Complexidade de circuitos

Caetano Jimenez Carezzato
Rodrigo Nonamor Pereira Mariano de Souza

16 de julho de 2001

Sumário

1 Introdução

Neste trabalho, vamos fazer um estudo de alguns aspectos computacionais do formalismo bem conhecido de circuito booleano. Circuitos booleanos realizam funções entre vetores de *bits*, e podem ser vistos como um modelo de computação. É natural então pesquisar-se medidas de complexidade para a realização de tais funções por circuitos. Vamos focalizar nosso estudo na descrição de algumas cotas inferiores para essas medidas, especialmente para um tipo especial de função denominada monótona.

2 Definições básicas

Faremos nesta seção uma série de definições básicas e tradicionais no estudo de circuitos booleanos. Vamos utilizar livremente alguns conceitos relativos a Teoria dos Grafos e Álgebra Booleana. Em particular, vamos usar os seguintes formalismos:

- Dado um grafo dirigido $G = (V, E)$ e $v \in V$, vamos denotar por $d_{in}(v)$ e $d_{out}(v)$ o grau de entrada e o de saída de v, respectivamente.
- Uma função booleana é uma função $f : \{0,1\}^n \to \{0,1\}^m$. Uma variável booleana n-ária x é uma variável que pode assumir valores em $\{0,1\}^n$. O valor de x na i-ésima posição será denotado por x_i.
- Uma base Ω é um conjunto finito ou infinito de funções booleanas.

Utilizaremos o termo *função booleana n-ária* para designar funções do tipo $f : \{0,1\}^n \to \{0,1\}$. Vamos dar a seguir uma definição formal de circuito booleano:

Definição 2.1 *Um circuito booleano C sobre uma base Ω é um grafo dirigido acíclico no qual se define o seguinte:*

- *C contém um ou mais vértices com grau de entrada 0, denominados entradas.*
- *C contém um ou mais vértices com grau de saída 0, denominados saídas.*
- *As entradas são associadas a variáveis booleanas ou a constantes 0 ou 1. Os demais vértices são associados a funções de Ω.*

Os vértices de C associados a funções (todos exceto as entradas) são denominados *portas*. A função associada a uma porta v será denotada por f_v.

A seguinte terminologia é padrão no estudo de circuitos:

- O *fan-in* de uma porta é o grau de entrada dessa porta.
- O *fan-out* de uma porta é o grau de saída dessa porta.
- O *tamanho* de um circuito C, denotado por Size(C), é o número de portas de C.

- A *profundidade* de um circuito C, denotada por $\text{Depth}(C)$, é o comprimento do caminho de maior comprimento de C. É fácil ver que um tal caminho tem origem em uma entrada e término em uma saída.

Da definição acima segue naturalmente a idéia de que circuitos realizam funções booleanas. Essa idéia será formalizada a seguir.

Um fato básico sobre grafos dirigidos acíclicos é a possibilidade de organizá-los em *ordem topológica*. Ou seja, dado um grafo dirigido acíclico $G = (V, E)$, é possível numerar seus vértices de modo que para toda aresta (i, j) verifica-se $i < j$. Isso torna legítima a seguinte definição:

Definição 2.2 *Sejam C um circuito e v uma porta. O resultado de v, denotado por* $\text{res}(v)$*, é definido como segue:*

- *Se v é uma entrada,* $\text{res}(v)$ *é o valor da variável ou a constante associada a v.*
- *Se v está associada a um função f e $(u_1, v), (u_2, v), \ldots, (u_{d_{in}(v)}, v)$ são as arestas de entrada de v, o resultado de v é o valor* $\text{res}(v) = f(\text{res}(u_1), \text{res}(u_2), \ldots, \text{res}(u_{d_{in}(v)}))$.

Organizando C em ordem topológica, temos que $\text{res}(i)$ depende do valor de uma variável ou uma constante, se i é uma entrada, ou dos resultados de portas j tais que $j < i$. Não há assim perigo de dependência circular, e $\text{res}(i)$ é uma função bem definida das variáveis de entrada. Se essas variáveis forem numeradas como $x_1, x_2, \ldots, x_n$, então $\text{res}(i)$ pode ser vista como uma função booleana n-ária.

A função computada por um circuito booleano é definida como segue:

Definição 2.3 *Seja C um circuito booleno com n entradas $x_1, x_2, \ldots, x_n$ e m saídas $y_1, y_2, \ldots, y_m$. A função computada por C é a função booleana $f : \{0,1\}^n \to \{0,1\}^m$ definida como segue: a imagem de $(x_1, x_2, \ldots, x_n)$ é $(\text{res}(y_1), \text{res}(y_2), \ldots, \text{res}(y_m))$, atribuindo-se $(x_1, x_2, \ldots, x_n)$ às entradas de C.*

Dada uma entrada x, vamos denotar a saída computada por um circuito C por $C(x)$. Podemos também utilizar um circuito com um número qualquer de saídas para computar funções n-árias, bastando para isso considerar o resultado em uma saída fixada.

Vamos a seguir definir uma classe de circuitos e funções que desempenham um papel importante no estudo de complexidade de circuitos. Primeiramente, definiremos uma ordem parcial para variáveis booleanas como segue: dadas variáveis booleanas n-árias x, y, dizemos que $x \leq y$ se para todo índice i, $1 \leq i \leq n$, $x_i \leq y_i$[1]. Segue a definição dessas funções e circuitos:

Definição 2.4 *Uma função booleana n-ária f tal que $f(x) \leq f(y)$ sempre que $x \leq y$ é denominada monótona. Um circuito C sobre a base $\{\wedge, \vee\}$ (denominada base monótona) é denominado monótono.*

No seguinte teorema, demonstramos que as funções computadas por circuitos monótonos são exatamente as funções monótonas.

[1] Essa ordem é denominada *ordem de Hamming*.

Teorema 2.1 *Uma função booleana f é monótona se e somente se pode ser computada por um circuito monótono.*

Demonstração: Seja f uma função monótona. Seja ϕ a seguinte fórmula, que pode claramente ser realizada por um circuito monótono:

$$\phi = \bigvee_{\{x:f(x)=1\}} \left(\bigwedge_{\{i:x_i=1\}} x_i \right) \tag{1}$$

Seja x tal que $f(x) = 1$. Então $x \in \{x : f(x) = 1\}$ e $\bigwedge_{\{i:x_i=1\}} x_i = 1$, e portanto $\phi(x) = 1$. Seja x tal que $\phi(x) = 1$. Então, pela construção de ϕ, existe x' tal que $f(x') = 1$ e $x_i = 1$ para todo i tal que $x'_i = 1$. Portanto, $x \geq x'$ e, pela monotonicidade de f, $f(x) = 1$. Temos assim que $f = \phi$.

Para demonstrar a recíproca, note primeiramente que a composição de funções monótonas é monótona. De fato, sejam g uma função monótona n-ária, $f_1, f_2, \ldots, f_n$ funções monótonas m-árias, e G a função m-ária definida como segue: $G(x) = g(f_1(x), f_2(x), \ldots, f_n(x))$. Sejam $x, y \in \{0,1\}^m$ tais que $x \geq y$. Então, pela monotonicidade de $f_1, f_2, \ldots, f_n$, $(f_1(x), f_2(x), \ldots, f_n(x)) \geq (f_1(y), f_2(y), \ldots, f_n(y))$ e assim, pela monotonicidade de g, $G(x) \geq G(y)$, e G é monótona.

Como $\wedge, \vee$ são funções monótonas, podemos demonstrar que um circuito monótono C computa uma função monótona por indução em Size(C) como segue:

- Size(C) = 1: então C é apenas uma variável ou uma constante, que são trivialmente funções monótonas.
- Size(C) = $k > 1$: seja v a saída de C e $(u_1, v), (u_2, v) \ldots, (u_{d_{in}(v)}, v)$ as portas de entrada de v. Seja C_{u_i} a parte de C que computa res(u_i), $1 \leq i \leq d_{in}(v)$[2]. Claramente C_{u_i} é um circuito monótono com a saída em u_i. Como Size(C_{u_i}) $< k$, temos, pela hipótese de indução, que res(u_i) é uma função monótona. Basta agora notar que res(v) = $f_v(\text{res}(u_1), \text{res}(u_2), \ldots, \text{res}(u_{d_{in}(v)}))$, ou seja, é uma composição de funções monótonas, e portanto é uma função monótona. Então, C computa uma função monótona.

Por indução, todo circuito monótono computa uma função monótona. ■

Uma base Ω é completa se qualquer função boolena pode ser computada por circuitos sobre Ω. $\Omega = \{\wedge, \vee, \neg\}$ é uma base completa, e será a adotada deste ponto em diante. Vamos também considerar que o *fan-in* de todas as portas $\wedge, \vee$ é igual a 2.

Classes de complexidade para circuitos e circuitos monótonos [4], [3, cap. 20, 22].

3 Medidas de complexidade para circuitos

Como no caso de máquinas de Turing, a noção de complexidade de computação para circuitos é expressa por funções relacionando o tamanho da entrada, ou seja, o número de variáveis boolenas, com propriedades de circuitos que indiquem uma quantidade de recursos necessária para a computação, como tamanho e profundidade. Como dissemos, o objetivo principal deste trabalho é

[2] Se C está em ordem topológica, essa parte inclui u_i e as portas com um rótulo menor que o de u_i.

descrever e estudar medidas de complexidade para a realização de funções booleanas por circuitos. Vamos nesta seção fornecer uma visão geral de algumas dessas medidas.

A noção de máquina de Turing como modelo de computação para a resolução de problemas de decisão parte da codificação de instâncias de problemas em palavras sobre algum alfabeto, que são dadas como entrada para a máquina. No caso de circuitos, a mesma técnica é utilizada. Instâncias são codificadas como vetores de *bits*, ou seja, como palavras sobre o alfabeto $\{0,1\}$, que são fornecidas como entrada para circuitos.

Dada uma palavra $x \in \{0,1\}^*$ e um circuito C com $|x|$ entradas, $C(x)$ é o resultado da computação de C com entrada $(x_1, x_2, \ldots, x_{|x|})$. Com isso, faz sentido falar em linguagens decididas por circuitos quando se considera famílias de circuitos, onde cada circuito decide a pertinência de palavras de tamanho igual ao de sua entrada:

Definição 3.1 *Seja $\mathcal{C} = \{C_n\}$ uma família de circuitos, onde C_i é um circuito com i entradas. Dizemos que $\mathcal{C}$ decide uma linguagem $L \subseteq \{0,1\}^*$ se, para todo $x \in \{0,1\}^*$, $C_{|x|}(x) = 1 \Leftrightarrow x \in L$.*

Uma linguagem corresponde então a uma família de funções booleanas n-árias. Em se tratando de problemas de decisão, cada função da família representa a pertinência de instâncias de um determinado tamanho.

Vamos ilustrar essa discussão com um problema de decisão famoso, que é $\mathcal{NP}$-completo: o problema $\textsc{Clique}_{n,k}$ consiste em, dado um grafo não-dirigido de n vértices G, decidir se G possui um clique de tamanho k. Dada uma instância de tamanho n de $\textsc{Clique}_{n,k}$, ou seja, um grafo não-dirigido G de n vértices, uma possível codificação dessa instância em vetores de *bits* é a seguinte: cada conjunto $\{i,j\}$ de dois vértices é associado a uma variável $x_{i,j}$, cujo valor é 1 se $\{i,j\}$ é uma aresta de G e 0 caso contrário. Considerando os $\binom{n}{2}$ conjuntos de 2 vértices, temos que, para um n fixo, $\textsc{Clique}_{n,k}$ é uma função $\{0,1\}^{\binom{n}{2}} \to \{0,1\}$, tal que $\textsc{Clique}_{n,k}(x) = 1$ se x tem um clique de tamanho k e $\textsc{Clique}_{n,k}(x) = 0$ caso contrário. Portanto, $\textsc{Clique}_{n,k}$ pode ser visto como uma família de funções. Assim, faz sentido entender famílias de circuitos como modelos de computação para a resolução de problemas de decisão.

No estudo de máquinas de Turing, o consumo de recursos para uma máquina em particular é descrito em termos de tempo (número de passos de computação) e espaço (espaço utilizada nas fitas durante a computação) com os quais a máquina opera. Ou seja, uma máquina de Turing opera em tempo $T(n)$, onde T é uma função $\mathbb{N} \to \mathbb{N}$, se o número de passos na computação de uma entrada x é limitado por $T(|x|)$, ocorrendo o análogo para espaço. No modelo de família de circuitos, são consideradas dois recursos: tamanho e profundidade. Assim, dada uma família de circuitos $\mathcal{C} = \{C_n\}$ e funções $S : \mathbb{N} \to \mathbb{N}, D : \mathbb{N} \to \mathbb{N}$, dizemos que $\mathcal{C}$ tem tamanho S se $\text{Size}(C_n) \leq S(n)$ e profundidade D se $\text{Depth}(C_n) \leq D(n)$ para $n \geq 1$.

Essa discussão sugere a definição de medidas de complexidade para funções booleanas (e, portanto, para problemas de decisão utilizando o modelo de circuito). Vamos definir quatro dessas medidas. Essas medidas baseiam-se na seguinte definição:

Definição 3.2 *Dada uma função boolena n-ária f, definimos*

- $\text{Size}(f)$*: tamanho de um circuito de menor tamanho que computa f.*
- $\text{Depth}(f)$*: profundidade de um circuito de menor profundidade que computa f.*
- $\text{monSize}(f)$*: tamanho de um circuito monótono de menor tamanho que computa f.*

- monDepth(f): *profundidade de um circuito monótono de menor profundidade que computa* f.

Dada uma família de funções $\mathcal{F} = \{f_n\}$, vamos denotar por Size($\mathcal{F}$) a função

$$n \to \text{Size}(f_n)$$

para $n \geq 1$. O mesmo se aplica para Depth, monSize e monDepth. Essas são as medidas de complexidade que estudaremos no restante do texto. Uma discussão mais aprofundada sobre essas medidas pode ser vista em [9].

Como é feito comumente, utilizaremos o termo complexidade de um problema como sinônimo de Size para esse problema, valendo o mesmo para o termo complexidade monótona, mas considerando monSize.

Como observação final, note que, utilizando uma base completa, é possível construir famílias de circuitos para decidir qualquer linguagem sobre $\{0, 1\}$, inclusive linguagens não-recursivas. Para que se possa fazer uma relação entre classes de complexidade para famílias de circuitos e para máquinas de Turing é preciso assumir algumas restrições sobre circuitos. Uma discussão sobre esse tema pode ser visto em [3].

Definição 3.3 *Uma família de circuitos* $\mathcal{C} = (C_0, C_1, \dots)$ *é denominada* uniforme *se existe uma Máquina de Turing* N *limitada logaritmicamente em relação ao espaço, de forma que dado como entrada a palavra* 1^n*,* N *escreve na saída a descrição de* C_n*. Dizemos que uma linguagem* L *tem circuitos polinomiais uniformes se existe uma família uniforme de circuitos polinomias que descrevem (ou decidem)* L*.*

Esta definição é razoavelmente forte pois nos permite relacionar circuitos polinomiais com computações polinomiais.

Teorema 3.1 *Uma linguagem* L *tem circuitos polinomiais uniformes* **se e somente se** $L \in \text{P}$

Note que responder a pergunta $\text{P} \stackrel{?}{=} \text{NP}$ é equivalente a decidir se existe algum circuito polinomial uniforme que computa um problema NP-completo.

3.1 Classes de complexidade para circuitos e circuitos monótonos

Podemos então definir algumas classes de complexidades para circuitos.

Definição 3.4 *Para todo* $k \geq 0$*, defina* NC^k *como sendo a classe de linguagens que podem ser decididas por famílias de circuitos com fan-in limitado, tamanho polinomial e profundidade* $\mathcal{O}(\log^k n)$*. Defina* NC *como sendo a união de todas as classes* NC^k*.*

Definição 3.5 *Para todo* $k \geq 0$*, defina* AC^k *como sendo a classe de linguagens que podem ser decididas por famílias de circuitos com fan-in ilimitado, tamanho polinomial e profundidade* $\mathcal{O}(\log^k n)$*. Defina* NC *como sendo a união de todas as classes* AC^k*.*

Um resultado bem interessante mas que não será provado é que AC = NC. Além disso, existe uma relação forte entre famílias de circuitos com fan-in limitado e complexidade de espaço que pode ser expressa pela seguinte afirmação: $\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{NC}^2$.

Analogamente podemos definir as classes P e TC como a classes de linguagens que têm circuitos de tamanho polinomiais e a classe de linguagens que têm circuitos polinomias uniformes respectivamente. Além disso, podemos definir a classe NP para circuitos não determinísticos de tamanho polinomial.

Definição 3.6 *Um circuito C é dito não determinístico, se existe nesse circuito um conjunto v de variáveis que não fazem parte da entrada (tal como uma fita testemunha). Dizemos que o valor de C por x é* 1 **se e somente se** *existe alguma valoração de v tal que $C(x) = 1$.*

Para o caso de circuitos monótonos, as definições são equivalentes e colocamos um m na frente dos nomes das classes. Dessa forma, temos os seguintes resultados (com mono significando o conjunto de todas as famílias de circuitos monótonos):

- $m\mathrm{P} \neq m\mathrm{NP}$
- $m\mathrm{P} \neq \mathrm{P} \cap \mathrm{mono}$
- $m\mathrm{AC}^0 \neq \mathrm{AC}^0 \cap \mathrm{mono}$
- $m\mathrm{NC}^1 \neq m\mathrm{NL}$
- $m\mathrm{TC}^0 \neq m\mathrm{NC}^1$
- $\mathrm{NC}^1 \cap \mathrm{mono} \nsubseteq m\mathrm{NC}$

Diversos problemas como Bisection width, Node Cover e Mochila não são monótonos e portanto não podem ser computados por circuitos monótonos. Mas, outros importantes como Caminho Hamiltoniano e Clique o são e portanto existem circuitos monótonos que os computam. A grande questão é: quão pequenos esses circuitos são?

Nas próximas duas seções, nos dedicaremos a estudar dois resultados muito importantes na área de complexidade de circuito que são o Teorema de Razborov a respeito do problema do Clique e o Teorema de Wigderson a respeito do problema da st-Conexidade.

4 Algumas cotas inferiores para a computação de funções booleanas

Vamos nesta seção exibir uma série de resultados referentes a cotas inferiores para as diversas medidas de complexidade introduzidas na Definição 3.2. Alguns resultados serão apenas enunciados, outros serão demonstrados ou será oferecida uma idéia da demonstração.

4.1 Existe uma função f tal que $\text{Size}(f)$ é exponencial

Vamos nesta seção demonstrar a existência de funções f tais que $\text{Size}(f)$ é exponencial. Isso não será feito exibindo uma função f que possui essa cota inferior. O método que usaremos consiste em determinar um limitante superior para o número total de funções computadas por circuitos com tamanho no máximo s, e com isso demonstrar que, para n grande o suficiente, existem mais funções n-árias do que um s exponencial. Isso implica na existência de funções que exigem circuitos de tamanho superior a s exponencial para serem computadas.

Teorema 4.1 (Shannon, 1949) *Existem n e uma função n-ária f tal que* $\text{Size}(f) \geq \frac{2^n}{n}$.

Demonstração: Dado $s > 0$, seja $N(s)$ o número de funções computadas por circuitos com exatamente s portas com *fan-in* 2, ou seja, portas $\wedge, \vee$. Vamos determinar um limitante superior para $N(s)$.

Observe primeiramente que se d é o número de portas $\neg$ e e o número de portas $\wedge, \vee$ de um circuito, então é possível obter um circuito computando a mesma função com $d \leq e$. Isso é verdade pois, utilizando as leis de DeMorgan, toda porta com as duas entradas negadas pode ser substituida por uma porta com apenas uma entrada negada, ou seja, é possível construir um circuito com no máximo uma de suas entradas negadas, e portanto no máximo uma porta $\neg$ para cada uma das portas $\wedge, \vee$.

Para obter $N(s)$, vamos considerar primeiramente circuitos com portas $\wedge, \vee$ apenas. Considerando apenas a topologia do circuito (ignorando as funções que rotulam as portas), temos que, para cada porta p, o número de possibilidades para a conexão de cada uma de suas entradas é limitado por $(n+2)+(s-1)$, pois temos $(n+2)$ entradas (n variáveis e as constantes $0, 1$), e $s-1$ portas restantes. Considerando as duas entradas de cada porta, obtemos $(n+s+1)^{2s}$ possibilidades. Note agora que, para cada topologia, fazendo uma permutação das portas, obtemos uma mesma topologia que foi considerada separadamente nesse cômputo. Como isso é verdade para cada uma das $s!$ permutações, podemos limitar o número de topologias por $\frac{(n+s+1)^{2s}}{s!}$.

Para cada topologia, podemos rotular cada uma das s portas com duas funções, e temos assim um limitante de $\frac{2^s(n+s+1)^{2s}}{s!}$ para o número de circuitos sem portas $\neg$. Para cada porta $\wedge, \vee$, existem três possibilidades com relação à negação das entradas, a saber, a ausência de negação, ou a negação de uma das duas entradas. Obtemos assim um limitante para $N(s)$:

$$N(s) \leq \frac{6^s(s+n+1)^{2s}}{s!} \leq \frac{6^s(s+2n)^{2s}}{s!}. \tag{2}$$

Como $s! \geq \frac{s^s}{e^s}$ e $(s+2n)^2 \leq s^2 + 8sn^2$, obtemos

$$N(s) \leq a^s \frac{(s^2+8sn^2)^s}{s^s} = a^s(s+8n^2)^s. \tag{3}$$

onde $a = 6e$.

Dado $S > 0$, vamos agora obter um limite superior para a soma de $N(s)$ para $1 \leq s \leq S$, denotada por $M(S)$:

$$M(S) = \sum_{s=1}^{S} a^s(s+8n^2)^s \leq \sum_{s=1}^{S} a^S(S+8n^2)^S = Sa^S(S+8n^2)^S. \tag{4}$$

Tomando $S = \frac{2^n}{n}$ para n suficientemente grande, obtemos:

$$M(S) = \frac{2^n}{n} a^{\frac{2^n}{n}} \left(\frac{2^n}{n} + 8n^2\right)^{\frac{2^n}{n}} < (a^2)^{\frac{2^n}{n}} \left(2\frac{2^n}{n}\right)^{\frac{2^n}{n}} = \left(2a^2\frac{2^n}{n}\right)^{\frac{2^n}{n}} < (2^n)^{\frac{2^n}{n}} = 2^{2^n}. \qquad (5)$$

Note agora que o número de funções booleanas n-árias é 2^{2^n}, e que o número de funções booleanas computadas por circuitos com no máximo S portas é limitado por $M(S)$. Temos assim que, para n suficientemente grande, existe uma função n-ária que não pode ser computada por circuitos de tamanho menor ou igual a $S = \frac{2^n}{n}$, ou seja, $\text{Size}(f) \geq \frac{2^n}{n}$. ■

O teorema acima é um resultado de *existência*, e atesta que a grande maioria das funções booleanas exige circuitos de tamanho exponencial. No entanto, a demonstração de cotas inferiores utilizando circuitos genéricos é difícil. Os melhores resultados obtidos com as técnicas conhecidas atualmente resultam em cotas inferiores lineares e logarítmicas[3], ou seja, não se sabe nada sobre a natureza da maioria das funções booleanas. Esse tópico é no entanto muito interessante, pois a demonstração de cotas inferiores superpolinomiais para problemas em $\mathcal{NP}$ pode ter implicações determinantes na questão $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$.

4.2 Uma função com cota inferior exponencial para complexidade monótona

Conforme discutido na seção anterior, é muito difícil a obtenção de cotas inferiores para funções utilizando circuitos genéricos. Por isso, existe o interesse no estudo de modelos restritos de circuitos. Um deles é o de circuito monótono, que será considerado nesta seção.

Vamos considerar novamente o problema $\textsc{Clique}_{n,k}$. Note que $\textsc{Clique}_{n,k}$ é uma família de funções monótonas, já que a adição de novas arestas não elimina um clique em um grafo. A primeira cota inferior superpolinomial para funções monótonas foi obtida por Razborov em 1985 [7], utilizando esse problema. O resultado que vamos enunciar a seguir é um melhoramento da cota de Razborov, e foi obtido por Alon e Boppana em 1987 [1].

Teorema 4.2 (Alon, Boppana, 1987) *Considere o problema* $\textsc{Clique}_{n,k}$ *com*

$$k = O\left(\left(\frac{n}{\log^2 n}\right)^{\frac{1}{3}}\right).$$

Então

$$\text{monSize}(\textsc{Clique}_{n,k}) = 2^{\Omega(\sqrt{k})}.$$

A demonstração dessa cota pode ser vista em [1, 13, 10].

Outras demonstrações de cotas inferiores para complexidade monótona foram importantes para a obtenção de *gaps* entre complexidade monótona e não-monótona. Atualmente, conhece-se o seguinte resultado:

Teorema 4.3 (Tardos, 1988) *O* gap *entre* Size *e* monSize *é exponencial.*

[3] O melhor resultado conhecido é $\text{Size} \geq 4n$ e $\text{Depth} \geq 3\log n$.

Ou seja, existem problemas cujas complexidades monótona e não-monótona diferem de maneira exponencial em seu crescimento. Tardos exibiu um problema em $\mathcal{P}$ com essa propriedade.

5 Relações entre complexidade de comunicação e profundidade e uma cota inferior para complexidade monótona

Vamos nesta seção obter uma cota inferior justa para a complexidade monótona do problema ST-CONEXIDADE: dado um grafo dirigido e G e vértices s, t de G, ST-CONEXIDADE$(G) = 1$ se e somente se existe um caminho dirigido de s a t.

O objetivo é provar o seguinte teorema:

Teorema 5.1 monDepth(ST-CONEXIDADE) $= \Theta(\log^2(n))$.

O que implica que $m\mathrm{NC}^1 \neq m\mathrm{NC}^2$. Acredita-se que para o caso não monótono esse resultado também valha, mas não se conhece nenhuma prova.

Na prova que apresentaremos são feitas relações entre complexidade de circuitos e complexidade de comunicação. Essas relações serão expressas nos *jogos* que apresentaremos a seguir.

5.1 Jogo 1

Dada $f : \{0,1\}^n \to \{0,1\}$, vamos definir um jogo de comunicação, G_f, como segue: existem dois jogadores, A e B, sendo que A possui um $x \in \{0,1\}^n$ tal que $f(x) = 1$ e B possui um $y \in \{0,1\}^n$ tal que $f(x) = 0$. Ambos os jogadores conhecem f, mas A não conhece y e B não conhece x. O objetivo do jogo é um dos jogadores tomar conhecimento de uma coordenada i tal que $x_i \neq y_i$.

Para determinar essa coordenada, é feita uma troca de *bits* entre os jogadores: a cada momento um jogador envia um *bit* para o outro. Essa troca é feita seguindo um conjunto de regras denominado *protocolo*, que especifica a cada momento qual jogador deve enviar um *bit* e qual é esse *bit*, dependendo apenas dos bits já enviados e de x e y. O protocolo também especifica quando a comunicação termina, e qual o resultado.

A complexidade de comunicação de um protocolo é o maior número de *bits* trocados durante a comunicação, obtido dentre os números de *bits* trocados considerando todas as entradas x, y.

A complexidade de comunicação de G_f, denotada por $C(G_f)$, é o mínimo dentre as complexidades de todos os protocolos que resolvem o jogo.

A relação entre complexidade de circuito e de comunicação está expressa no seguinte lema:

Lema 5.1 $C(G_f) = \mathrm{Depth}(G_f)$.

Demonstração: Vamos admitir nessa demonstração que todas as negações são feitas nas variáveis de entrada, ou seja, todas as portas são $\wedge$ ou $\vee$. Usando as leis de DeMorgan, todo circuito pode ser reduzido a um circuito como esse sem aumento de profundidade.

Vamos demonstrar primeiramente que $C(G_f) \leq \mathrm{Depth}(f)$. Isso será feito por indução em $\mathrm{Depth}(f)$. C irá denotar um circuito com profundidade $\mathrm{Depth}(f)$. Vamos descrever nessa demonstração um protocolo para o jogo com base no circuito.

- $\mathrm{Depth}(f) = 0$: então a saída do circuito é uma das entradas x_i ou $\neg x_i$, e nessa entrada sempre ocorre $x_i \neq y_i$. Não então necessidade de comunicação.

- Depth$(f) > 0$: seja u a saída de C, e v_1, v_2 as entradas de u. Considere os circuitos C_1, C_2 tais que a saída de C_1 é v_1 e a saída de C_2 é v_2, ou seja, $C = C_1 f_u C_2$, e f_1, f_2 as funções computadas por C_1 e C_2 respectivamente. O primeiro *bit* enviado depende de f_u. Temos duas possibilidades:

  - $f_u = \wedge$: como $f(x) = 1$ e $f(y) = 0$, devemos ter que $f_1(x) = f_2(x) = 1$ e $f_1(y) = 0$ ou $f_2(y) = 0$. O primeiro *bit* é então enviado pelo jogador B e especifica qual das funções f_1, f_2 é 0 em y. Digamos que seja a f_1. Então o restante do jogo G_f consiste no jogo G_{f_1}. O mesmo acontece caso o *bit* especifique f_2. Considerando a hipótese de indução, temos que $C(G_f) \leq 1 + \max\{C(G_{f_1}), C(G_{f_2})\} \leq 1 + \max\{\text{Depth}(f_1), \text{Depth}(f_2)\} \leq 1 + (\text{Depth}(f) - 1) = \text{Depth}(f)$.
  - $f_u = \vee$: então $f_1(y) = f_2(y) = 0$ e $f_1(x) = 1$ ou $f_2(x) = 1$. Basta notar que chegamos a uma situação análoga à anterior, se o jogador A enviar um *bit* indicando se f_1 ou f_2 é 1 com x.

  Nas duas situações, $C(G_f) \leq \text{Depth}(f)$, e por indução temos o resultado para qualquer circuito.

Vamos agora demonstrar que $\text{Depth}(f) \leq C(G_f)$. Vamos para isso considerar o seguinte jogo: sejam $U, V \subseteq \{0,1\}^n$ tais que $U \cap V = \emptyset$. A possui um $x \in U$ e B possui um $y \in V$. O objetivo é encontrar i tal que $x_i \neq y_i$. Vamos denotar esse jogo por $G_{U,V}$. Como G_f é o jogo $G_{f^{-1}(1),f^{-1}(0)}$, basta demonstrar que, se $C(G_{U,V}) = d$, então existe uma função $f : \{0,1\}^n \to \{0,1\}$ tal que $f(U) = 1$, $f(V) = 0$ e $\text{Depth}(f) \leq d$. Vamos usar indução em d:

- $d = 0$: então não há comunicação, e isso implica que há uma coordenada i tal que $x_i \neq y_i$ para todo $x \in U, y \in V$. Basta tomar f tal que $f(x) = x_i$ ou $f(x) = \neg x_i$ dependendo do valor de x_i para $x \in U$.

- $d > 0$: temos então um protocolo para $G_{U,V}$ com complexidade de comunicação d. Se o A é o primeiro a enviar um *bit* nesse protocolo, temos que o jogo é particionado em dois jogos $G_{U_0,V}, G_{U_1,V}$, onde U_0, U_1 é uma partição de U, cada um com complexidade de comunição $d-1$. Pela hipótese de indução, existem f_0, f_1 tais que $f_0(U_0) = f_1(U_1) = 1$, $f_0(V) = f_1(B) = 0$ e $\text{Depth}(f_0), \text{Depth}(f_1) \leq d - 1$. Então é fácil ver que $f = f_0 \vee f_1$ é como queríamos. Se B envia o primeiro *bit*, temos uma situação análoga, particionando agora V, obtendo g_0, g_1 tais que $g_0(U) = g_1(U) = 1, g_0(V_0) = g_1(V_1) = 0$, e tomando $f = g_0 \wedge g_1$.

■

5.2 Jogo 2

Vamos denotar por M_f uma versão monótona do jogo definido na seção anterior. Nessa versão, a função f é monótona, e o objetivo é encontrar uma coordenada i tal que $x_i = 1$ e $y_i = 0$.

Utilizando um raciocício semelhante ao da seção anterior, pode-se demonstrar que:

Lema 5.2 $C(M_f) = \text{monDepth}(f)$.

5.3 Jogo 3

Vamos considerar agora uma restrição de M_f com a mesma complexidade de comunicação. Dada uma função f monótona, dizemos que x é um mintermo se $f(x) = 1$ e, para todo $x' < x$, $f(x) = 0$, e uma maxtermo se $f(x) = 0$ e, para todo $x' > x$, $f(x') = 1$. Vamos denotar por $\hat{M}_f$ o jogo M_f onde x é um mintermo e y um maxtermo.

É evidente que $C(\hat{M}_f) \leq C(M_f)$. Um resultado melhor, cuja demonstração não é difícil, é o seguinte:

Lema 5.3 $C(\hat{M}_f) \leq C(M_f)$.

Pela Lema 5.2, segue que

Corolário 5.1 $\text{monDepth}(f) = C(\hat{M}_f)$.

É possível representar fatos sobre conexão em grafos utilizando mintermos e maxtermos. Note que:

- O conjunto de mintermos é o conjunto dos grafos que contém um caminho entre s t e nenhuma outra aresta. De fato, se G é mintermo, ST-CONEXIDADE$(G) = 1$, e portanto há um tal caminho. Se G possui alguma outra aresta e, então $G' = G - e$ é tal que $G' < G$ e ST-CONEXIDADE$(G') = 1$, absurdo, pois G é mintermo. Suponha agora que G possui apenas um caminho entre s e t e nenhuma outra aresta. Então ST-CONEXIDADE$(G) = 1$ e, para todo $G' < G$, não existe um caminho de s a t (pois G' é obtido retirando-se arestas do único caminho de s a t), ou seja, ST-CONEXIDADE$(G') = 0$, e G é portanto um mintermo.
- O conjunto de maxtermos é o conjunto dos grafos G cujos vértices podem ser particionados em conjuntos S, T tais que $s \in S$, $t \in T$ e G contém todas as arestas exceto as que vão de S para T. De fato, suponha que G é um maxtermo. Então ST-CONEXIDADE$(G) = 0$. Seja S o conjunto dos vértices atingíveis a partir de s e T o conjunto dos demais vértices. Note agora que, por G ser maxtermo, não é possível acrescentar uma aresta entre S e T mantendo ST-CONEXIDADE$(G) = 0$. Suponha agora que G é um grafo para o qual existe uma partição S, T como descrito acima. Então ST-CONEXIDADE$(G) = 0$, e o acréscimo de qualquer aresta irá tornar s e t conexos. Portanto, G é maxtermo.

Utilizando esses fatos, veremos que será útil considerar o jogo $\hat{M}_{\text{ST-CONEXIDADE}}$.

5.4 Jogo 4 e uma cota superior para st-Conexidade

Vamos utilizar o resultado da seção anterior para demonstrar uma cota superior para a complexidade de profundidade monótona para ST-CONEXIDADE. Note primeiramente que, pela discussão do final da seção anterior, o seguinte jogo, denominado KW, é uma formulação diferente para $\hat{M}_{\text{ST-CONEXIDADE}}$:

- A possui um caminho de s a t
- B possui uma coloração de vértices C por 0 e 1 tal que $C(s) = 0$ e $C(t) = 1$.

- O objetivo é encontrar uma aresta (u, v) no caminho de A tal que $C(u) = 0$ e $C(v) = 1$.

Pelo Corolário 5.1, para obter uma cota superior para monDepth(ST-Conexidade), basta então obter uma cota superior para a complexidade de comunicação de KW.

Proposição 5.1 $C(\text{KW}) = O(\log^2 n)$.

Demonstração: O protocolo que vamos apresentar consiste na simulação de uma busca binária no caminho de A. Em cada passo, o tamanho do caminho é reduzido pela metade, mantendo-se o invariante de que a origem do caminho tem cor 0 e o destino tem cor 1.

Se o caminho é uma única aresta, o objetivo já foi atingido: o caminho de A é a aresta procurada. Caso contrário, A pergunta a B a cor do vértice no meio de seu caminho. Para isso, A envia esse vértice a um custo de $O(\log n)$ *bits*, e recebe a resposta de B, que é um *bit*. Se a cor é 1, então o invariante é mantido tomando-se a primeira metade do caminho, e o protocolo continua nessa metade. Caso contrário, vale a mesma observação para a segunda metade. Em ambos os casos, o caminho diminui por um fator de 2 a um custo de comunicação de $O(\log n)$ *bits*. Como o caminho pode ter tamanho no máximo n, são executados $O(\log n)$ passos. No total, a complexidade de comunicação é $O(\log^2 n)$, como queríamos. ■

Isso demonstra que monDepth(ST-Conexidade) $= O(\log^2 n)$.

5.5 Jogo Fork

Nesta seção, vamos introduzir um jogo com o qual obteremos a ordem justa de crescimento de monDepth(ST-Conexidade). O jogo se chama Fork. Informalmente, neste jogo dois jogadores escolhem em um conjunto de caminhos dois caminhos com a mesma origem. Dessa forma, existe um vértice no qual esses caminhos se ramificam. O objetivo é determinar esse vértice. Uma formalização de Fork é como segue:

- São dados $n = wl$ vértices divididos em l camadas $l_1, \ldots, l_l$, cada uma com w vértices. São dados três vértices adicionais, denominados s, t_1 e t_2.
- O jogador 1 possui uma seqüência $(x_1, \ldots, x_l)$ de vértices, com $x_i \in l_i$ para $1 \leq i \leq l$. Para simplificar a notação, vamos definir $x_0 = s$ e $x_{l+1} = t_1$.
- O jogador 2 possui uma seqüência $(y_1, \ldots, y_l)$ de vértices, com $y_i \in l_i$ para $1 \leq i \leq l$. Vamos novamente definir $y_0 = s$ e $y_{l+1} = t_1$.
- O objetivo é determinar uma coordenada i tal que $x_i = y_i$ e $x_{i+1} \neq y_{i+1}$.

Há uma relação entre a complexidade de comunicação de Fork e a de KW:

Proposição 5.2 $C(\text{Fork}) \leq C(\text{KW})$.

Demonstração: Considere a seguinte transformação de uma instância de Fork em uma instância de KW:

- o vértice s de KW é o vértice s de Fork, e o vértice t de KW é o vértice t_1 de KW.

- o caminho de A é o caminho do jogador 1 de FORK.
- a coloração possuída por B é como segue: os vértices do caminho do jogador 2 são coloridos com 0, e os demais com 1.

Essa transformação é feita pelos jogadores, e em seguida o protocolo de KW é usado. Para demonstrar que esse protocolo pode ser aplicado, basta notar que a aresta (u, v) obtida em KW é neste caso tal que u pertence aos caminhos dos jogadores 1 e 2 (pertence ao caminho de 2 pois tem cor igual a 0) e v pertence apenas ao caminho de 1 (pois tem cor igual a 1). u então é o ponto de ramificação procurado. ■

5.6 Análise do Jogo Fork

O objetivo desta seção é analizar o jogo FORK como definido anteriormente. Vamos encontrar limitantes superiores e inferiores para a comunicação necessária para resolver o FORK. Dessa forma, completamos a prova do teorema 5.1.

Proposição 5.3 *A complexidade de comunicação do jogo* FORK *é* $\mathcal{O}(\log(w)\log(l))$.

Demonstração: Seja $F_{a,b}$ o jogo FORK de forma que as entradas são da forma $\overline{x} = (x_a, x_{a+1}, \ldots, x_b)$ e $\overline{y} = (y_a, y_{a+1}, \ldots, y_b)$. Consideramos $\overline{x}$ e $\overline{y}$ com duas coordenadas a mais. Uma $(a-1)$-coordenada de forma que $x_{a-1} = y_{a-1} = s$ é o ponto de origem dos caminhos. Uma $(b+1)$-coordenada de forma que $x_{b+1} = t_1$ e $y_{b+1} = t_2$ são os pontos finais dos caminhos.

A idéia básica é fazer busca binária.

Observe que se o comprimento dos caminhos é 1, ou seja, $a = b$, então o problema pode ser resolvido usando $\log(w)$ bits, de acordo com o seguinte protocolo.

- Jogador 1 envia sua entrada (isto requer $\log(w)$ bits)
- Jogador 2 responde com 1 se ele tem a mesma coordenada e 0 caso contrário (um bit)

Se eles têm a mesma coordenada, o ponto de fork é encontrado naquela coordenada e então os caminhos se separam para t_1 e t_2. Senão, o ponto de fork é a origem $s = x_{a-1} = y_{a-1}$.

Se o comprimento dos caminhos é maior que 1, ou seja, $a < b$, então o problema pode ser reducido pela metade usando $\log(w)$ bits (podemos assumir que $\frac{a+b}{2}$ é inteiro).

- Jogador 1 envia o vértice central do caminho $x_{(a+b)}2$ (isto requer $\log(w)$ bits)
- Jogador 2 verifica se eles têm o mesmo vértice central, ou seja, $x_{(a+b)}2 = y_{(a+b)}2$. Se sim, evia 1 senão, envia 0 (um bit)

Se os jogadores têm o mesmo ponto central, então o ponto de fork está entre o meio e o fim. Dessa forma, o jogo é reduzido a $F_{\frac{a+b}{2}+1,b}$. Por outro lado, se o ponto central difere, então existe um ponto de fork entre o início e o meio. Dessa forma, o jogo é reduzido para $F_{a,\frac{a+b}{2}-1}$. Dessa forma, o jogo FORK $F_{1,l}$ pode ser resolvido em $\log(l)$ iterações do protocolo, o que resulta em $\mathcal{O}(\log(w)\log(l))$ bits.

■

Até agora até que foi fácil. Temos que provar que esse limitante é justo.

5.6.1 Um limitante inferior para o jogo Fork

Para construirmos um limitante inferior, vamos considerar jogos que só funcionam com um subconjunto de todas as entradas possíveis. Vamos realizar alguns processos indutivos de forma a estabelecer uma conexão entre jogos de conjuntos diferentes. Vamos considerar dois tipos de transformações:

1. Dado um protocolo que funciona para algum conjunto, geramos um protocolo que que funciona para um conjunto de menor densidade usando um bit a menos de comunicação
2. Dado um protocolo que funciona para algum conjunto, transformamos o mesmo num protocolo que funciona para conjuntos de maior densidade mas com caminhos menores usando a mesma quantidade de comunicação

Pode ser que nessas transformações alguma computação "pesada" seja necessária, mas estamos levando em conta somente a complexidade de comunicação.

Inicialmente vamos definir um subjogo de FORK que somente funciona no subconjunto de entradas possíveis. Dado um subconjunto $S \subseteq [w]^l$, consideramos o jogo FORK^S em que o jogador 1 pega uma entrada $\overline{x} \in S$ e o jogador 2 pega uma entrada $\overline{y} \in S$ e eles têm o mesmo objetivo de encontrar um ponto de fork entre os dois caminhos.

Algumas definições Vamos trabalhar somente com subconjuntos de $[w]^j$ em que a potência j (ou seja, o comprimento do caminho) irá mudar de tempos em tempos. Denominamos w a *largura do jogo*. Durante esta discussão, w vai sempre ser constante.

Definição 5.1 *A* densidade *de um conjunto $S \subseteq [w]^l$ é definida como $\alpha(S) = \frac{|S|}{w^l}$.*

Dado um protocolo que resolve o jogo FORK parcial, estamos interessado na densidade em que tal protocolo funciona. Estamos interessados no protocolo minimal que irá trabalhar com um conjunto de densidade α.

Definição 5.2 *Um protocolo de comunicação será chamado um (α, l)-protocolo se ele trabalha para algum conjunto S de caminhos de comprimento l com densidade $\alpha(S) = \alpha$.*

Definição 5.3 *Denominamos $C(\alpha, l)$ a menor complexidade de comunicação de um (α, l)-protocolo.*

Observe que como só há um único conjunto de densidade 1, um protocolo para FORK é simplesmente um $(1, l)$-protocolo, dessa forma estamos interessados em $C(1, l)$.

Reduzindo a densidade Dado um protocolo que funciona para um conjunto de uma certa densidade, adaptaremos este protocolo de forma que funcione com um subconjunto com metade da densidade. O novo protocolo funcionará com um bit a menos de comunicação que o protocolo original.

Lema 5.4 *Se existe um (α, l)-protocolo que funciona com c bits e $c > 0$, então existe também um $(\frac{\alpha}{2}, l)$-protocolo qye funciona com $c - 1$ bits.*

Pelo lema acima, o melhor protocolo para um conjunto de densidade α irá requerer um bit a mais que o melhor protocolo para um conjunto de densidade $\frac{\alpha}{2}$. Temos então

Corolário 5.2 *Se $C(\alpha, l)$ é diferente de 0, então $C(\alpha, l)$ é maior que $C(\frac{\alpha}{2}, l)$ em pelo menos um bit, ou seja*

$$C(\alpha, l) \geq C(\frac{\alpha}{2}, l) + 1$$

Demonstração: Seja P um protocolo mínimo. A idéia da prova é dividir o conjunto S em dois conjuntos S_0 e S_1 de acordo com o primeiro bit transmitido pelo jogador 1. Construímos então um protocolo que não envia esse primeiro bit mas só irá funcionar com conjuntos de densidade $\frac{\alpha}{2}$.

■

Observe que para aplicarmos o lema acima, precisamos nos certificar que o protocolo consome pelo menos um bit de comunicação.

Lema 5.5 *Qualquer protocolo para um conjunto de densidade maior que $\frac{1}{w}$ requer no mínimo* 1 *bit de comunicação.*

Corolário 5.3 *Para todo l e todo $\alpha > \frac{1}{w}$, $C(\alpha, l) > 0$.*

Demonstração: A idéia da prova é a seguinte. Se não há comunicação, então cada jogador deve decidir sozinho o resultado, mas como a densidade é maior que $\frac{1}{w}$, existe sempre duas possibilidades para cada jogador o que pode levar a erro, logo é necessário pelo menos um bit.

■

Observe que usando esses lemas, podemos obter um limitante inferior de $\Omega(\log(w))$ bits de comunicação para FORK. Enquanto a densidade é maior que $\frac{1}{w}$, o lema 5.5 garante que podemos aplicar o lema 5.4. Desta forma,

$$C(1, l) \geq C(\frac{1}{2}, l) + 1 \geq C(\frac{1}{4}, l) + 2 \geq \cdots \geq C(\frac{1}{w}, l) + \log(2)$$

Mas isso ainda está longe do nosso objetivo.

Reduzindo o comprimento Para alcançarmos o nosso objetivo, precisamos manipular o tamanho do caminho. A ferramenta principal será o seguinte lema de "amplificação" que nos permite, usando um (α, l)-protocolo, construir outro protocolo que funciona num conjunto de caminhos menores (metade do tamanho) mas com densidade maior que α.

Lema 5.6 *Seja $\alpha \geq \frac{12}{w}$. Se existe um (α, l)-protocolo para FORK^S que usa c bits de comunicação, então existe também um $(\frac{\sqrt{\alpha}}{2}, \frac{l}{2})$-protocolo que usa o mesmo número de bits*

Observe que paa α no intervalo $\frac{12}{w} < \alpha < \frac{1}{4}$, ao usarmos esse lema a densidade do conjunto aumenta visto que

$$\frac{12}{w} < \alpha < \frac{1}{4} \Longrightarrow \frac{\sqrt{\alpha}}{2} > \alpha$$

para provar o lema vamos usar o seguinte fato:

Fato 5.1 *Considere uma matriz $n \times n$ de zeros e uns. Seja α a proporção de entradas* 1 *na matriz e seja α_i a proporção de entradas* 1 *na i-ésima linha. Dizemos que uma linha i é* densa *se $\alpha_i \geq \frac{\alpha}{2}$. Vale um dos seguintes casos:*

1. *existe alguma linha com $\alpha_i \geq \sqrt{\frac{\alpha}{2}}$*
2. *o número de linhas densas é no mínimo $\sqrt{\frac{\alpha}{2}}n$*

Demonstração: A prova desse fato pode ser feita facilmente por contradição, afinal se nenhum dos casos vale, então cada coluna tem densidade menor que $\sqrt{\frac{\alpha}{2}}$ e como o caso 2 não vale, então temos menos que $\sqrt{\frac{\alpha}{2}}n$ linhas densas, multiplicando esses dois valores e dividindo por n, obtemos um valor menor que n o que é contradição.

■

Demonstração: (do lema 5.6)

Inicialmente precisamos do sequinte

Definição 5.4 *Para todo $a \in [w]^{\frac{l}{2}}$ definimos $sufixo(a)$ como sendo o conjunto de possíveis suffixos b para a que formam um caminho em S,*

$$sufixo(a) = \{b \in [w]^{\frac{l}{2}} | a \circ b \in S\}$$

Podemos considerar então uma matriz cuja linhas e colunas correspondem a caminhos em $[w]^{\frac{l}{2}}$. Uma posição dessa matriz (a, b) é 1 se $a \circ b \in S$ e 0 caso contrário. A proporção de uns nessa matriz é a densidade de S.

A idéia básica é usar o fato 5.1 para gerarmos um protocolo novo dividindo a matriz pela sua densidade, afinal ou existe um prefixo de caminho em S com um monte de sufixos ou existem vários prefixos de caminhos em S que têm um monte de sufixos.

Para o primeiro caso usamos o conjunto de sufixos como novo conjunto para montarmos o protocolo novo e para o segundo caso, usamos o conjunto de prefixos "carregados" como novo conjunto para montarmos o protocolo novo e para o segundo caso. Dessa forma, o protocolo vai funcionar para caminhos com metade do tamanho e mesma complexidade de comunicação.

A idéia é colorir metade dos vértices de laranja e a outra metade de vermelho em cada "camada" $\frac{l}{2} + 1, \ldots, l$. Se para todo x, o sufixo $b_1(x)$ é colorido de laranja e o sufixo $b_2(x)$ é colorido de vermelho, então o objetivo é alcançado.

A idéia básica é mostrar que tal coloração existe mostrando que a probabilidade dessa coloração dentre todas as colorações é positiva.

Fato 5.2 *Existe uma coloração de todos os vértices de forma que para a maioria dos $a \in S'$ existem sufixos $b_1(a)$ e $b_2(a)$ de forma que*

- $a \circ b_1(a) \in S$ *e* $a \circ b_2(a) \in S$
- *Todos os vértices em $b_1(a)$ são laranjas*
- *todos os vértices em $b_2(a)$ são vermelhos*

Uma forma de colorir os vértices nas camadas $\frac{l}{2} + 1, \ldots, l$ é a seguinte:

- Escolha aleatoriamente $\frac{w}{2}$ caminhos $r_1, \ldots, r_{\frac{w}{w}}$ em $[w]^{\frac{l}{2}}$ e pinte todos os vértices de laranja.
- Nas camadas em que is caminhos pintaram menos que $\frac{w}{2}$ dos vértices, pinte randomicamente vértices de laranja para alcançar $\frac{w}{2}$ vértices laranjas.
- Pinte o resto dos vértices de vermelho

É necessário provar que tal método de coloração é capaz de gerar uma coloração de forma que para qualquer $a \in S'$, existem com altas probabilidades dois sufixos um de cada cor. Vamos omitir esta prova. A densidade do conjunto em que a afirmação anterior vale é no mínimo $\frac{\sqrt{\alpha}}{2} w^{\frac{l}{2}}$.

A idéia é que o jogador 1 concatena a entrada $x \in [w]^{\frac{l}{2}}$ e $b_1(x)$ criando um caminho $x \circ b_1(x)$ em S e o jogador 2 concatena a entrada $y \in [w]^{\frac{l}{2}}$ e $b_2(y)$ criando um caminho $y \circ b_2(y)$ em S. Os jogadores rodam o protocolo e encontrar o ponto de fork desejado.

Dessa forma, consegue-se resolver o problema de densidade $\frac{\sqrt{\alpha}}{2}$ com comprimento $\frac{l}{2}$ utilizando-se a mesma comunicação do protocolo oroginal.

Este resultado é forte o suficiente para concluirmos a seção.

■

Conclusão Vamos finalmente mostrar que qualquer protocolo que resolve Fork usa no mínimo $\Omega(\log(w)\log(l))$ bits de comunicação. Pelo lema 5.5 sabemos que durante todas as transformações nós não encontramos um protocolo que usa só zero bits. Para simplificar os cálculos, assumiremos que l é uma potência de 2.

Inicialmente, observe que $C(1, l) \geq C(\frac{2}{\sqrt{w}}, l)$. Ao realizarmos uma série de $\Omega(\log(w))$ transformações descritas no lema 5.4 (o lema 5.5 nos permite fazer isso), obtemos

$$C(\frac{2}{\sqrt{w}}, l) \geq C(\frac{16}{w}, l) + \Omega(\log(w))$$

Usando o lema 5.6 temos que

$$C(\frac{16}{w}, l) \geq C(\frac{2}{\sqrt{w}}, \frac{l}{2})$$

e portanto

$$C(\frac{2}{\sqrt{w}}, l) \geq C(\frac{2}{\sqrt{w}}, \frac{l}{2}) + \Omega(\log(w))$$

Observe que podemos iterar os últimos dois passos $\log(l)$ vezes. Obtemos então

$$C(1, l) \geq \Omega(\log(l)\log(w))$$

Podemos finalmente concluir usando o resultado acima e a proposição 5.3 que a complexidade de comunicação do jogo Fork é $\Theta(\log(l)\log(w))$. Observe que isto é suficiente para provar o Teorema 5.1 por causa do lema 5.2.

6 Considerações Finais

Como pode-se observar, os principais resultados como os para o Clique ou Fork não estão muito bem provados ou desenvolvidos, principalmente as partes mais importantes.

O problema é que após cada um de nós escrever, separadamente em latex, 61 páginas de exercícios resolvidos da Nami, 22 páginas para Yoshiko e 30 páginas de texto para o Yoshi, NÓS NÃO AGÜENTAMOS MAIS. Foi mal, mas nós estamos cansados e estamos amarelando devido ao excesso de tarefas. Nada pessoal, mas nós precisamos descançar um pouquinho. Apesar dos nossos longos esforços, não conseguimos fazer uma monografia como gostaríamos.

Referências

[1] N. Alon and R. B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, 1988.

[3] O. Goldreich. Introduction to complexity theory. Lecture Notes, 1999. `http://www.wisdom.weizmann.ac.il/~oded/cc-sum.html`.

[4] M. Grigni and M. Sipser. Monotone complexity. In M. S. Peterson, editor, *Boolean Function Complexity*, volume 169 of *Lecture Note Series*, pages 57–75. Cambridge University Press, 1992. `http://www.mathcs.emory.edu/~mic/papers/4.ps`.

[5] M. Grigni and M. Sipser. Monotone separation of logarithmic space from logarithmic depth. *Journal of Computer and System Sciences*, 50:433–437, 1995. `http://www.mathcs.emory.edu/~mic/papers/2b.ps`.

[6] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[7] A. A. Razborov. Lower bounds for the monotone complexity of some boolean functions. *Soviet Math. Dokl.*, 31(2):354–357, 1985.

[8] J. Savage. Introduction to computational complexity. Scribe notes of the course, 2001. `http://www.cs.brown.edu/courses/cs159/syllabus.html`.

[9] J. E. Savage. *The complexity of computing*. John Wiley & Sons, 1976.

[10] V. Vinay and P. R. Subramanya. Computational complexity theory. Scribe notes of the course, 1997. `http://www.imsc.ernet.in/~iarcs/elnotes/cc.ps.gz`.

[11] I. Wegener. *The complexity of boolean functions*. Teubner, Stuttgard, 1987. `http://eccc.uni-trier.de/eccc-local/ECCC-Books/wegener_book_readme.html`.

[12] A. Wigderson and M. Karchmer. Monotone circuits for conectivity require super-logarithmic depth. *Siam J. Disc. Math.*, 3(2):255–265, 1990.

[13] U. Zwick. Boolean circuit complexity. Scribe notes of the course, May 1994. `http://www.math.tau.ac.il/~zwick/scribe-boolean.html`.