UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DAINF - DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

YURI SOCHER BICHIBICHI

LIMITANTE INFERIOR PARA SAT E CONSEQUÊNCIAS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2017

YURI SOCHER BICHIBICHI

LIMITANTE INFERIOR PARA SAT E CONSEQUÊNCIAS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Gustavo A. G. Lugo
UTFPR

Coorientador: Murilo V. G. da Silva
UFPR

CURITIBA
2017

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Coordenação do Curso de Bacharelado em Sistemas de Informação

TERMO DE APROVAÇÃO

“LIMITANTE INFERIOR PARA SAT E CONSEQUÊNCIAS”

por

“Yuri Socher Bichibichi”

Este Trabalho de Conclusão de Curso foi apresentado como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação na Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Curitiba. O(a)(s) aluno(a)(s) foi(ram) arguido(a)(s) pelos membros da Banca de Avaliação abaixo assinados. Após deliberação a Banca de Avaliação considerou o trabalho _____.

_____ **Murilo Vicente Gonçalves da Silva** (Presidente - UTFPR/Curitiba)	_____ **Cesar Augusto Tacla** (Avaliador 1 - UTFPR/Curitiba)
_____ **Nicollas Mocelin Sdroievski** (Avaliador 2 - UFPR/Curitiba)	_____ **Profa. Leyza Baldo Dorini** (Professora Responsável pelo TCC – UTFPR/Curitiba)
_____ **Prof. Leonelo Dell Anhol Almeida** (Coordenador do curso de Bacharelado em Sistemas de Informação – UTFPR/Curitiba)	

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso.”

RESUMO

BICHIBICHI, Yuri. Limitante Inferior para SAT e Consequências. 2017. 39 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Sistemas de Informação, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

Neste trabalho é realizado um levantamento sobre os recentes limitantes inferiores em tempo e espaço para o problema SAT. Também é demonstrada uma prova alternativa para **EXP** $\neq$ **EXPSPACE** $\Rightarrow$ **P** $\neq$ **PSPACE** sem utilizar argumento de preenchimento. Em seguida é feita uma demonstração original de que **PSPACE** $\neq$ **NEXP** $\Rightarrow$ **NP** $\not\subset$ **PolyL**. Este resultado pode ser usado para provar melhores limitantes inferiores para SAT utilizando memória logarítmica e pode auxiliar na busca da solução do problema **P** vs **L**.

Palavras-chave: Complexidade Computacional. SAT. limitante inferior em espaço. PolyL.

ABSTRACT

BICHIBICHI, Yuri. Lower-bound for SAT and Consequences. 2017. 39 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Sistemas de Informação, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

In this work is realized a survey about recents time and space lower-bounds for SAT problem. Also, is demonstrated an alternative proof for **EXP** $\neq$ **EXPSPACE** $\Rightarrow$ **P** $\neq$ **PSPACE** without using padding argument. After that is shown an original demonstration that **PSPACE** $\neq$ **NEXP** $\Rightarrow$ **NP** $\not\subset$ **PolyL**. This result can be used to prove betters lower-bounds for SAT with memory logarithmic and can help the search for the solution of the problem **P** vs **L**.

Keywords: Computational Complexity. SAT. space lower-bound. PolyL.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

MT	Máquina de Turing
MTND	Máquina de Turing Não-Deteminística
ID	configuração de uma Máquina de Turing $\mathcal{U}$ Máquina de Turing Universal
DTIME	classe de problemas decidíveis por uma MT determinística em certo tempo
NTIME	classe de problemas decidíveis por uma MT não-determinística em certo tempo
DTISP	classe de problemas decidíveis por uma MT determinística em certo tempo e espaço
NTISP	classe de problemas decidíveis por uma MT não-determinística em certo tempo e espaço
L	classe de problemas decidíveis por uma MT determinística em espaço logarítmico
PolyL	classe de problemas decidíveis por uma MT determinística em espaço polilogarítmico
NL	classe de problemas decidíveis por uma MT não-determinística em espaço logarítmico
P	classe de problemas decidíveis por uma MT determinística em tempo polinomial
NP	classe de problemas decidíveis por uma MT não-determinística em tempo polinomial
coNP	classe de problemas decidíveis por uma MT co-não-determinística em tempo polinomial
PSPACE	classe de problemas decidíveis por uma MT determinística em espaço polinomial
poly	função que retorna um polinômio da entrada
exp	função que retorna uma função exponencial da entrada
polylog	função que retorna um polinômio de log da entrada (ex.: $\log^2 n$ é polylog de n)

EXP	classe de problemas decidíveis por uma MT determinística em tempo exponencial
EXPSPACE	classe de problemas decidíveis por uma MT determinística em espaço exponencial 2-
EXPTIME	classe de problemas decidíveis por uma MT determinística em tempo "duplamente exponencial"
CNF	forma normal conjuntiva para fórmulas boolianas
SAT	problema de decisão para satisfatibilidade de fórmulas boolianas em CNF
QBF	fórmula booliana quantificada
TQBF	problema de determinar se um QBF é verdadeiro
PH	hierarquia polinomial

LISTA DE SÍMBOLOS

$\mathcal{U}$	Máquina de Turing Universal
O	limite assintótico superior (inclusive)
o	limite assintótico superior (não-inclusive)
Ω	limite assintótico inferior (inclusive)
ω	limite assintótico inferior (não-inclusive)
Θ	limite assintótico inferior e superior simultaneamente

SUMÁRIO

1 INTRODUÇÃO

Existem problemas que são decidíveis em tempo polinomial e formam na classe **P**. Existem problemas que verificáveis em tempo polinomial e formam uma classe maior denominada **NP**. A maioria dos pesquisadores em Complexidade Computacional acredita que existem problemas que estão na classe **NP** mas que não estão na classe **P**, ou seja, **P** $\neq$ **NP**. De acordo com Fortnow (2009), o problema **P** vs **NP** é considerado o maior problema em aberto na área da Ciência da Computação e um dos maiores problemas da matemática.

Um exemplo de problema que está em **NP** e que acredita-se não estar em **P** é o problema da satisfatibilidade (SAT). Provar que SAT não está em **P** seria suficiente para provar que **P** $\neq$ **NP**. Uma abordagem possível para provar que SAT não está em **P** é provar um limitante inferior em tempo para o problema que seja alto o suficiente (por exemplo, para uma entrada de tamanho n, provar que o problema SAT não pode ser resolvido em menos de 2^n passos). Outra abordagem que contribui com o problema é provar um limitante inferior em espaço (por exemplo, para uma entrada de tamanho n, provar que SAT usa no mínimo $\sqrt{n}$ blocos de memória; um resultado como este diminui a possibilidade de que SAT esteja em **P**).

Atualmente ainda não há nenhuma prova de que SAT seja sequer superlinear em tempo, i.e., para uma entrada de tamanho n e uma constante c, são necessários um número de passos superior a cn. Também não há nenhuma prova de que SAT precise de mais memória que logarítmica. Neste trabalho são feitas algumas provas alternativas para enunciados conhecidos e é provado um teorema relacionado ao limite inferior em espaço para SAT.

1.1 Justificativa

Como dito anteriormente, o problema **P** vs **NP** é considerado o maior problema em aberto na área da Ciência da Computação e um dos maiores problemas da matemática (FORTNOW, 2009). Também está na lista dos problemas do milênio (INSTITUTE, 2018) e dos problemas de Smale (SMALE, 1998).

1.2 Objetivo Geral

O objetivo geral do trabalho é provar enunciados conhecidos de maneiras alternativas e fazer uma contribuição relacionada ao limitante inferior em espaço para SAT.

1.3 Objetivos Específicos

Abaixo estão listados os objetivos específicos:

- provar que **NEXP** $\neq$ **PSPACE** implica que SAT não pode ser decidido em espaço polilogarítmico;

1.4 Estrutura/Organização

O trabalho está organizado como descrito: o Capítulo 2 contém o referencial teórico utilizado, em especial a definição de SAT na Seção 2.6 e o Teorema da Hierarquia de Tempo na Seção 2.5.

Uma descrição dos trabalhos relacionados aos assuntos tratados aqui podem ser encontrados no Capítulo 3. Também é apresentado o limitante inferior em tempo e espaço para SAT de Arora e Barak (2009), que é uma versão simplificada do limitante originalmente obtido por Fortnow (2000).

No Capítulo 4 são apresentados os resultados, os quais estão subdivididos entre provas alternativas para resultados conhecidos e resultados originais (no caso, o teorema relacionado ao limitante inferior em espaço para SAT).

Finalmente, as conclusões do trabalho e sugestões para trabalhos futuros estão disponíveis no Capítulo 5. Um apêndice está disponível no final do documento para definições mais básicas que as encontradas no Capítulo 2.

2 Levantamento Bibliográfico

Neste capítulos são expostos conceitos fundamentais utilizados no decorrer deste trabalho. Todas as definições, exceto quando indicado o contrário, são baseadas em Arora e Barak (2009). Neste trabalho é assumido familiaridade do leitor com a notação e definições usuais de Teoria de Grafos. Estas notações e definições podem ser encontradas no Apêndice A.

Neste capítulo é inicialmente são feitas definições básicas, como notação assintótica, Máquinas de Turing e problemas de decisão (Seções 2.1-2.3). Em seguida são definidas variações do modelo clássico de Máquina de Turing, seguida pela definição de Máquina de Turing Universal e a definição de configuração de uma Máquina de Turing. Na Seção 2.4 são introduzidas as classes de complexidade usadas neste trabalho. Posteriormente é definido e provado o Teorema da Hierarquia de Tempo, o teorema que prova que existem limitantes acima de tempo linear (Seção 2.5; note que uma vez que este trabalho busca tal limitante, este teorema, que garante a existência de tal limitante, é fundamental). Depois são feitas as definições de fórmula booleana, problema de satisfatibilidade, tautologia e fórmula booleana quantificada (Seção 2.6); estas definições são usadas na seção Seção 2.7 na demonstração de resultados relevantes da área. Finalmente, a última seção aborda Hierarquia de Tempo e Máquina de Turing Alternante.

2.1 Definições Básicas e Notação

Nesta seção são apresentadas as definições e notações utilizadas em todo o trabalho.

2.1.1 Notação Assintótica

Geralmente a eficiência de algoritmos é definida de acordo com o número de operações executadas em função do tamanho da entrada, ou seja, a eficiência de um algoritmo é capturada por uma função t recebendo uma entrada n, onde $n \in \mathbb{N}$ e $t(n)$ define o máximo de operações do algoritmo para toda entrada de tamanho n. No entanto t depende muito da implementação do algoritmo; para ignorar estes detalhes é utilizada a notação assintótica a qual é definida a seguir.

Definição 2.1.1 (Notação Assintótica)**.** Para f e g, funções $\mathbb{N} \to \mathbb{N}$, é definido:

1. $f = O(g)$ se existe uma constante c tal que $f(n) \leq c \cdot g(n)$ para todo n grande o suficiente;
2. $f = \Omega(g)$ se $g = O(f)$;
3. $f = \Theta(g)$ se $f = O(g)$ e $g = O(f)$;
4. $f = o(g)$ se para todo $\epsilon > 0$, $f(n) \leq \epsilon \cdot g(n)$ para todo n grande o suficiente, e;
5. $f = \omega(g)$ se $g = o(f)$.

As seguintes classes de função são relevantes neste trabalho:

Definição 2.1.2 (Função sublinear)**.** Uma função $f(n)$ é sublinear se $f(n) = o(n)$.

Definição 2.1.3 (Função superlinear)**.** Uma função $f(n)$ é superlinear se $f(n) = \omega(n)$.

Definição 2.1.4 (Função subpolinomial)**.** Uma função $f(n)$ é subpolinomial se $f(n) = O(n^{o(1)})$.

Definição 2.1.5 (Função polilogarítmica)**.** Uma função $f(n)$ é polilogarítmica se f é polinômio de $\log n$, ou seja, para a_k e k constantes, $f(n) = a_k(\log n)^k + ... + a_1(\log n) + a_0$.

Seja ϵ maior que 0. Exemplos de funções sublineares são: $n^{1-\epsilon}$, $n/\log n$. Exemplos de funções superlineares são: $n^{1+\epsilon}, n \log n$. Exemplos de funções subpolinomiais são: $\log^2 n, 2^{\sqrt{\log n}}$. Exemplos de funções polilogarítmicas são: $\log^2 n$, $(\log^3 n) + (\log n)$.

2.1.2 Máquina de Turing

Um modelo clássico e concreto de computação é a Máquina de Turing (MT), fundamental neste trabalho. De acordo com a Tese de Church-Turing, este modelo define o conceito matemático de funções computáveis, assim como é equivalente a qualquer dispositivo computacional fisicamente realizável. Por isso é considerado suficiente para muitos resultados em complexidade computacional. A seguir é feita uma definição formal de MT:

Definição 2.1.6 (Máquina de Turing)**.** Uma MT é definida por uma tupla (Γ, Q, δ) onde:

1. Γ é o alfabeto da fita, geralmente é o conjunto $\{0, 1, \square, \triangleright\}$, onde $\square$ é o simbolo especial branco e $\triangleright$ representa o inicio de uma fita;
2. Q é o conjunto de estados em que uma MT pode estar; assume-se que $\{q_{start}, q_{halt}\} \subseteq Q$, e;
3. $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$, onde $k \geq 2$, define a função de transição de uma MT.

Se a máquina está em um estado $q \in Q$ e; $(\sigma_1, \sigma_2, ..., \sigma_k)$ são os símbolos sendo lidos nas k fitas e; $\delta(q, (\sigma_1, ..., \sigma_k)) = (q', (\sigma'_2, ..., \sigma'_k), z)$, onde $z \in \{L, S, R\}^k$; então, no próximo passo:

1. os símbolos σ da máquina em suas últimas $k - 1$ fitas são trocados pelos símbolos σ' e;
2. a máquina estará no estado q' e;
3. seus k cabeçotes irão se mover para a esquerda (L), direita (D) ou manter a posição (S) de acordo com z.

Se a máquina tentar mover algum cabeçote para esquerda quando não existir mais posições à esquerda, então o cabeçote mantém a posição.

A fita de entrada contém $\triangleright$ em seu início seguido pela entrada da máquina e o restante preenchido com $\square$. O restante das fitas contém o símbolo $\triangleright$ em seu início e têm o restante preenchido com $\square$. Todas os cabeçotes são inicializadas na primeira posição e a máquina começa no estado q_{start}. Cada passo computacional é realizado utilizando a função δ como

descrito anteriormente, porém a máquina para ao alcançar o estado q_{halt}. Em complexidade computacional, geralmente há o interesse apenas por máquinas que param após finitos passos para qualquer entrada.

O resultado da computação de MT M utilizando uma entrada $x \in \{0,1\}^*$ é denotado por $M(x)$. É definido que M computa uma função $f : \{0,1\}^* \to \{0,1\}^*$ se $\forall x\ M(x) = f(x)$. Caso M resolva um problema de decisão (ou seja, cuja resposta é SIM ou NÃO), tipicamente a função é definida como sendo $f : \{0,1\}^* \Rightarrow \{0,1\}$.

É definido que M é tempo $t(n)$ se, para toda entrada de tamanho n, sejam necessários até $t(n)$ passos. Também é definido que M é espaço $s(n)$ se, para toda entrada de tamanho n, sejam necessários até $s(n)$ posições das fitas de trabalho.

2.1.3 Função tempo e espaço-construtível

Em complexidade computacional, funções que delimitam o tempo de execução de MT's geralmente são *tempo-construíveis* e *espaço-construíveis* , pois do contrário podem aparecer resultados anômalos.

Definição 2.1.7 (Função tempo-construível)**.** A função $t : \mathbb{N} \to \mathbb{N}$ é *tempo-construível* se $t(n) \geq n$ e existir uma MT M que compute a função $x \mapsto \llcorner t(|x|) \lrcorner$ em tempo $t(n)$ ($\llcorner t(|x|) \lrcorner$ é a representação em binário de $t(|x|)$). Exemplos de funções tempo-construíveis são n, $n \log n$, n^2, 2^n. A restrição $t(n) \geq n$ é para dar tempo do algoritmo ler a entrada.

Definição 2.1.8 (Função espaço-construível)**.** A função $s : \mathbb{N} \to \mathbb{N}$ é *espaço-construível* se $s(n) \geq \log n$ e existir uma MT M que compute a função $x \mapsto \llcorner s(|x|) \lrcorner$ em espaço $s(n)$. A restrição $s(n) \geq \log n$ é para que o algoritmo tenha espaço para salvar o índice da posição de memória que está sendo lida.

2.1.4 Problemas de decisão

Um problema de decisão é uma linguagem $L \subseteq \{0,1\}^*$. Apesar de ser um conjunto de *strings* apenas de 0's e 1's, outros tipos de dados podem estar contidos no conjunto quando codificados em binário (por exemplo, L pode conter *strings* em binário que representam grafos). É dito que uma MT M decide L se, $\forall x \in \{0,1\}^*, x \in L \Leftrightarrow M(x) = 1$.

2.2 Variações da Máquina de Turing

Nesta seção são definidas variações da Máquina de Turing, entre elas: Máquina de Turing com Acesso Aleatório, a qual se aproxima do modelo usado em computadores modernos, e; Máquina de Turing Não-Determinística, um modelo que não se propõe a ser fisicamente realizável, mas que é uma ferramenta matemática extremamente útil. Note que, para a Máquina de Turing com Acesso Aleatório, é conhecido que a diferença de desempenho entre este modelo e a MT clássica é apenas polinomial.

Definição 2.2.1 (Máquina de Turing com Acesso Aleatório (RAM))**.** Uma máquina R é RAM se o acesso a qualquer posição de qualquer fita seja feito em tempo $O(1)$ independente da última posição acessada.

Definição 2.2.2 (Máquina de Turing Não-Determinística (MTND))**.** É semelhante a MT, porém tem duas funções de transição δ_0 e δ_1 e um estado especial q_{accept}. Quando uma MTND M computa uma função ela escolhe qual função de transição usará. Para todo input x, $M(x) = 1$ se existe uma sequência de escolhas (não-determinísticas) que fazem M alcançar o estado q_{accept}. Caso não seja possível alcançar este estado, então M utiliza qualquer sequência de escolhas e alcança o estado q_{halt}. Neste caso $M(x) = 0$.

Definição 2.2.3 (Máquina de Turing Co-Não-Determinística)**.** É semelhante a MTND, porém a máquina retorna 1 se e somente se ambas as funções de transição δ_0 e δ_1 alcançam o estado q_{accept}.

2.3 Máquina de Turing Universal e Configuração

Nesta seção é feita a definição de Máquina Universal, a qual é usada em vários teoremas neste trabalho, em especial o Teorema da Hierarquia de Tempo (Seção 2.5), e; é feita a definição de Configuração de uma dada Máquina de Turing, que é usada na definição de Grafo de Configurações (Definição 2.7.1).

2.3.1 Máquina de Turing Universal

Máquinas de Turing podem ser descritas por *strings* que codificam sua tabela de transição. Neste trabalho é assumido que toda *string* em $\{0,1\}^*$ representa alguma MT (caso a representação seja inválida então é definido que *string* representa uma MT canônica com apenas 1 estado e que rejeita qualquer entrada) e que toda MT é representável por infinitas maneiras (i.e., dada a descrição de uma máquina, sempre é possível criar outra definição equivalente que seja maior). Para uma MT M, denota-se por $\llcorner M \lrcorner$ a representação de M na forma de *string*. Para uma *string* x, denota-se por M_x a máquina representada por x. Uma Máquina de Turing Universal, definida por $\mathcal{U}$, é uma máquina que simula outra máquina dado sua descrição.

Teorema 2.3.1 (Máquina de Turing Universal Eficiente (ARORA; BARAK, 2009))**.** *Existe uma MT $\mathcal{U}$ tal que $\forall x, \alpha \in \{0,1\}^*, \mathcal{U}(x, \alpha) = M_\alpha(x)$, onde M_α é a MT representada pela string α. Para t o número de passos executados por $M_\alpha(x)$, então $\mathcal{U}(x, \alpha)$ para em até $ct \log t$ passos, onde c depende apenas do número do tamanho alfabeto de M_α, do número de fitas e do número de estados.*

2.3.2 Configuração

Definição 2.3.1 (Configuração ou Descrição Instantânea (ID))**.** A *configuração* ou *descrição instantânea* de uma Máquina de Turing é uma *string* que contém o estado da máquina, o conteúdo das fitas e a posição do cabeçote em cada fita. É evidente que se a complexidade de espaço de uma MT é $s(n)$, então o tamanho da configuração é $\Theta(s(n))$.

2.4 Problemas de decisão e Linguagens

Nesta seção são definidas algumas das classes de complexidade usadas neste trabalho. Classes de complexidade agrupam problemas de decisão de acordo com o tempo e o espaço mínimos necessários para que sejam decididas (note que certos problemas podem estar ou não na classe dependendo do modelo de computação utilizado).

Definição 2.4.1 (Computação limitada por espaço)**.** Seja $s : \mathbb{N} \to \mathbb{N}$ e $L \subseteq \{0,1\}^*$. É dito que $L \in$ **SPACE**$(s(n))$ se há uma constante c e uma máquina determinística M que decida L tal que até $c \cdot s(n)$ posições das fitas de trabalho de M (i.e., todas as fitas menos a fita de entrada) são acessadas durante a computação de qualquer entrada de tamanho n. De maneira semelhante, é dito que $L \in$ **NSPACE**$(s(n))$ se há uma máquina não-determinística M que decide L tal que não use mais que $c \cdot s(n)$ posições não-brancas das fitas de trabalho para toda entrada de tamanho n.

A seguir são definidas outras classes usadas neste trabalho, algumas das quais estão representadas na Figura 1. Para uma entrada de tamanho n:

1. **DTIME**$[t(n)]$ e **NTIME**$[t(n)]$ são classes de complexidade que, para c constante, todo problema da classe pode ser resolvido em $c \cdot t(n)$ passos por um algoritmo determinístico e não-determinístico respectivamente;
2. **DTISP**$[t(s), s(n)]$ é a classe de problemas que, para c e c' constantes, podem ser resolvidos por uma máquina determinística que utiliza ao mesmo tempo $c \cdot t(s)$ passos e espaço $c' \cdot s(n)$. A **NTISP**$[t(s), s(n)]$ é o equivalente não-determinístico;
3. **L** é a classe de problemas que, para uma constante c, podem ser resolvidos utilizando memória de tamanho $c \log n$ (note que $\log n^c = c \log n$);
4. **P** $= \bigcup_{c>0}$ **DTIME**$[n^c]$;
5. **NP** $= \bigcup_{c>0}$ **NTIME**$[n^c]$. Definindo de outra maneira, uma linguagem $L \subseteq \{0,1\}^*$ está em **NP** se existe uma função polinomial $p : \mathbb{N} \to \mathbb{N}$ e uma Máquina de Turing M de tempo polinomial (chamada de *verificador* de L) tal que, para todo $x \in \{0,1\}^*$, $x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}$ tal que $M(x,u) = 1$. Se $x \in L$ e $u \in \{0,1\}^{p(|x|)}$ tal que $M(x,u) = 1$, então u é chamado de *certificado* para x (com relação a uma linguagem L e máquina M);
6. Uma linguagem $L \subseteq \{0,1\}^*$ está em **coNP** se existe uma função polinomial $p : \mathbb{N} \to \mathbb{N}$ e uma Máquina de Turing M de tempo polinomial tal que, para todo $x \in \{0,1\}^*$,

$x \in L \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)}$ tal que $M(x,u) = 1$;

7. **PSPACE** $= \bigcup_{c>0}$ **SPACE**$[n^c]$;
8. **EXP** $= \bigcup_{c>0}$ **DTIME**$[2^{n^c}]$;
9. **EXPSPACE** $= \bigcup_{k>0}$ **SPACE**$[2^{n^k}]$;

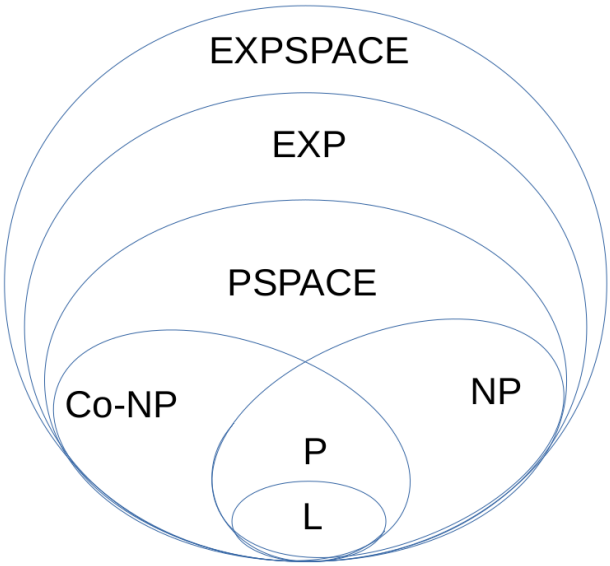

Figura 1 – Gráfico mostrando relações entre classes de forma resumida. Fonte: Autoria Própria.

Definição 2.4.2 (Redução e completude)**.** Uma linguagem $L \subseteq \{0,1\}^*$ é *Karp redutível em tempo-polinomial* para uma linguagem $L' \subseteq \{0,1\}^*$, denotado por $L \leq_p L'$, se existir uma função computável em tempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$, $x \in L$ se e somente se $f(x) \in L'$.

Define-se que L' é **NP**-difícil se $L \leq_p L'$ para todo $L \in$ **NP**. Define-se que L' é **NP**-completa se L' é **NP**-difícil e $L' \in$ **NP**. Uma linguagem é **coNP**-completa se ela estiver em **coNP** e toda linguagem em **coNP** é redutível em tempo-polinomial para ela. De forma semelhante, uma linguagem L' é **PSPACE**-difícil se para toda $L \in$ **PSPACE**, $L \leq_q L'$. Se $L' \in$ **PSPACE** então L' é **PSPACE**-completa.

2.5 Teorema da Hierarquia de Tempo e Espaço

Nesta seção é enunciado e provado o Teorema da Hierarquia de Tempo, provavelmente o teorema mais importante deste capítulo, pois define que existem problemas não podem ser resolvidos em tempo menor que superlinear. De maneira informal, o Teorema da Hierarquia de Tempo diz que quanto mais tempo uma máquina tem a disposição, mais problemas ela pode resolver. Ressalva-se que a versão do teorema provado aqui é para o modelo de Máquina de Turing, porém é possível provar teoremas correlatos usando outros modelos (no caso da RAM, o *overhead* fica maior).

Teorema 2.5.1. *Sejam f, g funções tempo-construtível tal que $f(n)\log f(n) = o(g(n))$, então* **DTIME**$(f(n)) \subsetneq$ **DTIME**$(g(n))$

Demonstração. Para uma função tempo-construtível g, seja uma máquina D como se segue: *"para uma entrada x, executa por $g(|x|)$ passos a máquina Universal $\mathcal{U}$ para simular M_x recebendo x. Se $\mathcal{U}$ retorna um bit $b \in \{0,1\}$ nesse tempo, então D retorna o bit contrário, ou seja, $1-b$. Senão D retorna 0"*. Aqui, relembrando, M_x é a máquina representada pela *string* x.

Por definição D termina de executar em $cg(n)$ passos, então a linguagem decidia por D está em **DTIME**$(g(n))$. Seja L a linguagem decidida por D. Afirma-se que $L \notin$ **DTIME**$(f(n))$. Suponha por absurdo que, para uma constante c', exista uma máquina M que execute em $c'f(n)$ e $M(x) = D(x)$.

O tempo para simular M é de até $c''c'f(n)\log(c'f(n)) = c''c'g(n)^{1-o(1)}$, onde c'' é uma constante que depende do tamanho do alfabeto e número de estados de M. Para um n grande o suficiente, $cg(n) > c''c'g(n)^{1-o(1)}$; suponha que o menor valor de n para deixar a inequação verdadeira seja n_0. Seja x_M uma *string* que represente M tal que o tamanho de x_M seja pelo menos n_0. Então $D(x_M)$ irá obter o resultado $b = M(x_M)$ em $c''c'g(n)^{1-o(1)}$ passos; porém, pela definição de D, $D(x_M) = 1 - b \neq M(x_M)$, o que é uma contradição. □

De maneira análoga ao Teorema de Hierarquia de Tempo, é possível fazer o mesmo para espaço:

Teorema 2.5.2. *Sejam f, g funções tempo-construtível tal que $f(n) = o(g(n))$, então* **DSPACE**$(f(n)) \subsetneq$ **DSPACE**$(g(n))$

A prova para o Teorema de Hierarquia de Espaço é semelhante ao análogo em tempo, porém no caso de Máquinas de Turing o *overhead* é apenas constante. Mais detalhes sobre a prova em Arora e Barak (2009).

2.6 Fórmulas Booleanas Quantificadas

Nesta seção são definidos temas básicos relacionados a fórmulas booleanas, necessários para as definições de problema de satisfatibilidade e tautologia. Estes por sua vez são necessários para a definição de fórmulas booleanas quantificadas, definição esta necessária necessária para a próxima seção. Mais informações sobre fórmulas booleanas podem ser encontradas em Papadimitriou (1994).

2.6.1 Fórmulas Booleanas

Uma *fórmula booleana* sobre as variáveis $u_1, ..., u_n$ consiste de variáveis e de operadores lógicos "E" ($\wedge$), "OU" ($\vee$) e "NÃO" ($\neg$). Por exemplo, $(u_1 \wedge u_2) \vee (u_2 \wedge u_3) \vee (u_3 \wedge u_1)$ é uma fórmula booleana. Se φ é uma fórmula booleana sobre as variáveis $u_1, ..., u_n$ e $z \in 0, 1^n$, então

$\varphi(z)$ denota o valor de φ quando as variáveis de φ recebem os valores de z (onde 1 identifica "Verdadeiro" e 0 identifica "Falso"). A fórmula φ é *satisfazível* se existe uma valoração z tal que $\varphi(z)$ é "Verdadeiro". Senão, afirma-se que φ é *não-satisfazível*. Se para toda valoração z, $\varphi(z)$ é verdadeiro, então a fórmula é uma *tautologia*.

Uma fórmula booleana sobre as variáveis $u_1, ..., u_n$ está em *forma normal conjuntiva* (CNF) se segue a forma:

$$\bigwedge_i \Big(\bigvee_j v_{i_j} \Big)$$

onde cada v_{i_j} é uma variável u_k ou a negação $\neg u_k$. Os termos v_{i_j} são chamados de *literais* da fórmula e os termos $(\vee_j v_{i_j})$ são chamados de *cláusulas*. De acordo com o teorema de Cook-Levin, SAT é um problema **NP**-completo. O problema de decidir se uma fórmula é tautologia é um problema **coNP**-completo.

2.6.2 Fórmulas Booleanas Quantificadas

Uma fórmula booleana quantificada (QBF) é uma fórmula na forma

$$Q_1 x_1 Q_2 x_2 \ ... \ Q_n x_n \varphi(x_1, x_2, ..., x_n)$$

onde $Q_i \in \{\exists, \forall\}$, $x_i \in \{0, 1\}$ e φ é uma fórmula booleana (não quantificada). Note que $\exists x_1, x_2, ..., x_n \varphi(x_1, x_2, ..., x_n)$ é equivalente ao problema SAT e que $\forall x_1, x_2, ..., x_n \varphi(x_1, x_2, ..., x_n)$ é equivalente ao problema de tautologia. O problema de decidir se uma QBF é verdadeira é chamada de TQBF.

2.7 O problema TQBF e Complexidade de Espaço

Esta seção começa definindo o que é o grafo de configurações de uma Máquina de Turing, seguida por um lema que associa QBF às Máquinas de Turing, e posteriormente são provados que TQBF é **PSPACE**-completo e o teorema de Savitch, o qual relaciona as classes **SPACE** e **NSPACE**.

Definição 2.7.1 (Grafo de configurações)**.** Seja uma entrada x de tamanho n e uma máquina M que sempre para tal que a complexidade de espaço seja $s(n)$. O grafo $G_{M,x}$, chamado de grafo de configurações, é um grafo direcionado onde um vértice corresponde a entrada x e o restante dos vértices correspondem a todas as configurações de M processando x (o grafo contém apenas configurações possíveis de se alcançar). O grafo tem uma aresta partindo de C até C' se C' é alcançável por C depois de um passo. Alterando M para apagar todas as fitas de trabalho antes de parar, então existe uma única configuração C_{accept} onde M para e retorna 1. Isto significa que M aceita a entrada x se e somente se existe um caminho em $G_{M,x}$ de C_{start} até C_{accept}.

Observe que cada vértice em $G_{M,x}$ pode ser descrito usando $cs(n)$ bits (onde c depende do tamanho do alfabeto e do número de fitas) e $G_{M,x}$ tem no máximo $2^{cs(n)}$ vértices.

Lema 2.7.1. *Seja M uma MT que utiliza espaço $s(n)$, uma entrada x e um grafo de configurações correspondente $G_{M,x}$. Existe uma fórmula CNF $\varphi_{M,x}$ de tamanho $O(s(n))$ tal que, para duas strings C e C', $\varphi_{M,x}(C, C') = 1$ se e somente se C e C' são adjacentes em $G_{M,x}$.*

A prova para o lema anterior está em Arora e Barak (2009). Faz-se a ressalva de que o teorema também funciona para RAM, no entanto neste caso o tamanho de cada fórmula $\varphi_{M,x}$ é $O(\text{poly}(s(n)))$.

Lema 2.7.2. *Seja L uma linguagem em* **PSPACE**, *então $L \leq_p$ TQBF.*

Demonstração. Seja M uma máquina que decide L em espaço $s(n)$ e seja x uma *string* e $n = |x|$. Esta prova envolve construir uma QBF a partir de x de tamanho $O(s(n)^2)$ tal que a fórmula é verdadeira se e somente se M aceita x. Seja $m = O(s(n))$ o número de bits necessários para codificar uma configuração qualquer de M. Pelo Lema 2.7.1, existe uma fórmula booleana $\varphi_{M,x}$ tal que, para duas *strings* C, C', $\varphi_{M,x}(C, C') = 1$ se e somente se C e C' correspondem a duas configurações adjacentes no grafo de configurações $G_{M,x}$. Seja:

$$\psi = Q_1 x_1 Q_2 x_2 \ ... \ Q_n x_n \varphi(x_1, x_2, ..., x_n) \tag{1}$$

uma fórmula booleana quantificada de n variáveis, onde o tamanho de φ é m. A QBF $\psi^{C,C'}$, a qual ainda será definida, é uma fórmula verdadeira se e somente se existe um caminho direcionado partindo de C até C' em $G_{M,x}$. Perceba que, usando esta notação, $\psi^{C_{start},C_{accept}}$ é verdadeira se e somente se M aceita x.

A fórmula ψ será definida recursivamente. A notação $\psi_i^{C,C'}$ será usada para se referir a uma fórmula que é verdadeira se e somente se existe um caminho de tamanho máximo 2^i partindo de C até C' em $G_{M,x}$. Note que $\psi = \psi_m$ e que $\psi_0 = \varphi_{M,x}$. A parte importante nesta parte da prova é mostrar que: existe um caminho de tamanho máximo 2^i partindo de C até C' se e somente se existe uma configuração C'', de modo que existe um caminho de tamanho máximo 2^{i-1} partindo de C até C'', e existe um caminho de tamanho máximo 2^{i-1} partindo de C'' até C'. Utilizando C'' é possível definir ψ_i como: $\psi_i^{C,C'} = \exists C'' \ \psi_{i-1}^{C,C''} \ \wedge \psi_{i-1}^{C'',C'}$.

Note que a definição anterior para ψ_i acabou deixando a fórmula muito grande de modo que ψ_m teria tamanho exponencial. Ao invés disso é possível definir $\psi_i^{C,C'}$ como se segue:

$$\exists C'' \forall D_1 \forall D_2 \big((D_1 = C \wedge D_2 = C'') \vee (D_1 = C'' \wedge D_2 = C') \big) \Rightarrow \psi_{i-1}(D_1, D_2) \tag{2}$$

(Nesta equação "=" e "⇒" são simplificações que podem ser trocadas por fórmulas booleanas equivalentes, mais detalhes sobre esta transformação em Arora e Barak (2009).) Perceba que $size(\psi_i) \leq size(\psi_{i-1}) + O(m)$, então $size(\phi_m) \leq O(m^2)$.

□

Teorema 2.7.3. *TQBF é* **PSPACE***-completo*

Demonstração. A primeira parte da prova envolve mostrar que $L \leq_p$ TQBF para todo $L \in$ **PSPACE** e pode ser encontrada no Lema 2.7.2. A segunda parte é demonstrar de TQBF $\in$ **PSPACE**. Seja:

$$\psi = Q_1 x_1 Q_2 x_2 \ ... \ Q_n x_n \varphi(x_1, x_2, ..., x_n) \tag{3}$$

uma fórmula booleana quantificada de n variáveis, onde o tamanho de φ é dado por m. Primeiramente é mostrando que existe um algoritmo simples A o qual decide se ψ é verdadeiro em espaço $O(n + m)$. Considere que φ possa receber constantes de valor 0 ("falso") ou 1 ("verdadeiro"). Se $n = 0$ (não há variáveis), então a fórmula contém apenas constantes e pode ser avaliada em tempo e espaço $O(m)$, então será assumido que $n > 0$. Para $b \in \{0, 1\}$, seja $\psi_{\restriction x_1=b}$ a notação indicando que o primeiro quantificador Q_1 é removido e todas as ocorrências de x_1 são trocadas por uma constante b. Então o algoritmo A funciona como a seguir: se $Q_1 = \exists$, então retorna 1 se e somente se pelo menos um entre $A(\psi_{\restriction x_1=0})$ ou $A(\psi_{\restriction x_1=1})$ retorna 1; se $Q_1 = \forall$, então então retorna 1 se e somente se ambos $A(\psi_{\restriction x_1=0})$ e $A(\psi_{\restriction x_1=1})$ retornam 1. Pela definição de $\exists$ e $\forall$ é nítido que o retorno de A corresponde a qualquer ψ.

Seja $s_{n,m}$ a notação para o espaço usado por A em fórmulas com n variáveis e descrição de tamanho m. A parte importante nesta parte da prova é mostrar que a computação de ambos $A(\psi_{\restriction x_1=0})$ e $A(\psi_{\restriction x_1=1})$ pode ser feita utilizando o mesmo espaço. Especificamente, após computar $A(\psi_{\restriction x_1=0})$, o algoritmo A pode guardar guardar 1 bit com esta computação e reutilizar o espaço restante para computar $A(\psi_{\restriction x_1=1})$. Assumindo que A usa espaço $O(m)$ para escrever $A(\psi_{\restriction x_1=b})$ em suas chamadas recursivas, então $s_{n,m} = s_{n-1,m} + O(m)$. Dado que $s_{0,m} = O(m)$, então $s_{n,m} = O(n \cdot m)$. □

Na prova anterior os vértices do grafo de configurações podem ter grau 2; para grau 2 o grafo de configurações é gerado a partir de uma máquina não-determinística e isso implica que TQBF é **NPSPACE**-completo. E como TQBF $\in$ **PSPACE**, então **PSPACE** $=$ **NPSPACE**. Em seguida é definido o Teorema de Savitch, que é semelhante à prova anterior, mas que cria uma associação de complexidade entre as duas classes: **PSPACE** e **NPSPACE**.

Teorema 2.7.4 (Teorema de Savitch)**.** *Para qualquer função espaço-construível* $s : \mathbb{N} \to \mathbb{N}$ *onde* $s(n) \geq \log n$, **NSPACE***(*$s(n)$*)* $\subseteq$ **SPACE***(*$s(n)^2$*)*

Demonstração. Seja $L \in$ **NSPACE**$(s(n))$ uma linguagem decidível por uma MT M tal que para todo $x \in \{0, 1\}^n$, o grafo de configurações $G = G_{M,x}$ tenha até $V_M = 2^{O(s(n))}$ vértices, e determinar se $x \in L$ é equivalente a determinar se C_{accept} é alcançável a partir de C_{start} neste grafo. Então existe uma função REACH?(u, v, i) a qual retorna 0 ou 1 se existir um caminho de u até v de tamanho até 2^i; neste caso existe z tal que $d(u, z) \leq 2^{i-1}$ e $d(z, v) \leq 2^{i-1}$. Então, dado u, v, i e REACH?, todos os vértices z são testados (a custo de espaço $O(\log V_M)$) e a saída é 1 se REACH?$(u, z, i - 1) =$ REACH?$(z, v, i - 1) = 1$. Semelhante à prova anterior, este algoritmo faz n invocações recursivas e reutiliza o espaço em cada invocação. Assim, seja $s_{V_M,i}$ a complexidade de espaço de REACH?(u, v, i) de um grafo de V_M vértices, então

$s_{V_M,i} = s_{V_M,i-1} + O(\log V_M)$, logo $s_{V_M,\log V_M} = O(\log^2 V_M) = O(s(n)^2)$. (Note que C_{accept} é alcançado por C_{start} em até V_M passos.) □

2.8 Hierarquia Polinomial

Nesta seção são definidas mais classes usadas no trabalho e Máquina de Turing Alternante, ambas as definições usadas no capítulo de estado da arte (Capítulo 3).

2.8.1 Generalizando **NP** e **coNP**

Acredita-se que existem classes contendo problemas mais difíceis do que os problemas contidos em **NP** e **coNP** como a classe descrita a seguir.

Definição 2.8.1. A classe Σ_2^p é o conjunto de toda linguagem L para a qual existe uma MT M de tempo polinomial e uma função q polinomial tal que, para todo $x \in \{0,1\}^*$:

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)}\ \forall v \in \{0,1\}^{q(|x|)} M(x,u,v) = 1$$

Perceba que a Σ_2^p contém ambos **NP** e **coNP**. A definição anterior pode ser generalizada na chamada Hierarquia Polinomial tal como é enunciada a seguir:

Definição 2.8.2 (Hierarquia Polinomial (PH))**.** Para $i \geq 1$, a linguagem L pertence a Σ_i^p se existe uma MT M de tempo polinomial e um q polinomial tal que:

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)}\ ...\ Q_i u_i \in \{0,1\}^{q(|x|)} M(x,u_1,...,u_i) = 1$$

onde Q_i corresponde a $\forall$ ou $\exists$ quando i é respectivamente par ou ímpar. A hierarquia polinomial é o conjunto $\text{PH} = \cup_i \Sigma_i^p$.

Perceba que $\Sigma_1^p =$ **NP**. Para cada i é definido $\Pi_i^p = \left\{\overline{L} : L \in \Sigma_i^p\right\}$. Então $\Pi_1^p =$ **coNP**. Note também que, para todo i, $\Sigma_i^p \subseteq \Pi_{i+1}^p \subseteq \Sigma_{i+2}^p$, então PH também pode ser definido como $\cup_{i>0} \Pi_i^p$.

2.8.2 Máquinas de Turing Alternantes

Uma Máquina de Turing Alternante (MTA) é uma generalização do modelo de máquinas de Turing não-determinísticas. De maneira semelhante à MT não-determinísticas, Máquinas de Turing Alternantes têm duas funções de transição que possam escolher em cada passo. Com exceção dos estados q_{accept} e q_{halt}, todos os estados são marcados com $\exists$ ou $\forall$. Estados marcados com $\exists$ funcionam de maneira similar que em máquinas não-determinísticas, de modo que *existe* uma escolha que alcança o estado q_{accept}. Em estados marcados com $\forall$ ambas as escolhas (função de transição) alcançam o estado q_{accept} (de maneira análoga às máquinas não-determinísticas, máquinas neste estilo são chamadas de *co-não-determinísticas*). O termo "Alternante" se refere a alternância entre estados marcados com $\exists$ e $\forall$.

Definição 2.8.3 (Tempo Alternante)**.** Para todo $t : \mathbb{N} \to \mathbb{N}$ é dito que uma MT M alternante está no tempo $t(n)$ se, para toda entrada $x \in \{0,1\}^*$ e para toda possível sequência de escolha de função de transição, M para até no máximo $t(|x|)$ passos.

É definido que L é **ATIME**$(t(n))$ se existe uma constante c e uma MTA M de tempo $c \cdot t(n)$ se, para todo $x \in \{0,1\}^*$, M aceita x se e somente se $x \in L$. A seguir a definição de aceitar a entrada:

Seja $G_{M,x}$ o grafo de configurações de M recebendo x, onde, recapitulando, $G_{M,x}$ é acíclico e há uma aresta saindo da configuração C até a configuração C' se e somente se C' é alcançável por C após um passo de M. Alguns dos vértices são marcados com "ACCEPT" seguindo as regras abaixo sobre todo o grafo:

- A configuração C_{accept}, onde a máquina alcança o estado q_{accept}, é marcada com "ACCEPT";
- Se a configuração C é um estado marcado com $\exists$ e se há uma aresta saindo de C até C', onde C' é marcado com "ACCEPT", então C também é marcado com "ACCEPT";
- Se a configuração C é um estado marcado com $\forall$ e ambos C' e C'' alcançáveis em um passo são marcados com "ACCEPT", então C também é marcado com "ACCEPT";

É dito que M aceita x se, ao marcar todos os possíveis vértices com "ACCEPT", o vértice C_{start} também é marcado com "ACCEPT".

Também é útil definir classes com número de alternações constante. Para todo $i \in \mathbb{N}$, Σ_i**TIME**$(t(n))$ (resp. Π_i**TIME**$(t(n))$) é o conjunto de linguagens decidíveis por uma MTA M em $t(n)$ passos quando o estado inicial é $\exists$ (resp. $\forall$) e quando para toda entrada e para todo caminho direcionado iniciando na configuração do grafo de configurações, M alterna no máximo $i-1$ vezes o estado de uma marcação para outra. Fica então que para todo $i \in \mathbb{N}$, $\Sigma_i^p = \cup_c\ \Sigma_i$**TIME**$(n^c)$ e $\Pi_i^p = \cup_c \Pi_i$**TIME**(n^c).

Para um número ilimitado de alternações, é definido que **AP** $= \cup_c$ **ATIME**(n^c). Note que TQBF $\in$ **AP** e que **AP**, da mesma forma que TQBF, utiliza memória polinomial. Então fica que **AP** $=$ **PSPACE**. Também é conhecido que **APSPACE** $=$ **EXP**.

3 Estado da Arte

De acordo com Melkebeek et al. (2007) a maioria dos cientistas da área de complexidade acreditam que o problema de satisfatibilidade necessita de tempo exponencial linear (2^n) para ser decidido no pior caso – aliás, este é o custo de resolver o problema por força bruta. Além disso, acredita-se que seja necessário memória linear – note que, se fosse menor que isto, a conjectura anterior já estaria errada.

Inspirado na técnica de Kannan (1983), o primeiro limitante inferior em tempo e espaço para satisfatibilidade foi provado por Fortnow (2000) nos anos '90. O teorema provado é na verdade um pouco mais genérico como se segue:

Teorema 3.0.1 (Fortnow (2000))**.** *Para todo ϵ real positivo, a satisfatibilidade não pode ser decidida por uma máquina que respeite ambas as restrições ao mesmo tempo:*

1. *uma RAM (co-não)determinística que executa em $n^{1+o(1)}$ passos;*
2. *uma RAM (co-não)determinística que executa em tempo polinomial e use espaço $n^{1-\epsilon}$;*

Observe que, em particular, usando a notação da Seção 2.4, isso quer dizer que SAT $\notin$ **DTISP**$[n^{1+o(1)}, n^{1-\epsilon}]$. O Teorema 3.0.1 implica que não há (co-não)determinístico algoritmo que resolva o problema de satisfatibilidade em tempo $n^{1+o(1)}$ e espaço $n^{1-\epsilon}$ ao mesmo tempo. Mais tarde Lipton e Viglas (LIPTON; VIGLAS, 1999; FORTNOW et al., 2005) provaram que para memória subpolinomial ($n^{o(1)}$) o tempo deve ser superlinear (no caso deles, $n^{\sqrt{(2)-o(1)}}$). Fortnow e van Melkebeek melhoraram o limitante inferior quando a memória é polilogarítmica (LIPTON; VIGLAS, 1999; FORTNOW, 2000), exigindo que o expoente do tempo fosse a proporção áurea $\phi \approx 1.618$. Mais tarde Williams (WILLIAMS, 2005; WILLIAMS, 2008) também melhorou o limitante inferior, puxando o expoente para $2\cos(\pi/7) \approx 1.801$.Ou seja, sabe-se atualmente que SAT $\notin$**DTISP**$[n^{2\cos(\pi/7)}, n^{o(1)}]$.

A seguir o teorema considerado estado-da-arte na área de limitantes inferiores para satisfatibilidade em RAM's determinísticas:

Teorema 3.0.2 (Teorema Master para algoritmos determinísticos, Melkebeek et al. (2007))**.** *Para todo número real c e d tal que $(c-1)d < 1$ ou $cd(d-1) - 2d + 1 < 0$, existe um número real positivo e tal que a satisfatibilidade não pode ser resolvida por uma máquina que respeite ambas as restrições ao mesmo tempo:*

1. *RAM co-não-determinística que executa em n^c passos;*
2. *RAM co-não-determinística que executa em n^d passos e memória n^e;*

Além disso, conforme a constante e se aproxima de 1 por baixo, c se aproxima de 1 por cima e d é fixado.

Note que para uma máquina que respeite a primeira restrição, c não pode ser menor que 1; De acordo com a definição de função tempo-construtível (ver Subseção 2.1.3), o número

de passos precisa ser pelo menos linear. O mesmo vale para a segunda restrição de modo que d precisa ser maior ou igual a 1. Note também que a segunda restrição é um caso especial da primeira para quando $d \leq c$. Desse modo, os valores interessantes para c e d satisfazem $d \geq c \geq 1$.

O Teorema 3.0.2 se aplica quando c e d satisfazem uma disjunção de duas condições (ver definição de disjunção na Seção 2.6). Para $d > 2$ a condição $(c-1)d < 1$ é menos rigorosa que $cd(d-1) - 2d + 1 < 0$; para $d < 2$ a situação se inverte. Um gráfico envolvendo estes limites pode ser visto na Figura 2. A primeira condição pode ser usada de modo que valores cada vez maiores de d fazem c se aproximar de 1. Desse modo, o resultado de Fortnow é um corolário de do Teorema 3.0.2. A segunda condição não funciona para d com valores grandes e $c \geq 1$. A partir do Teorema 3.0.2, é possível conseguir um limitante inferior em tempo e espaço quando $c = d$. Desse modo a primeira condição implica em um limitante inferior de $n^{d-o(1)}$ para espaço subpolinomial, onde $d > 1$ satisfaz $d(d-1) = 1$, i.e., $d = \phi \approx 1.618$. A segunda condição implica em um limitante inferior em tempo de $n^{d-o(1)}$ para espaço subpolinomial, onde $d > 1$ satisfaz $d^2(d-1) - 2d + 1 = 0$. A solução dessa equação é $2\cos(\pi/7) \approx 1.801$, que é maior que ϕ. Deste modo, o Teorema Master também captura o resultado de Williams.

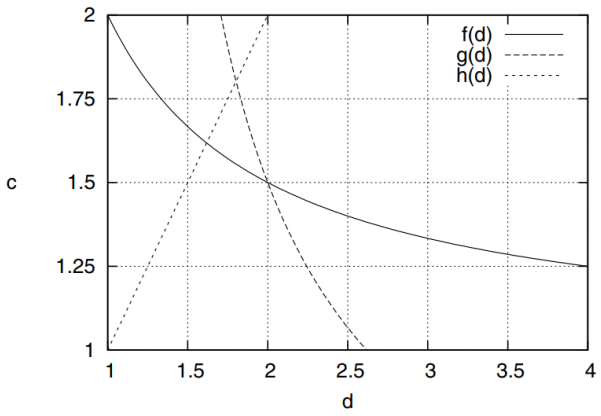

Figura 2 – Teorema Master para algoritmos determinísticos: $f(d)$ resolve $(c-1)d = 1$ para c, $g(d)$ resolve $cd(d-1) - 2d + 1 = 0$ para c, e $h(d)$ é a função identidade). Fonte: Melkebeek et al. (2007).

De acordo com (MELKEBEEK et al., 2007) é improvável que técnicas parecidas com as usadas por Williams consigam um limitante inferior em tempo acima do linear para memória linear.

Limitantes inferiores para satisfatibilidade em máquinas determinísticas estão relacionados com o problema P vs NP. De modo semelhante, no contexto de NP vs coNP, é possível criar limitantes inferiores para satisfatibilidade em máquinas não-determinísticas. O melhor limitante inferior em tempo e espaço conhecido até o momento para memória subpolinomial é tempo $n^{\sqrt[3]{4}-o(1)}$ (Diehl, Melkebeek e Williams (2007)).

Restringindo o modelo computacional para MT, o melhor limitante inferior em tempo

e espaço conhecido atualmente é de Santhanam (2001), que provou que $t \cdot s \geq n^{2-o(1)}$. Ou seja, para espaço subpolinomial, o tempo fica em $n^{2-o(1)}$.

Atualmente não é conhecido um limitante inferior para espaço em SAT melhor que logarítmico, então o problema **NP** vs **NL** é ainda está em aberto. Note que se **P** $\neq$ **NP** como é amplamente acreditado então **NP** $\neq$ **NL**.

3.1 Limitante Inferior em tempo e espaço para SAT

Nesta subseção é provada uma versão simplificada do limitante inferior em tempo e espaço para SAT baseada na prova de Kannan. Esta prova pode ser encontrada em (ARORA; BARAK, 2009). Observa-se que esta prova utiliza apenas MT's.

Lema 3.1.1. **DTISP**$(n^{12}, n^2) \subseteq \Sigma_2$**TIME**$(n^8)$

Demonstração do Lema 3.1.1. Esta prova é similar ao Teorema de Savitch e a prova de que TQBF é **PSPACE**-completo. Suponha que L é decidida por M em tempo n^{12} e espaço n^2 para *string* de entrada de tamanho n. Considere o grafo de configurações $G_{M,x}$ para M recebendo $x \in \{0,1\}^*$. Cada configuração neste grafo pode ser descrita por uma *string* de tamanho $O(n^2)$. Aqui, $x \in L$ se e somente se há um caminho saindo de C_{start} até C_{accept} em até n^{12} passos. Há este caminho se e somente se *existem* n^6 configurações $C_1, \ \ldots \ , C_{n^6}$ (note que estas configurações exigem $O(n^8)$ para serem escritas) tal que se $C_1 = C_{start}$ então C_{n^6} é a configuração que aceita a entrada e *para todo* $i \in [n^6]$ a configuração C_i é computada a partir de C_{i-1} em n^6 passos. Esta condição pode ser verificada em, por exemplo, tempo $O(n^6)$, então existe uma Σ_2-MT de tempo $O(n^8)$ que decida L.

Seja M a máquina que decida L. Para uma MT M' e constantes c e d, a máquina M aceita uma *string* x se e somente se:

$$\exists\{C_1, ..., C_{n^6}\}; C_1 = C_{start} \wedge C_{n^6} = C_{accept} \wedge \ \forall i \in [n^6] \ M'(C_{i-1}, C_i) = 1$$

onde M' verifica se C_i é alcançável por C_{i-1} em n^6 passos. Utilizando o Lema 2.7.1, M' utiliza tempo $O(n^6)$. □

Lema 3.1.2. *Suponha que* **NTIME**$(n) \subseteq$ **DTIME**$(n^{1.2})$*. Então* Σ_2**TIME**$(n^8) \subseteq$ **NTIME**$(n^{9.6})$

Demonstração do Lema 3.1.2. Usando a equivalência entre tempo alternante e hierarquia polinomial, L está em Σ_2**TIME**(n^8) se e somente se existe uma MT M tal que, para as constantes c e d:

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{c|x|^8} \ \forall v \in \{0,1\}^{d|x|^8} \ M(x,u,v) = 1$$

onde M executa em $O(|x|^8)$ passos. Seja $|x| = n$ e $|u| = cn^8$. Pela hipótese sabe-se que **NTIME**$(n) \subseteq$ **DTIME**$(n^{1.2})$, então utilizando um argumento de preenchimento (*padding argument*) existe um algoritmo determinístico D que, recebendo x e u, executa em tempo

$O((n^8)^{1.2}) = O(n^{9.6})$ e retorna 1 se e somente se existe algum $v \in \{0,1\}^{dn^8}$ tal que $M(x,u,v) = 0$. Assim,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{c|x|^8}\ D(x,u) = 0$$

implica que $L \in$ **NTIME**$(n^{9.6})$. □

Teorema 3.1.3. *SAT* $\notin$ **DTISP***$(n^{1.2-o(1)}, n^{0.2})$.*

Demonstração. Suponha que SAT $\notin$ **DTISP**$(n^{1.2-o(1)}, n^{0.2})$, então **NTIME**$(n) \subseteq$ **DTISP**$(n^{1.2}, n^{0.2})$; usando argumento de preenchimento (*padding argument*) **NTIME**$(n^{10}) \subseteq$ **DTISP**(n^{12}, n^2); usando o Lema 3.1.1, existe a implicação de que **DTISP**$(n^{12}, n^2) \subseteq \Sigma_2$**TIME**$(n^8)$; pelo Lema 3.1.2, Σ_2**TIME**$(n^8) \subseteq$ **NTIME**$(n^{9.6})$; logo, tem-se que **NTIME**$(n^{10}) \subseteq$ **DTISP**$(n^{12}, n^2) \subseteq \Sigma_2$**TIME**$(n^8) \subseteq$ **NTIME**$(n^{9.6})$, porém isto contradiz o Teorema da Hierarquia de Tempo. □

4 Resultados

Neste capítulo são enunciados os resultados, os quais foram divididos entre *provas alternativas para resultados conhecidos* e *demonstrações originais*.

4.1 Provas alternativas de resultados conhecidos

Em todos os enunciados desta subseção o modelo de computação utilizado é o RAM (apesar de que a prova também funciona para MT).

4.1.1 Se **EXP** $\neq$ **EXPSPACE** então **P** $\neq$ **PSPACE**

Utilizando o argumento de preenchimento (*padding argument*) já sabe-se que, se **P** $=$ **PSPACE**, então **EXP** $=$ **EXPSPACE**. Fazendo uma simples contraposição conclui-se que se **EXP** $\neq$ **EXPSPACE** então **P** $\neq$ **PSPACE**. A prova descrita nesta seção não utiliza argumento de preenchimento.

Lema 4.1.1. *Seja $t(n)$ uma função tempo-construtível e $s(n)$ uma função espaço-construtível tal que $s(n) > n$. Se TQBF $\in$* **DTIME***$[n^c]$, então* **DTISP***$[t(n), s(n)] \subseteq$* **DTIME***$[(s(n)\log(t(n)))^c]$*

Demonstração. Seja x uma entrada de tamanho n e L uma linguagem em **DTISP**$[t(n), s(n)]$. Então existe uma máquina M que decida L em $t(n)$ passos e utilize $s(n)$ blocos de memória. Seja $m = O(s(n))$ o número de bits necessários para codificar uma configuração qualquer de M. Pelo Lema 2.7.1, existe uma fórmula booleana $\varphi_{M,x}$ tal que, para duas *strings* C, C', $\varphi_{M,x}(C, C') = 1$ se e somente se C e C' correspondem a duas configurações adjacentes no grafo de configurações $G_{M,x}$. Seja:

$$\psi = Q_1 x_1 Q_2 x_2 \ ... \ Q_n x_n \varphi(x_1, x_2, ..., x_n) \tag{4}$$

uma fórmula booleana quantificada de $\log t(n)$ variáveis, onde o tamanho de φ é m. A QBF $\psi^{C,C'}$, a qual ainda será definida, é uma fórmula verdadeira se e somente se existe um caminho direcionado partindo de C até C' em $G_{M,x}$. Perceba que, usando esta notação, $\psi^{C_{start},C_{accept}}$ é verdadeira se e somente se M aceita x.

A fórmula ψ será definida recursivamente. A notação $\psi_i^{C,C'}$ será usada para se referir a uma fórmula que é verdadeira se e somente se existe um caminho de tamanho máximo 2^i partindo de C até C' em $G_{M,x}$. Note que $\psi = \psi_m$ e que $\psi_0 = \varphi_{M,x}$. A parte importante nesta parte da prova é mostrar que: existe um caminho de tamanho máximo 2^i partindo de C até C' se e somente se existe uma configuração C'', de modo que existe um caminho de tamanho máximo 2^{i-1} partindo de C até C'', e existe um caminho de tamanho máximo 2^{i-1} partindo de C'' até C'. Utilizando C'' é possível definir ψ_i como: $\psi_i^{C,C'} = \exists C'' \ \psi_{i-1}^{C,C''} \ \wedge \psi_{i-1}^{C'',C'}$.

Note que a definição anterior para ψ_i acabou deixando a fórmula muito grande de modo que ψ_m teria tamanho exponencial. Ao invés disso é possível definir $\psi_i^{C,C'}$ como se segue:

$$\exists C''\forall D_1\forall D_2\big((D_1 = C \wedge D_2 = C'') \vee (D_1 = C'' \wedge D_2 = C')\big) \Rightarrow \psi_{i-1}(D_1, D_2) \quad (5)$$

(Nesta equação "=" e "⇒" são simplificações que podem ser trocadas por fórmulas booleanas equivalentes, mais detalhes sobre esta transformação em Arora e Barak (2009).) Perceba que $size(\psi_i) \leq size(\psi_{i-1}) + O(m)$, então $size(\phi_m) \leq O(m^2)$.

Decidir se ψ é verdadeiro é um problema que está em TQBF. Pela hipótese do enunciado, TQBF $\in$ **DTIME**$[n^c]$, então o custo de resolver este problema é $|\psi|^c = (s(n)\log(t(n)))^c$ □

Lema 4.1.2. **DSPACE***[$s(n)$]* $\subseteq$ **DTISP***[$2^{s(n)}, s(n)$]*

Demonstração. Seja M uma máquina que, recebendo uma entrada de tamanho n, utilize $t(n)$ passos e $s(n)$ blocos de memória. Para que M utilize $t(n)$ passos são necessários existir pelo menos $t(n)$ configurações distintas ao executar M. Dado o tamanho da memória, o grafo $G_{M,x}$ criado a partir de M e x pode ter no máximo $2^{s(n)}$ vértices diferentes. Assim, $t(n) \leq 2^{s(n)}$, logo $\log(t(n)) \leq s(n)$. □

Lema 4.1.3. *Seja f uma função tempo e espaço construível tal que $f(n) \geq n$. Se TQBF $\in$* **DTIME***$[n^c]$ então* **DSPACE***$[f(n)] \in$* **DTIME***$[f(n)^{2c}]$.*

Demonstração. De acordo com o Lema 4.1.2, **DSPACE**$[f(n)] \subseteq$ **DTISP**$[2^{f(n)}, f(n)]$. De acordo com o Lema 4.1.1, **DTISP**$[2^{f(n)}, f(n)] \subseteq$ **DTIME**$[(f(n)\log(2^{f(n)}))^c] =$ **DTIME**$[(f(n)^2)^c]$ = **DTIME**$[f(n)^{2c}]$. □

Teorema 4.1.4. *Se* **EXP** $\neq$ **EXPSPACE** *então* **P** $\neq$ **PSPACE**

Demonstração. Suponha que **P** = **PSPACE**, então TQBF $\in$ **DTIME**(n^c) para uma constante c. Seja L uma linguagem tal que $L \notin$ **EXP** e $L \in$ **EXPSPACE**. Seja $s(n)$ o espaço utilizado para decidir L. Pelo Lema 4.1.3, isto significa que **DSPACE**$[s(n)] \subseteq$ **DTIME**$[s(n)^{2c}]$. Porém **DTIME**$[s(n)^{2c}] \subseteq$ **EXP**, então $L \in$ **EXP**, logo **EXP** = **EXPSPACE**. □

4.2 Demonstrações Originais

Nesta seção são apresentados os resultados originais deste trabalho.

4.2.1 Resultado relacionado a limitante inferior em espaço para SAT

Nesta subseção é demonstrado um resultado relacionado a limitante inferior para SAT em espaço. Em todos os enunciados desta subseção o modelo de computação utilizado é o RAM (apesar de que a prova também funciona para MT).

Lema 4.2.1. *Seja $t(n)$ uma função tempo-construtível e $s(n)$ uma função espaço-construtível, então* **DTIME**$(t(n), s(n)) \subseteq \Sigma_2$**TIME**$(s(n)\sqrt{t(n)})$.

Demonstração. Esta prova é uma generalização do Lema 3.1.1. Suponha que L é decidida por M em tempo $t(n)$ e espaço $s(n)$. Considere o grafo de configurações $G_{M,x}$ para M recebendo $x \in \{0,1\}$. Cada configuração neste grafo pode ser descrita por uma *string* de tamanho $O(s(n))$. Aqui, $x \in L$ se e somente se há um caminho saindo de C_{start} até C_{acecpt} em até $t(n)$ passos. Há este caminho se e somente se *existem* $\sqrt{t(n)}$ configurações $C_1, \ldots, C_{\sqrt{t(n)}}$ (exigindo espaço de tamanho $O(s(n)\sqrt{t(n)})$ para serem escritas) tal que se $C_1 = C_{start}$ então $C_{\sqrt{t(n)}}$ é a configuração que aceita a entrada e *para todo* $i \in [\sqrt{t(n)}]$ a configuração C_i é computada a partir de C_{i-1} em $\sqrt{t(n)}$ passos. Esta condição pode ser verificada em, por exemplo, $O(s(n)\sqrt{t(n)})$ passos, então existe uma Σ_2-TM de tempo $O(s(n)\sqrt{t(n)})$ que decida L. □

Lema 4.2.2. *Seja $f(n)$ uma função tempo construtível e sejam k, k' constantes. Suponha que* **NTIME**$(n) \subseteq$ **DTISP**$(n^k, \log n^{k'})$. *Então* Σ_2**TIME**$(f(n)) \subseteq$ **DSPACE**$(\log f(n)^{kk'})$

Demonstração. Esta prova é semelhante ao Lema 3.1.2. Usando a equivalência entre tempo alternante e hierarquia polinomial, L está em Σ_2**TIME**$(f(n))$ se e somente se existe uma MT M tal que, para as constantes c e d:

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{cf(|x|)}\ \forall v \in \{0,1\}^{df(|x|)}\ M(x,u,v) = 1$$

onde M executa em $O(f(|x|))$ passos. Seja $|x| = n$ e $|u| = cf(n)$. Se **NTIME**$(n) \subseteq$ **DTISP**$(n^k, \log n^{k'})$, então utilizando um argumento de preenchimento o algoritmo determinístico D recebendo x e u, executa em tempo $O(f(n)^k)$ e retorna 1 se e somente se existe algum $v \in \{0,1\}^{df(n)}$ tal que $M(x,u,v) = 0$. Assim,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{cf(|x|)}\ D(x,u) = 0$$

implica que $L \in$ **NTIME**$(f(n)^k)$. Pelo enunciado, **NTIME**$(n) \subseteq$ **DTISP**$(n^k, \log n^{k'})$. A utilização do argumento de preenchimento novamente implica que

$$\textbf{NTIME}(f(n)^k) \subseteq \textbf{DTISP}((f(n)^k)^k, \log(f(n)^k)^{k'}) \subseteq \textbf{DSPACE}(\log f(n)^{kk'})$$

. □

Lema 4.2.3. *Para uma função f, $f(2^n) = n^{O(1)}$ se e somente se f for uma função polilogarítmica ou outra de menor ordem.*

Demonstração. De acordo com a Seção 2.1, uma função polilogarítmica está em $O(\log^{O(1)}(n))$. Para uma constante k qualquer, seja $f(n) = O(\log^k(n))$. Então para f recebendo 2^n como entrada:

$$f(2^n) = O(\log^k(2^n)) = O((\log 2^n)^k) = O(n^k) \tag{6}$$

Suponha que g seja maior assintoticamente que qualquer outra função polilogarítmica. Então, para uma constante k, $g(n) = \omega(\log^k n)$. Conforme a equação a seguir, existe uma função $h(n)$ tal que $g(n) = \log^{h(n)} n$:

$$\begin{aligned} h'(n) &= \log^{h(n)} n \\ \log_{\log n} h'(n) &= h(n) \log_{\log n} \log n \\ \log_{\log n} h'(n) &= h(n) \end{aligned} \tag{7}$$

Para $g(n) = \log^{h(n)} n$, $h(n) = \omega(1)$ ($h(n)$ não pode ser constante senão g seria uma função polilogarítmica). Para g recebendo 2^n como entrada:

$$g(2^n) = \log^{\omega(1)}(2^n) = (\log 2^n)^{\omega(1)} = n^{\omega(1)} \tag{8}$$

□

Lema 4.2.4. *Para toda função f tal que $f(n) = \omega(\log(n))$, $2^{f(n)} = n^{\omega(1)}$.*

Demonstração. Pelo enunciado, $f(n) = \omega(\log(n))$. Então $f(n) = \log(n) \cdot f'(n)$, onde $f'(n) = f(n)/\log(n) = \omega(\log(n))/\log(n) = \omega(1)$. Logo, $f(n) = \log(n) \cdot \omega(1)$.

Conforme demonstrado a seguir, $2^{f(n)} = n^{\omega(1)}$:

$$\begin{aligned} n^{g(n)} &= 2^{f(n)} \\ n^{g(n)} &= 2^{\log(n)\omega(1)} \\ \log(n^{g(n)}) &= \log(2^{\log(n)\omega(1)}) \\ g(n)\log(n) &= \log(n)\omega(1) \\ g(n) &= \omega(1) \end{aligned} \tag{9}$$

□

Lema 4.2.5. *Para toda função polilogarítmica f tal que $f(n) = \log(n) \cdot \omega(1)$ e uma constante k, se* **NTIME**$(2^n) \not\subseteq$ **DSPACE**(n^k)*, então SAT $\notin$* **DTISP**$(n^k, f(n))$*.*

Demonstração. Pela hipótese do enunciado existe uma linguagem L que está em **NTIME**(2^n) mas que não está em **DSPACE**(n^k). Suponha por absurdo que SAT $\in$ **DTISP**$(n^k, f(n))$, então **NTIME**$(2^n) \subseteq$ **DTISP**$(2^{kn}, f(2^n))$. Pelo Lema 4.2.3 sabe-se que **DTISP**$(2^{kn}, f(2^n)) =$ **DTISP**$(2^{kn}, n^{O(1)})$. Pelo Lema 4.2.1, **DTISP**$(2^{kn}, n^{O(1)}) \subseteq \Sigma_2$**TIME**$(2^{kn/2} n^{O(1)})$. Finalmente, pelo Lema 4.2.2, Σ_2**TIME**$(2^{kn/2} n^{O(1)}) \subseteq$ **DSPACE**$(\log(2^{kn/2} n^{O(1)})^{O(1)}) =$ **DSPACE**$(n^{O(1)})$, o que contradiz a definição de L. □

Teorema 4.2.6. *Para uma constante k, se* **NTIME**$(2^n) \not\subseteq$ **DSPACE**(n^k)*, então* **NP** $\not\subset$ **PolyL**.

Demonstração. Se **NTIME**$(2^n) \not\subseteq$ **DSPACE**(n^k) então **NEXP** $\neq$ **PSPACE**. Utilizando o argumento do preenchimento, isto implica que **NP** $\neq$ **NL**. Pelo Lema 4.2.5, isto também implica que SAT não pode usar memória poligoratímica, ou seja, **NP** $\not\subset$ **PolyL**. □

Note que, pelo Teorema 4.2.6, provar que **NTIME**$(2^n) \not\subseteq$ **DSPACE**(n^k) implicaria que $\forall k \geq 1$, **NP** $\neq$ **L**^{k}.

5 Considerações Finais

Apesar de Complexidade Computacional ser uma área difícil de contribuir é também uma área com muitos problemas em aberto. O resultado provado no Teorema 4.2.6 deve ser verificado pois o grau de formalismo desenvolvido não cobre todas as alternativas abordadas na sequência. Trabalhos futuros podem aumentar o grau de formalismo; tentar remover a hipótese sobre o teorema; verificar se o limitante é o mesmo em outros modelos de computação, por exemplo em máquinas com fitas multidimensionais e em computadores quânticos; verificar se o limitante é o mesmo utilizando um modelo probabilístico; verificar o limitante em outros problemas próximos ao SAT, por exemplo #SAT; verificar se o limitante provado neste trabalho tem alguma implicação em tamanho de circuitos ou profundidade de circuitos.

Referências

ARORA, S.; BARAK, B. **Computational complexity: A modern approach**. 1. ed. [S.l.]: Cambridge University Press, 2009. ISBN 9780521424264,0521424267. Citado 7 vezes nas páginas 13, 14, 17, 20, 22, 28 e 31.

BOLLOBAS, B. **Graph Theory: An Introductory Course**. 1st ed. 1979. corr. 3rd printing. ed. [S.l.]: Springer, 1994. (Graduate Texts in Mathematics). ISBN 0387903992. Citado na página 39.

DIEHL, S.; MELKEBEEK, D. van; WILLIAMS, R. **A new time-space lower bound for nondeterministic algorithms solving tautologies**. [S.l.], 2007. Citado na página 27.

FORTNOW, L. Time–space tradeoffs for satisfiability. **Journal of Computer and System Sciences**, Elsevier, v. 60, n. 2, p. 337–353, 2000. Citado 2 vezes nas páginas 13 e 26.

FORTNOW, L. The status of the p versus np problem. **Communications of the ACM**, ACM, v. 52, n. 9, p. 78–86, 2009. Citado na página 12.

FORTNOW, L. et al. Time-space lower bounds for satisfiability. **Journal of the ACM (JACM)**, ACM, v. 52, n. 6, p. 835–865, 2005. Citado na página 26.

INSTITUTE, C. M. **P vs NP Problem**. 2018. <http://www.claymath.org/millennium-problems/p-vs-np-problem>. Acessado em 22/04/2018. Citado na página 12.

KANNAN, R. Alternation and the power of nondeterminism. In: ACM. **Proceedings of the fifteenth annual ACM symposium on Theory of computing**. [S.l.], 1983. p. 344–346. Citado na página 26.

LIPTON, R. J.; VIGLAS, A. On the complexity of sat. In: IEEE. **Foundations of Computer Science, 1999. 40th Annual Symposium on**. [S.l.], 1999. p. 459–464. Citado na página 26.

MELKEBEEK, D. V. et al. A survey of lower bounds for satisfiability and related problems. **Foundations and Trends® in Theoretical Computer Science**, Now Publishers, Inc., v. 2, n. 3, p. 197–303, 2007. Citado 3 vezes nas páginas , 26 e 27.

PAPADIMITRIOU, C. H. **Computational Complexity**. [S.l.]: Addison-Wesley, 1994. ISBN 0201530821,9780201530827. Citado na página 20.

SANTHANAM, R. Lower bounds on the complexity of recognizing sat by turing machines. **Information Processing Letters**, Elsevier North-Holland, Inc., v. 79, n. 5, p. 243–247, 2001. Citado na página 28.

SMALE, S. Mathematical problems for the next century. **The mathematical intelligencer**, Springer, v. 20, n. 2, p. 7–15, 1998. Citado na página 12.

WILLIAMS, R. Better time-space lower bounds for sat and related problems. In: IEEE. **Computational Complexity, 2005. Proceedings. Twentieth Annual IEEE Conference on**. [S.l.], 2005. p. 40–49. Citado na página 26.

WILLIAMS, R. R. Time-space tradeoffs for counting np solutions modulo integers. **Computational Complexity**, Springer, v. 17, n. 2, p. 179–219, 2008. Citado na página 26.

Apêndices

APÊNDICE A – Teoria de Grafos

Nesta seção são definidos alguns conceitos básicos de grafos baseados em Bollobas (1994).

Define-se *grafo* o par de conjuntos de *vértices* $V = \{v_1, v_2, ..., v_n\}$, de *arestas* $E \subset \{\{v_i, v_j\}$ t.q. $v_i, v_j \in V\}$. É chamado de *grau de um vértice* o número de arestas conectadas a um vértice v_i, o qual é denotado por $g(v_i)$. Em grafos *direcionados* cada aresta contém também o sentido para a qual aponta.

É chamado de *caminho* o grafo formados pelo conjunto de vértices $\{v_1, v_2, ..., v_n\}$ e no qual as arestas seguem o padrão $\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_{n-1}, v_n\}$. Note que, com exceção das "pontas", todos os vértices têm grau 2. Um *ciclo* é semelhante a um caminho, contendo também o par $\{v_n, v_1\}$ no conjunto de arestas (note que assim todos os vértices têm grau 2). Um grafo que não contém dentro de si nenhum ciclo é chamado de *grafo acíclico*.

Sejam u e v vértices de um grafo G. Se G é *conexo* então é existe um caminho entre u e v.

Definição A.0.1 (Distância entre dois vértices)**.** Para um grafo G e dois vértices u e v, $d(u, v)$ define o tamanho do menor caminho entre u e v. Note que se este caminho não existir então d retorna ∞. Perceba também que, em um grafo direcionado, este caminho precisa seguir o sentido das arestas partindo de u.