots, \dot{f_k}$. Isso encerra a prova de que o algoritmo produz uma família C_k^V-perfeita.

Considere agora o número de colorações geradas pelo algoritmo, ou seja, o número de execuções da linha 18. Observe que

- a linha 04 induz $p - 1$ iterações,
- a linha 05 induz $p' - 1$ iterações,
- a linha 06 induz $\mathsf{C}_{k-1}^{2k-1} < 2^{2k-1}$ iterações,
- a linha 09 induz $(p'-1)^L < (p')^{1+\lg k}$ iterações,
- a linha 10 induz $\mathsf{C}_k^{3k} < 2^{3k}$ iterações.

Como existe um número primo entre n e $2n$, temos $p < 2n$. Analogamente, $p' < 2k^2$. Portanto, o número total de execuções da linha 18 é menor que

$$\begin{aligned} 2n \cdot 2k^2 \cdot 2^{2k-1} \cdot (2k^2)^{1+\lg k} \cdot 2^{3k} &= 2n \cdot 2k^2 \cdot \exp_2(2k-1) \cdot (2k^2)^{1+\lg k} \cdot \exp_2(3k) \\ &= n\, k^4\, k^{2\lg k} \exp_2(\lg k + 5k + 2) \\ &= n \exp_2(4\lg k + 2\lg^2 k + \lg k + 5k + 2) \\ &= n \exp_2(5\lg k + 2\lg^2 k + 5k + 2)\,, \end{aligned}$$

onde $\exp_2(x) = 2^x$.

Como $2\lg^2 k < k + 19$ e $5\lg k < k + 8$, podemos dizer que o algoritmo gera uma família C_k^V-perfeita de $6k$-colorações com menos que

$$2^{7k+29}\, n$$

membros.

7 Família perfeita de k-colorações

Esta seção transforma as $6k$-colorações da seção anterior em k-colorações. Preliminarmente, convém discutir k-colorações K-perfeitas para um subconjunto fixo K de V_G com k vértices. Diz-se que tais colorações são **minimais**.

Hashing perfeito minimal. Não é difícil transformar a $6k$-coloração K-perfeita discutida na primeira parte da seção anterior em uma k-coloração K-perfeita. Para isso, acrescente à lista de constantes (6) os vetores

$$R[1\,..\,6k] \quad \text{e} \quad S[1\,..\,6k]\,,$$

onde

- $R[1\,..\,6k]$ é o vetor característico do subconjunto $\Gamma_1 \cup \cdots \cup \Gamma_k$ de $\{1, \ldots, 6k\}$, sendo Γ_j o conjunto $\{s_{j-1} + M(\gamma_j, v', p', 2b_j^2) : v' \in B_j\}$ e
- $S[h]$ é o número $R[1] + \cdots + R[h]$.

Como $|\Gamma_1 \cup \cdots \cup \Gamma_k| = k$, o vetor R tem posto $5k$, ou seja, tem exatamente $5k$ bits 0. Observe também que $S[h] \in \{0, 1, \ldots, k\}$. Pode-se dizer que o par (R, S) é uma "tabela de compressão".

Agora troque o algoritmo SB pelo seguinte:

Algoritmo SB-II(v)
0 $v' := M(\alpha, v, p, k^2)$
1 $j := M(\beta, v', p', k)$
2 $i := f_j$
3 $s_0 := 0$
4 $h := s_{j-1} + M(\gamma_i, v', p', 2b_j^2)$
5 se $S[h] = 0$ então devolva 1 senão devolva $S[h]$

Uma família C_k^V-perfeita de k-colorações. O algoritmo SB-II pode ser usado para modificar o bloco de linhas 13–18 do algoritmo FAMÍLIA-PERFEITA-SB de modo que ele passe a produzir k-colorações. Eis o resultado:

Algoritmo FAMÍLIA-PERFEITA-SB-II (n, k)

```
01  seja p o primeiro número primo maior que n
02  seja p o primeiro número primo maior que k^2
03  L := 1 + ⌊lg k⌋
04  para cada α em {1, ..., p−1} faça
05      para cada β em {1, ..., p'−1} faça
06          para cada vetor de bits Q[1..2k−1] de posto k−1 faça
07              (b_1, ..., b_k) := DECOD (Q[1..2k−1])
08              se 2b_1^2 + ··· + 2b_k^2 < 6k então
09                  para cada elemento (γ_1, ..., γ_L) de {1, ..., p'−1}^L faça
10                      para cada vetor de bits F[1..3k] de posto k faça
11                          (f_1, ..., f_k, f_{k+1}) := DECOD (F[1..3k])
12                          se 1 ≤ f_j ≤ L para j := 1, ..., k então
13                              para cada vetor de bits R[1..6k] de posto 5k faça
14                                  para v := 1, ..., n faça
15                                      v' := M(α, v, p, k^2)
16                                      j := M(β, v', p', k)
17                                      i := f_j
18                                      h := 2b_1^2 + ··· + 2b_{j−1}^2 + M(γ_i, v', p', 2b_j^2)
19                                      c(v) := R[1] + ··· + R[h]
20                                      se c(v) = 0 então c(v) := 1
21                                  imprima c(1), ..., c(n)
```

Quantas colorações o algoritmo produz, ou seja, quantas vezes a linha 21 é executada? Eis as estimativas:

- a linha 04 induz $p - 1 < 2n$ iterações,
- a linha 05 induz $p' - 1 < 2k^2$ iterações,
- a linha 06 induz $\mathsf{C}_{k-1}^{2k-1} < 2^{2k-1}$ iterações,
- a linha 09 induz $(p' - 1)^L < (2k^2)^{1+\lg k}$ iterações,
- a linha 10 induz C_k^{3k} iterações,
- a linha 13 induz C_k^{6k} iterações.

Como se sabe, $\mathsf{C}_k^n \le (n\mathrm{e}/k)^k$ para todo n e todo k [10, p.1077]. Em particular, $\mathsf{C}_k^3 \le (3\mathrm{e})^k < 2^{3.03k}$ e $\mathsf{C}_k^6 \le (6\mathrm{e})^k < 2^{4.03k}$. Portanto, o número total de execuções da linha 21 é menor que

$$\begin{aligned} 2n \cdot 2k^2 \cdot 2^{2k-1} \cdot (2k^2)^{1+\lg k} \cdot 2^{3.03k} \cdot 2^{4.03k} &< 2^{9.1k}\, k^{2\lg k}\, 2^{\lg k}\, k^4\, 2^2\, n \\ &< 2^{9.1k}\, 2^{2\lg^2 k}\, 2^{5\lg k}\, 2^2\, n\,. \end{aligned}$$

Podemos calcular isso de maneira um pouco mais precisa. De acordo com Stănică [15],

$$\mathsf{C}_k^{mk} < \frac{1}{\sqrt{2\pi k}} \, \frac{m^{mk+\frac{1}{2}}}{(m-1)^{(m-1)k+\frac{1}{2}}} < \frac{\sqrt{m}}{\sqrt{(m-1)k}} \, \frac{m^{mk}}{(m-1)^{(m-1)k}}$$

para todo m, donde

$$\mathsf{C}_k^{3k} < \frac{\sqrt{3}}{\sqrt{2k}} \, \frac{3^{3k}}{2^{2k}} < \frac{1.3}{\sqrt{k}} \, 2^{2.8k} \qquad \text{e} \qquad \mathsf{C}_k^{6k} < \frac{\sqrt{6}}{\sqrt{5k}} \, \frac{6^{6k}}{5^{5k}} < \frac{1.1}{\sqrt{k}} \, 2^{4.0k} \, .$$

Portanto, o número total de execuções da linha 21 é menor que

$$\begin{aligned} 2n \cdot 2k^2 \cdot 2^{2k-1} \cdot (2k^2)^{1+\lg k} \cdot \tfrac{1.3}{\sqrt{k}} \, 2^{2.8k} \cdot \tfrac{1.1}{\sqrt{k}} \, 2^{4.0k} &< 2^{8.8k} \, k^{2\lg k} \, k^3 \, 2^{\lg k} \, 2^3 \, n \\ &< 2^{8.8k} \, 2^{2\lg^2 k} \, 2^{4\lg k} \, 2^3 \, n \, . \end{aligned}$$

Como $2\lg^2 k < k + 19$ e $4 \lg k < k + 5$, podemos dizer que o algoritmo gera uma família C_k^V-perfeita de k-colorações com menos que

$$2^{11k+27} \, n$$

membros. (Na verdade, essa delimitação é excessiva, pois não estamos levando em conta o efeito das linhas 08 e 13 do algoritmo.) De modo mais grosseiro, pode-se dizer que temos uma família C_k^V-perfeita com

$$2^{O(k)} \, n$$

k-colorações. (O trabalho de Schmidt e Siegel [12] prova a existência de uma família C_k^V-perfeita de k-colorações ainda menor, com apenas $2^{O(k)} \lg^2 n$ membros. A construção explícita de uma tal família usa as idéias empregadas acima, mas leva a "compressão" um passo adiante.)

8 Juntando as duas partes

O algoritmo COLOR-CODING que esboçamos na seção 2 é o resultado da superposição dos algoritmos COLORED-kMST e FAMÍLIA-PERFEITA-SB-II discutidos nas seções 3 e 7 respectivamente. Para superpor esses dois algoritmos basta

> trocar a linha 22 ("imprima $c(1) \ldots, c(n)$") do algoritmo FAMÍLIA-PERFEITA-SB-II por uma invocação do algoritmo COLORED-kMST.

O algoritmo COLOR-CODING que acabamos de descrever resolve o problema kMST e consome tempo

$$\begin{aligned} 2^{11k+27} \, n \cdot O(3^k \, n^2) &= O(2^{11k} \, 3^k \, n^3) \\ &= O(2^{11k} \, 2^{1.6k} \, n^3) \\ &= O(2^{13k} \, n^3) \, . \end{aligned}$$

Em termos mais grosseiros, podemos dizer que o consumo de tempo é $2^{O(k)} \, n^3$.

9 Apêndice: Prova dos Lemas 5.2 e 5.1

Seja $V := \{1, \ldots, n\}$ e p um número primo maior que n. Dados números naturais β e l, seja $M(\beta, *, p, l)$ a l-coloração de V que já definimos em (1), isto é,

$$M(\beta, v, p, l) := (\beta v \bmod p) \bmod l$$

para todo v em V. (Convém lembrar que $a \bmod b := b$ se b divide a.) Dado um subconjunto K de V e um número l, para $j = 1, \ldots, l$, seja B_j o subconjunto

$$\{v \in K : M(\beta, v, p, l) = j\} \tag{7}$$

de K. Trata-se do mesmo conjunto já definido em (7). É claro que B_j depende de β e de l, ainda que a notação não deixe isso explícito. Seja $b_j := |B_j|$ e $k := |K|$.

Lema [7]: Se $l \geq k$ então existe β no conjunto $\{1, \ldots, p-1\}$ tal que

$$\textstyle\sum_{j=1}^{l} \mathsf{C}_2^{b_j} \;<\; \frac{k^2}{l}\,.$$

Prova: Comecemos com uma observação sobre aritmética modular, válida para todo número z em $\{-p+1, \ldots, -2, -1\} \cup \{1, 2, \ldots, p-1\}$. Como p é primo, existe um inteiro β tal que $\beta z \bmod p = 1$. Em particular, existe um tal β em $\{1, \ldots, p-1\}$. De modo mais geral, para todo i em $\{1, \ldots, p-1\}$, existe β em $\{1, \ldots, p-1\}$ tal que $\beta z \bmod p = i$. Logo, o conjunto de todos os números da forma $\beta z \bmod p$ com $\beta = 1, \ldots, p-1$ é igual a $\{1, \ldots, p-1\}$. Segue daí que, para cada i em $\{1, \ldots, p-1\}$, existe apenas um β em $\{1, \ldots, p-1\}$ tal que $\beta z \bmod p = i$. Assim, para qualquer $P \subseteq \{1, \ldots, p-1\}$,

$$|\{\beta \in \{1, \ldots, p-1\} : \beta z \bmod p \in P\}| \;\leq\; |P|\,. \tag{8}$$

Feita essa observação preliminar, considere a soma $\sum_{\beta=1}^{p-1} \sum_{j=1}^{l} \mathsf{C}_2^{b_j}$. Se rearranjarmos os termos dessa soma teremos

$$\textstyle\sum_{\beta=1}^{p-1} \sum_{j=1}^{l} \mathsf{C}_2^{b_j} \;=\; \sum_{\{x,y\} \in \mathsf{C}_2^K} \sum_{\beta=1}^{p-1} [M(\beta, x, p, l) = M(\beta, y, p, l)]\,,$$

onde expressões da forma $[E = E']$ valem 1 se $E = E'$ e 0 em caso contrário. Mas

$$\begin{aligned}
&\textstyle\sum_{\beta=1}^{p-1} [M(\beta, x, p, l) = M(\beta, y, p, l)] \\
&= \textstyle\sum_{\beta=1}^{p-1} [(\beta(x-y) \bmod p) \bmod l = l] && (9)\\
&= |\{\beta \in \{1, \ldots, p-1\} : \beta(x-y) \bmod p \in A \cup B\}| && (10)\\
&\leq |A| + |B| && (11)\\
&\leq 2\,\frac{p-1}{l}\,. && (12)
\end{aligned}$$

Na linha (10),

$$A := \{l, 2l, 3l, \ldots, \xi l\} \quad \text{e} \quad B := \{p-l, p-2l, p-3l, \ldots, p-\xi l\}\,,$$

com $\xi = \lfloor p/l \rfloor$. A linha (11) decorre imediatamente de (8). A linha (12) vale porque p é primo e portanto $\lfloor p/l \rfloor \leq (p-1)/l$, Logo,

$$\begin{aligned} \sum_{\beta=1}^{p-1} \sum_{j=1}^{l} \mathsf{C}_2^{b_j} &\leq \mathsf{C}_2^k \cdot \sum_{\beta=1}^{p-1} [M(\beta, x, p, l) = M(\beta, y, p, l)] \quad (13) \\ &< k^2 \frac{p-1}{l} \,. \quad (14) \end{aligned}$$

Segue daí que existe β em $\{1, \ldots, p-1\}$ tal que $\sum_{j=1}^{l} \mathsf{C}_2^{b_j} < k^2/l$, como queríamos demonstrar. □

Lema 5.2: Existe β no conjunto $\{1, \ldots, p-1\}$ tal que

$$|B_1|^2 + \cdots + |B_k|^2 < 3k \,.$$

Prova: De acordo com o lema acima, existe β em $\{1, \ldots, p-1\}$ tal que $\sum_{j=1}^{k} \mathsf{C}_2^{b_j} < k^2/k = k$. Logo,

$$\begin{aligned} |B_1|^2 + \cdots + |B_k|^2 &= \sum_{j=1}^{k} b_j^2 \\ &= \sum_{j=1}^{k} b_j + 2 \sum_{j=1}^{k} b_j(b_j - 1)/2 \\ &= k + 2 \sum_{j=1}^{k} \mathsf{C}_2^{b_j} \\ &< k + 2k \\ &= 3k \,, \end{aligned}$$

como queríamos demonstrar. □

Lema 5.1: Para qualquer parte B de V, algum γ em $\{1, \ldots, p-1\}$ é bom para B.

Prova: De acordo com o lema acima, existe γ em $\{1, \ldots, p-1\}$ tal que

$$\sum_{j=1}^{k^2} \mathsf{C}_2^{b_j} \;<\; k^2/k^2 \;=\; 1 \,.$$

Portanto, $\mathsf{C}_2^{b_j} = 0$ para todo j e assim $|B_j| \leq 1$ para todo j. Logo γ é bom para B. □

10 Agradecimentos

Sou grato a Edson N. Cáceres (Universidade Federal do Mato Grosso do Sul, Campo Grande) pelas interessantes discussões sobre a classe FPT e por ter chamado minha atenção para a dissertação de Betzler [4].

Referências

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. 2, 3, 4

[2] Sanjeev Arora and George Karakostas. A 2+epsilon approximation algorithm for the k-MST problem. In *Symposium on Discrete Algorithms (SODA)*, pages 754–759, 2000. 1

[3] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, 2002. 2, 3

[4] Nadja Betzler. Steiner tree problems in the analysis of biological networks. Master's thesis, Universität Tübingen, 2006. Advisors: R. Niedermeier et al. 2, 3, 4, 18

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001. 7, 8

[6] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, 1999. 2

[7] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. 7, 8, 17

[8] Naveen Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th FoCS*, 1996. 1

[9] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. 2

[10] A.M. Odlyzko. *Handbook of Combinatorics*, volume 2, chapter 22, Asymptotic Enumeration Methods, pages 1063–1229. MIT Press and North-Holland, 1995. 15

[11] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. Spanning trees – short or small. *SIAM J. on Discrete Mathematics*, 9(2):178–200, 1996. Preliminary version appeared in SODA'1994, p.546-555. 1

[12] Jeanette P. Schmidt and Alan Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM J. Comput.*, 19(5):775–786, 1990. 16

[13] J. Scott, T. Ideker, R.M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Computational Biology*, 13(2):133–144, 2006. 2, 3, 4

[14] C. Slot and P. van Emde Boas. On tape versus core an application of space efficient perfect hash functions to the invariance of space. In *STOC '84: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 391–400. ACM Press, 1984. 10, 11

[15] P. Stănică. Good lower and upper bounds on binomial coefficients. *J. Inequalities in Pure and Applied Mathematics*, 2, 2001. http://jipam.vu.edu.au. 16