Universidade Federal de Pernambuco
Centro de Ciências Exatas e da Natureza
Departamento de Matemática

Pós-graduação em Matemática

Uma Redução do Problema de Fatorização de Inteiros para o Problema de Programação 0-1

Fábio Happ Botler

Recife

Universidade Federal de Pernambuco
Centro de Ciências Exatas e da Natureza
Departamento de Matemática

Fábio Happ Botler

Uma Redução do Problema de Fatorização de Inteiros para o Problema de Programação 0-1

Trabalho apresentado ao Programa de Pós-graduação em Matemática do Departamento de Matemáticada Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Matemática.

Orientador: Sóstenes Luiz Soares Lins

Recife

Catalogação na fonte
Bibliotecária Joana D'Arc L. Salvador, CRB 4-572

Botler, Fábio Happ.
Uma redução do problema de fatorização de inteiros para o problema de programação 0-1 / Fábio Happ Botler - Recife: O Autor, 2011.
vii, 36 folhas.

Orientador: Sóstenes Luiz Soares Lins
Dissertação (Mestrado) - Universidade Federal de Pernambuco. CCEN. Matemática, 2011.

Inclui bibliografia e apêndice.

1.Análise combinatória . 2.Algoritmos. I.Lins, Sóstenes Luiz Soares (orientador). II. Título.

511.6 (22.ed.) MEI 2011-009

Dissertação submetida ao Corpo Docente do Programa de Pós-graduação do Departamento de Matemática da Universidade Federal de Pernambuco como parte dos requisitos necessários para a obtenção do Grau de Mestrado em Matemática.

Aprovado: _____

Sóstenes Luiz Soares Lins, UFPE
Orientador

_____

Manoel José Soares Machado Lemos, UFPE

_____

Tereza Bernarda Ludermir, UFPE

UMA REDUÇÃO DO PROBLEMA DE FATORIZAÇÃO DE INTEIROS PARA O PROBLEMA DE PROGRAMAÇÃO 0-1

Por

Fábio Happ Botler

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE MATEMÁTICA
Cidade Universitária – Tels. (081) 2126 - 8414 – Fax: (081) 2126 - 8410
RECIFE – BRASIL

Fevereiro – 2011

Agradecimentos

Agradeço à minha família, amigos, colegas e professores por toda a força e apoio.

Agradeço também a Capes pelo financiamento da pesquisa.

Resumo

O problema de Fatorização de Inteiros, assim como os outros em $\mathcal{NP}$, pode ser reduzido em tempo polinomial para o problema de Satisfabilidade, devido ao Teorema de Cook. O problema de Satisfabilidade, por sua vez, pode ser reduzido facilmente ao problema de Programação Inteira. Este trabalho apresenta uma dessas reduções, isto é, Fatorização $\longrightarrow$ Programação Inteira e algumas particularidades encontradas. Obtemos uma redução de ordem $O(n^2)$ no número de dígitos binários de um inteiro N a ser fatorado e, além disso, encontramos algumas propriedades locais da matriz final que podem auxiliar um possível estágio de pré-processamento.

Palavras Chave: Redução, Fatoração, Fatorização, Programação Inteira, Programação 0-1

Abstract

The Integer Factorization Problem, as the others $\mathcal{NP}$ problems, may be reduced in polynomial time to the Satisbiability Problem, due to Cook's Theorem. The Satisfiability Problem, in turn, can easily be reduced to the Integer Programming Problem. This work presents one of such reductions, i.e, Factorization $\longrightarrow$ Integer Programming and some praticularities found. We obtain a reduction of order $O(n^2)$ on the number of binary digits of an integer N to be factored and, furthermore, we found some local properties of the final matrix that can help a possible pre-processing stage.

Keywords: Reduction, Factorization, Integer Programming

Sumário

Capítulo 1

Introdução

Dentro da Matemática, a Combinatória é uma das áreas com maior número de aplicações em outros ramos da ciência, especialmente na Ciência da Computação. O desenvolvimento dos computadores em termos de potência e portabilidade permitiu vários avanços na indústria, particularmente com a implementação de algoritmos de Otimização Combinatória.

Um algoritmo é uma lista de instruções para se resolver um determinado problema. Em geral, classificamos um algoritmo pelo tempo que ele leva para resolver este problema.

Este trabalho consiste em uma ligação entre dois tradicionais problemas na matemática: O Problema de **Fatorização de Inteiros** [8] e o Problema de **Programação Inteira** [5]. O Problema de Fatorização de Inteiros se resume a dado um inteiro $z = p_1^{a_1} \cdots p_k^{a_k}$, onde $p_1, \ldots, p_k$ são primos, encontrar $p_1, \ldots, p_k$. O Problema de Programação Inteira pode ser escrito como

$$\begin{cases} \text{Max} & c \cdot x \\ \text{Sujeito a} & A \cdot x \leq b \\ & x \text{ inteiro} \end{cases}$$

onde $c \in \mathbb{Z}^n$, A é uma matriz $m \times n$ inteira e $b^T \in \mathbb{Z}^m$.

Ao tentar resolver um problema, é muito comum se deparar com outros problemas previamente conhecidos e talvez já resolvidos. Nestes casos é possível usar um algorimo que resolva o segundo problema como sub-rotina do algoritmo para resolver o primeiro. É possível também fazer a transformação de um problema em outro, neste caso denominamos **Redução** entre os problemas.

A ligação que procuramos neste trabalho é uma redução do problema de Fatorização de Inteiros para o problema de Programação 0-1, uma particularidade do problema de Programação Inteira onde as variáveis são binárias. Em outras palavras, é possível transformar rapidamente um problema de Fatorização em um problema de Programação Inteira.

Os algoritmos são classificados de acordo com sua **Complexidade** e esta teoria leva o nome de *Teoria da Complexidade Computacional* [7, 12]. A complexidade de um algoritmo depende do número de operações básicas que ele faz ao tentar resolver um problema. Dizemos que um algoritmo A resolve um problema em **Tempo Polinomial** se existe um polinômio $P(x)$ tal que o número de operações básicas que A faz para todas as entradas de tamanho n é menor que $P(n)$.

Assim, os algoritmos podem ser divididos em classes. Algumas dessas classes são $\mathcal{P}$,$\mathcal{NP}$,co-$\mathcal{NP}$. Resumidamente, dizemos que um problema está em $\mathcal{P}$ se existe um algoritmo que o resolve em tempo polinomial; um problema está em $\mathcal{NP}$ se existe um algoritmo que verifica se uma solução é válida em tempo polinomial. A maioria dos problemas em Otimização Combinatória pertence a uma dessas duas classes, o que as torna muito mais interessantes e importantes.

Outro conceito interessante é o de **Completude** $\mathcal{NP}$. Um problema R em $\mathcal{NP}$ é dito **$\mathcal{NP}$-completo** se todo problema em $\mathcal{NP}$ pode ser reduzido em tempo polinomial a R.

Um importante resultado na teoria de Complexidade Computacional é o **Teorema de Cook** – Cook-Levin [4, 9]– que afirma que o Problema de Satisfabilidade Booleana é $\mathcal{NP}$-Completo e, portanto, todo problema na classe $\mathcal{NP}$ pode ser reduzido em tempo polinomial a um problema de satisfabilidade.

O Teorema de Cook fornece uma redução em tempo polinomial canônica de qualquer problema em $\mathcal{NP}$ para o problema de Satisfabilidade. Porém, por mais que esta redução seja polinomial, ainda é bastante cara em termos computacionais. Neste trabalho apresentamos uma redução mais eficiente para o problema de Satisfabilidade e fazemos uso de uma redução simples do problema de Satisfabilidade para o problema de Programação 0-1. Com essa redução em mãos é possível encontrar uma série de particularidades que podem auxiliar na fatorização inicial.

É possível construir, para cada inteiro a ser fatorado, um Programa 0-1 com apenas um ponto inteiro de onde podemos extrair facilmente a fatorização do inteiro dado. Além disso, obtemos várias relações entre as restrições que permitem uma contração mais eficiente do politopo ao usar o *Método dos Cortes Paralelos*.

Capítulo 2

O Problema de Fatorização

Seja N um inteiro qualquer. Sabemos que N pode ser escrito de forma única como produto de números primos. O problema de Fatorização de Inteiros é exatamente descobrir que produto é este.

A princípio podemos considerar a possibilidade de testar todos os inteiros menores do que N a fim de encontrar um que o divida. Quanto tempo este procedimento precisaria? Para um número pequeno o método é bastante prático, levando menos do que 2 segundos para fatorar um número de 9 dígitos num computador simples. Porém, se aumentarmos um dígito no número a ser fatorado, precisaremos testar 10 vezes mais números para encontrar um inteiro que divida o número dado. Isso tomaria um tempo também 10 vezes maior, de forma que para um número de 20 dígitos levaríamos $10^{10} \times 2$ segundos para fatorar, isto é, aproximadamente, 650 anos.

O exemplo dado acima é um algoritmo de força bruta, que procura testar todos os candidatos a fatoração do inteiro N. O tempo de processamento dele é de $(2 \times 10^{-10}) \times 10^{D}$ segundos, onde D é o número de dígitos de N. Quando o tempo de processamento é uma relação do tipo $T(N) = c \times a^{f(N)}$, onde $f(N)$ é uma função polinomial no tamanho da variável N, dizemos que o algoritmo é de *tempo exponencial*.

De fato, não é conhecida a existência de nenhum algoritmo eficiente para a fatoração de um inteiro. Existem algoritmos com tempos de processamento melhores do que o exemplo acima, como por exemplo o *Lenstra's Algorithm* [6] ou uma melhora proposta em [3]. Multiplicação, portanto, é denominada uma função de via única, pois, enquanto multiplicar dois números é uma tarefa computacionalmente prática, fatorar um número é inviável.

2.1 Criptografia RSA

A criptografia é uma ferramenta de extrema importância utilizada tanto no armazenamento quanto na transferência da informação e já desempenhou papel fundamental em diversas guerras e conflitos. Um exemplo importantíssimo foi a Segunda Guerra Mundial [13]. A Alemanha se comunicava usando o *ENIGMA*, uma criptografia extremamente forte, e a Inglaterra, após muito trabalho, conseguiu quebrar o código alemão e prever a ação inimiga.

Os sistemas de criptografia consistem de um algoritmo para criptografar e outro para descriptografar. Esses algoritmos, em geral, dependem de uma chave que é conhecida somente pela pessoa que vai criptografar e pela pessoa que vai descriptografar a mensagem. Uma vez conhecida a chave, é possível ter acesso às informações das mensagens criptografadas. Portanto, a forma com que essas chaves vão chegar às pessoas que pretendem trocar mensagens criptografadas é um problema importante a ser considerado. As pessoas podem simplesmente se encontrar uma vez e trocar uma palavra que será usada como chave nas próximas mensagens, porém, o ato de repetir a chave em várias mensagens e o de usar uma palavra como chave são duas falhas graves de segurança, isto é, uma pessoa que queira ter acesso às informações das mensagens pode utilizar esses fatos para descobrir mais facilmente a chave e ler as mensagens. Na segunda guerra, a Alemanha distribuía entre seus postos militares livros com senhas diárias aleatórias, isto é, todo dia usavam senhas diferentes que não eram palavras em si.

Por mais que pareça uma solução simples, distribuir livros com chaves é um método bastante caro. O conjunto de chaves é trabalhoso e demorado para ser criado e os livros devem ser distribuídos com uma frequência alta. Na inglaterra, uma economia de 10% no custo da distribuição de chaves já era uma economia bastante considerável nas despesas militares.

O problema de distribuição de chaves se tornou ainda mais crítico com a invenção do computador e, posteriormente, da internet. A internet passou a ser usada para importantes transações bancárias e os custos da distribuição de chaves eram inviáveis.

O algoritmo RSA, descrito em 1978 por Ron Rivest, Adi Shamir e Leonard Adleman [11], é a mais famosa aplicação de Teoria dos Números e o primeiro sistema de criptografia a resolver convenientemente o problema de distribuição de chaves. Sua força e segurança se deve exatamente à dificuldade de fatoração de números grandes. O algoritmo depende do seguinte resultado

Teorema. *Se p e q são primos e $n = pq$, então*

$$m^{de} \equiv m \,(mod\, n) \quad \Leftrightarrow \quad de \equiv 1 \,(mod\, \Phi(n))$$

O conjunto (e, n) é conhecido por todos os usuários da rede e é usado para cripografar a mensagem m. A função de Euler $\Phi(n) = (p-1)(q-1)$ depende

dos fatores primos de n, ou seja, só pode ser calculada com o conhecimento de p e q. Por outro lado, sabendo-se $\Phi(n)$ é possível calcular d que é a chave para descriptografar a mensagem. Portanto, uma pessoa que só conheça e e n pode apenas criptografar as mensagens a fim de que o dono da chave d descriptografe.

Com isso, fica claro que uma pessoa que tenha o conhecimento de um método rápido para a fatoração de um inteiro n pode, ao interceptar uma mensagem criptografada, obter o conteúdo original da mensagem.

2.2 Por que quebrar o RSA?

Se fizermos uma rápida pesquisa pelo google, podemos encontrar facilmente sites que usam criptografia. Em particular, o próprio site de busca do google usa o RSA para troca de chaves. Bancos, e-mails, compras on-line e outras várias atividades na internet confiam no RSA para a troca segura de informação.

Por outro lado, a segurança que a criptografia trás permite que criminosos troquem informações na rede com a mesma privacidade online de uma transação bancária. Por este motivo, os órgãos de segurança tentam proibir o uso de criptografias fortes. De fato, eles são grandes empregadores de matemáticos com o objetivo de quebrar estas criptografias.

Para a Matemática, a fatoração também representa um papel teórico bastante importante: Uma vez que o problema de Fatorização está na classe $\mathcal{NP}$, resolvê-lo, no sentido de encontrar um algoritmo polinomial, ou apontar suas dificuldades são fortes indicadores de que $\mathcal{P} = \mathcal{NP}$ ou $\mathcal{P} \neq \mathcal{NP}$, respectivamente.

Um resultado muito animador relacionado ao problema de fatoração é o algoritmo *AKS* [2], que decide primalidade em tempo polinomial. Assim, é possível dizer em tempo polinomial se um número N dado tem fatoração diferente da trivial $(1 \times N)$ ou não. Porém, o AKS funciona de forma a decidir essa existência sem poder apresentá-la.

Capítulo 3

Problemas, Algoritmos e Reduções

Em geral temos uma noção intuitiva do que é um problema, um algoritmo e o que é o tempo de processamento de um algoritmo. Não pretendemos aqui apresentar uma definição formal destes objetos, é possível encontrar uma definição interessante em [12] juntamente com uma demonstração elegante do *Teorema de Cook*.

Informalmente um **Problema** é uma questão ou uma tarefa, por exemplo, *"Dado um grafo G, G tem um emparelhamento perfeito?"* ou *"Dado um grafo G, encontre o menor caminho que passa por todos os vértices"*.

Um **Algoritmo** é uma lista de instruções para se resolver um problema. É possível construir uma infinidade de algoritmos para cada problema, portanto vamos comparar os algoritmos pelo tempo necessário para se chegar à solução.

O **Tempo de Processamento** de um algoritmo será a quantidade de passos básicos necessários à resolução, onde um passo básico pode ser uma soma, produto, troca de 0 para 1 ou alguma outra operação simples.

Voltando ao primeiro exemplo, podemos observar que a dificuldade de encontrar um emparelhamento perfeito em um grafo G depende do tamanho de G, ou seja, da sua quantidade de arestas e vértices. O mesmo acontece com o tempo de processamento. Neste caso dizemos que o grafo G é uma *instância* do problema de Emparelhamento perfeito. Assim, dizemos que um algoritmo A resolve o problema P em *Tempo Polinomial* se existe um polinômio $p(x)$ tal que para toda instância de tamanho n de P a quantidade de passos básicos realizada por A é menor que $p(n)$.

Aqui será deixada uma brecha na teoria. As definições de *Tamanho de Instância* e *passo básico* foram deixadas muito vagas. De fato, são definições que dependem da codificação do problema e um aprofundamento nesta direção irá se afastar muito do nosso propósito. Podemos então entender um passo básico como uma operação que transforma o objeto que está sendo processado através de uma pequena mudança em sua estrutura. Portanto teremos que, um algoritmo polinomial A obtém, ao final de seu processo, um objeto com um tamanho polinomial

dependente do tamanho da instância inicial, ou seja, existe um polinômio $q(x)$ tal que, para toda instância inicial de tamanho n, o objeto final tem tamanho menor que $q(n)$.

Como já foi mencionado anteriormente, uma **Redução** é uma "transformação" de um problema P para um problema P'. Em outras palavras, uma redução R é um algoritmo que transforma instâncias de P em instâncias de P'. Se R é um algoritmo polinomial, então vamos dizer que R é uma **Redução em Tempo Polinomial** de P para P'.

Logo, não é difícil de perceber que, se existe um algoritmo polinomial A para P' e uma redução polinomial R de P para P', o algoritmo AR obtido pela aplicação de R e, em seguida, de A na instância inicial também será um algoritmo polinomial.

Capítulo 4

O Problema de Satisfabilidade

Este capítulo apresenta um pouco da teoria de Lógica Booleana e um desenvolvimento paralelo, interessante à redução final, sobre Funções Booleanas. Além disso, apresentamos também a primeira parte da redução, o que, portanto, torna este o capítulo principal desta dissertação.

Também chamado de SAT, o Problema de Satisfabilidade Booleana é um problema $\mathcal{NP}$-completo, o que significa que todo problema na classe $\mathcal{NP}$ pode ser reduzido em tempo polinomial para SAT. Este fato, devido ao *Teorema de Cook*, explica sua enorme importância para a Otimização Combinatória, uma vez que um algoritmo eficiente para o SAT implica em um algoritmo eficiente para todos os outros problemas em $\mathcal{NP}$.

As variáveis booleanas podem assumir os valores *Verdadeiro* (ou 1) e *Falso* (ou 0); Os operadores booleanos são $\vee$ (ou), $\wedge$ (e) e $\neg$ (negação) e respeitam as seguintes regras:

- $0 \vee 0 = 0$
- $0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$
- $1 \wedge 1 = 1$
- $0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0$
- $\neg 0 = \overline{0} = 1$
- $\neg 1 = \overline{1} = 0$

O conceito de **Expressão Booleana** pode ser definido indutivamente, da seguinte forma:

1) Seja x uma variável booleana, então x é uma expressão booleana;

2) Sejam E_1, E_2 expressões booleanas, então $E_1 \vee E_2$, $E_1 \wedge E_2$, $\neg E_1$ são expressões booleanas.

Para um estudo um pouco mais aprofundado veja [15].

Vamos dizer que uma expressão booleana E é **Satisfatível** se existe alguma atribuição de suas variáveis tal que E seja avaliada *Verdadeira*.

O problema de Satisfabilidade então é

Seja E uma expressão booleana, E é satisfatível?

4.1 Funções Booleanas

Noções Básicas

- Uma **Função Booleana** é uma função $f : \{0,1\}^n \longrightarrow \{0,1\}$.
- Seja $x \in \{0,1\}$, definimos $\overline{x} = 0 \Leftrightarrow x = 1$.
- Se $a = (a_1, \ldots, a_n) \in \{0,1\}^n$, definimos $\overline{a} = (b_1, \ldots, b_n) = (\overline{a_1}, \ldots, \overline{a_n})$.

Devemos introduzir neste momento duas funções básicas. Dado $a = (a_1, \ldots, a_n)$, queremos $m_a(x)$ e $s_a(x)$ duas funções tais que

$$m_a(x) = 1 \quad \Leftrightarrow \quad x = a$$

$$s_a(x) = 0 \quad \Leftrightarrow \quad x = a$$

Adotemos a seguinte notação: $x^1 = x$ e $x^0 = \overline{x}$. Observe que $x^x = 1$ e $x^{\overline{x}} = 0$ para todo $x \in \{0,1\}$. Portanto, para $x = (x_1, \ldots, x_n)$,

$$m_a(x) = \bigwedge x_i^{a_i} = x_1^{a_1} \wedge \ldots \wedge x_n^{a_n}$$

$$s_a(x) = \bigvee x_i^{\overline{a_i}} = x_1^{\overline{a_1}} \vee \ldots \vee x_n^{\overline{a_n}}$$

satisfazem a definição. Então chamaremos $m_a(x)$ o **Minterm** de a e $s_a(x)$ o **Maxterm** de a.

Em geral, temos que qualquer função booleana pode ser escrita em termos de Minterms e Maxterms, como propõe o seguinte teorema.

Teorema. *Seja $f(x)$ uma função booleana, então*

$$f(x) \quad = \quad \bigvee_{a \in f^{-1}(1)} m_a(x) \quad = \quad \bigwedge_{a \in f^{-1}(0)} s_a(x)$$

Estas representações são chamadas de Forma Normal Disjuntiva e Forma Normal Conjuntiva de f respectivamente (DNF e CNF).

Dualidade de Funções Booleanas

Definição. *Seja $f, f^* : \{0,1\}^n \longrightarrow \{0,1\}$ duas funções booleanas tais que $f^*(x) = \overline{f(\overline{x})}$. Dizemos que f^* é a função dual de f.*

Não é difícil de perceber que $f^{**} = f$,

$$f^{**}(x) = \overline{f^*(\overline{x})} = \overline{\overline{f(\overline{x})}} = f(x)$$

Definição. *Seja $f : \{0,1\}^n \longrightarrow \{0,1\}$ uma função booleana. Dizemos que f é **Auto-Dual** se para todo $a \in \{0,1\}^n$ vale*

$$f(a) = 1 \quad \Leftrightarrow \quad f(\overline{a}) = 0$$

Vamos ver que as funções auto-duais tem suas representações em DNF e CNF de um certo modo simétricas.

Algumas Funções Auto-Duais

- $Odd(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3) \vee (\overline{x_1} \wedge x_2 \wedge \overline{x_3})$

  A função Ímpar leva x em 1 se e somente se $\sum x_i$ é ímpar.

- $Atle2(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \overline{x_3}) \vee (x_1 \wedge \overline{x_2} \wedge x_3) \vee (\overline{x_1} \wedge x_2 \wedge x_3)$

  A função Pelo Menos 2 leva x em 1 se e somente se $\sum x_i \geq 2$.

- $Even(x_1, x_2, x_3) = \overline{Odd(x_1, x_2, x_3)}$

  A função Par.

- $Atmo1(x_1, x_2, x_3) = \overline{Atle2(x_1, x_2, x_3)}$

  A função No Máximo 1.

Não é coincidência o fato dessas duas últimas funções serem Auto-Duais. De fato, temos

Teorema. *f Auto-Dual se e somente se $\overline{f}$ Auto-Dual.*

Demonstração.

$$\overline{f}(x) = 1 \quad \Leftrightarrow \quad f(x) = 0 \quad \Leftrightarrow \quad f(\overline{x}) = 1 \quad \Leftrightarrow \quad \overline{f(\overline{x})} = 0$$

A volta é análoga. □

Lema. $m_a^* = s_{\overline{a}} \quad e \quad s_a^* = m_{\overline{a}}$

Demonstração.

$$m_a^*(x) = \overline{m_a(\overline{x})} = 0 \quad \Leftrightarrow \quad m_a(\overline{x}) = 1 \quad \Leftrightarrow \quad \overline{x} = a \quad \Leftrightarrow \quad x = \overline{a}$$

Portanto $m_a^* = s_{\overline{a}}$.

$$s_a^*(x) = \overline{s_a(\overline{x})} = 1 \quad \Leftrightarrow \quad s_a(\overline{x}) = 0 \quad \Leftrightarrow \quad \overline{x} = a \quad \Leftrightarrow \quad x = \overline{a}$$

Portanto $s_a^* = m_{\overline{a}}$.

□

Tome $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ e observe que

$$m_a(x) = x_1^{a_1} \wedge \ldots \wedge x_n^{a_n} \quad \text{e} \quad m_a^* = s_{\overline{a}} = x_1^{a_1} \vee \ldots \vee x_n^{a_n}$$

$$s_a(x) = x_1^{\overline{a_1}} \vee \ldots \vee x_n^{\overline{a_n}} \quad \text{e} \quad s_a^* = m_{\overline{a}} = x_1^{\overline{a_1}} \wedge \ldots \wedge x_n^{\overline{a_n}}$$

Então seja φ a função dualização, isto é $\varphi(f) = f^*$. Esta última observação nos nos dá indícios de que devemos ter $\varphi(\vee) = \wedge$ e $\varphi(\wedge) = \vee$ (isto é, $\varphi(g \vee h) = g \wedge h$ e $\varphi(g \wedge h) = g \vee h$). Este fato será demonstrado mais adiante, primeiramente vamos a um último resultado sobre funções auto-duais.

Lema. *f é Auto-Dual se e somente se $f^* = f$, ou seja, f é um ponto fixo de φ.*

Demonstração.

($\Rightarrow$) $f^*(x) = 1 \quad \Leftrightarrow \quad \overline{f(\overline{x})} = 1 \quad \Leftrightarrow \quad f(\overline{x}) = 0 \quad \Leftrightarrow \quad f(x) = 1$

Portanto $f^*(x) = 1 \quad \Leftrightarrow \quad f(x) = 1$, *logo* $f^* = f$.

($\Leftarrow$) $f(x) = 1 \quad \Leftrightarrow \quad f^*(x) = 1 \quad \Leftrightarrow \quad \overline{f(\overline{x})} = 1 \quad \Leftrightarrow \quad f(\overline{x}) = 0$

Logo, f é auto-dual.

□

Lema. Lema de Representação Dual

Seja φ a função dualização, então $\varphi(\vee) = \wedge$ e $\varphi(\wedge) = \vee$.

Demonstração. *Seja f uma função booleana. Observe que acontece um dos 4 casos, para g e h funções booleanas:*

1 $f = g \wedge h$

2 $f = g \vee h$

3 $f = x_i$

4 f *é constante*

Nos dois últimos casos não acontece $\wedge$ *nem* $\vee$*, portanto, suponha que* $f = g \wedge h$*. Queremos mostrar que* $f^* = g^* \vee h^*$*, então,*

$$\begin{aligned} \varphi(f)(x) &= f^*(x) = \overline{f(\overline{x})} = \overline{g(\overline{x}) \wedge h(\overline{x})} = \\ &= \overline{g(\overline{x})} \vee \overline{h(\overline{x})} = g^*(x) \vee h^*(x) \end{aligned}$$

O caso 2 é demonstrado analogamente.

A demonstração segue por indução no tamanho da expressão, isto é, g *e* h *são funções obviamente menores que* f*.* □

A partir deste resultado podemos observar o seguinte fato: Dada uma função em sua forma disjuntiva normal,

$$f = \bigvee_{a \in f^{-1}(1)} m_a$$

nós podemos encontrar a forma conjuntiva normal de f^* apenas por uma aplicação simples de φ, isto é, trocando $\vee$ por $\wedge$ e $\wedge$ por $\vee$. Da mesma forma, dada f em sua forma conjuntiva normal, encontramos f^* na forma disjuntiva normal.

Em particular, se f é auto-dual, então $f^* = f$ e, portanto, a aplicação de φ troca sua CNF por sua DNF e virse-versa.

Para finalizar, vamos introduzir um último tipo de função que aparecerá em nossas conclusões finais.

Definição. *Dizemos que uma função booleana é* ***Simétrica*** *se*

$$f(x) = f(\overline{x})$$

para todo x*.*

Por exemplo, seja $h(z, x)$ definida por

$$h(z, x) = 1 \quad \Leftrightarrow z = f(x)$$

onde f é uma função Auto-Dual.

Lembre-se que para uma função Auto-Dual vale $f(\overline{x}) = \overline{f(x)}$ e, portanto, temos que

$$h(\overline{z}, \overline{x}) = 1 \quad \Leftrightarrow \quad \overline{z} = f(\overline{x}) \quad \Leftrightarrow \quad \overline{z} = \overline{f(x)} \quad \Leftrightarrow \quad z = f(x) \quad \Leftrightarrow \quad h(z, x) = 1$$

portanto, h é uma função simétrica.

Esta construção será usada mais adiante para a primeira parte da redução.

Em particular, se escrevermos uma função simétrica h em sua CNF,

$$h(x) = \bigwedge_{a \in f^{-1}(0)} s_a(x)$$

temos também que $h(x) = \bigwedge_{a \in f^{-1}(0)} (s_a(x) \wedge s_{\overline{a}}(x))$, pois $h(a) = 0 \Leftrightarrow h(\overline{a}) = 0$. Isto é, se $s_a(x)$ é maxterm de h então $s_{\overline{a}}(x)$ também é.

4.2 A redução do problema de Fatoração para Satisfabilidade

A seguinte construção, proposta por Lins, S. (2010), é inspirada na Teoria de Circuitos Booleanos. É possível construir um circuito que computa $p \cdot q$ para quaisquer p e q dados [14]. Este circuito pode ser baseado no algoritmo escolar para a multiplicação. Por exemplo, ao tentar calcular 11×13 usando a base decimal obtemos

$$\begin{array}{cccc} & & 1 & 1 \\ \times & & 1 & 3 \\ \hline & & 3 & 3 \\ & 1 & 1 & \\ \hline & 1 & 4 & 3 \end{array}$$

porém, se estivermos usando um sistema binário, vamos obter

$$\begin{array}{cccccccc} & & & & 1 & 0 & 1 & 1 \\ \times & & & & 1 & 1 & 0 & 1 \\ \hline & & & & 1 & 0 & 1 & 1 \\ & & & 0 & 0 & 0 & 0 & \\ & & 1 & 0 & 1 & 1 & & \\ & 1 & 0 & 1 & 1 & & & \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

Assim, para cada número na tabela, colocaremos uma variável booleana, de forma que qualquer conjunto de atribuições que represente uma multiplicação terminando no número a ser fatorado leve a uma fatoração. Observe que existem, a princípio, pelo menos 4 atribuições distintas para cada número N, isto é, relativas a $1 \cdot N$, $p \cdot q$, $q \cdot p$, $N \cdot 1$.

A ideia, portanto, é escrever uma expressão booleana que seja satisfeita apenas por atribuições que representem multiplicações.

4.2.1 Preliminares

Para escrever uma expressão que compute a fatoração iremos fazer uso de subexpressões que computem relações intermediárias entre as variáveis. Essas subexpressões farão bastante uso de algumas das funções booleanas apresentadas anteriormente.

Sejam $R_1, \ldots, R_m$ relações entre as variáveis que devem ser satisfeitas. Para construirmos uma expressão booleana que seja satisfeita se somente se todas as relações forem satisfeitas vamos fazer

$$E = R_1 \wedge \cdots \wedge R_m$$

Por exemplo, seja E uma expressão que é satisfeita se e somente se $a = b$ e $c \neq d$ (ou seja, $c = \overline{d}$), onde a, b, c, d são variáveis binárias. Vamos fazer uso da função igualdade que será apresentada a seguir e satisfaz

$$I(x, y) = 1 \Leftrightarrow x = y$$

Não é difícil perceber que

$$E = I(a, b) \wedge I(c, \overline{d})$$

Além disso, para a última parte da redução, é interessante que todas as sub-expressões sejam apresentadas em forma conjuntiva, a fim de que a expressão final seja apresentada também em forma conjuntiva.

4.2.2 Igualdades

Para utilizar as expressões booleanas que vimos nos exemplos anteriores é necessário definir uma expressão geral que represente a igualdade. Isto é, uma função I tal que $I(x, y) = 1 \Leftrightarrow x = y$. Podemos, a fim de encontrar a expressão final em forma próxima a uma forma conjuntiva, construir a expressão igualdade $I(x, y)$ pela sua CNF. Sabemos que $I(x, y) = 0 \Leftrightarrow (x, y) = (1, 0)$ ou $(x, y) = (0, 1)$. Logo,

$$I(x, y) = (\overline{x} \vee y) \wedge (x \vee \overline{y})$$

Observe que x e y podem ser substituídos por expressões, a fim de que queiramos igualá-las.

No caso em que uma variável, digamos x, deve sempre assumir o valor 0 ou 1, a expressão fica um pouco mais simples:

$$I(x, 0) = \overline{x} \qquad I(x, 1) = x$$

4.2.3 Somas

Neste tópico procuraremos construir uma expressão $S(A, B, Z)$ que seja satisfeita se e somente se $A + B = Z$. Para isso, iremos primeiro construir uma expressão para somar 3 bits.

A soma de 3 bits devolve, em geral, um número com 2 bits. Portanto queremos uma função $g : \{0, 1\}^3 \longrightarrow \{0, 1\}^2$. Sabemos que dados $a, b, c \in \{0, 1\}$ existem únicos $x, y \in \{0, 1\}$ tais que $a + b + c = 2x + y$. Assim, iremos escrever $g(a, b, c) = (x, y)$. Então observe que temos que $x = 1$ se e somente se $a + b + c \geq 2$ e $y = 1$ se e somente se $a + b + c$ é ímpar. Assim,

$$g(a, b, c) = (Atle2(a, b, c), Odd(a, b, c))$$

Portanto, adicionaremos à expressão geral uma sub-expressão que represente $g(a,b,c) = (x,y) = (Atle2(a,b,c), Odd(a,b,c))$, ou seja,

$$I(x, Atle2(a,b,c)) \wedge I(y, Odd(a,b,c))$$

porém, esta sub-expressão não está em forma conjuntiva. Podemos expandí-la a fim de obter uma forma conjuntiva.

$$\begin{aligned}
&I(x, Atle2(a,b,c)) \\
&= (\overline{x} \vee Atle2(a,b,c)) \wedge (x \vee \overline{Atle2(a,b,c)}) \\
&= (\overline{x} \vee Atle2(a,b,c)) \wedge (x \vee Atmo1(a,b,c)) \\
&= (\overline{x} \vee ((a \vee b \vee c) \wedge (a \vee b \vee \overline{c}) \wedge (a \vee \overline{b} \vee c) \wedge (\overline{a} \vee b \vee c))) \\
&\wedge (x \vee ((\overline{a} \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee b \vee \overline{c}) \wedge (a \vee \overline{b} \vee \overline{c}))) \\
&= (\overline{x} \vee a \vee b \vee c) \wedge (\overline{x} \vee a \vee b \vee \overline{c}) \wedge (\overline{x} \vee a \vee \overline{b} \vee c) \wedge (\overline{x} \vee \overline{a} \vee b \vee c) \\
&\wedge (x \vee \overline{a} \vee \overline{b} \vee \overline{c}) \wedge (x \vee \overline{a} \vee \overline{b} \vee c) \wedge (x \vee \overline{a} \vee b \vee \overline{c}) \wedge (x \vee a \vee \overline{b} \vee \overline{c})
\end{aligned}$$

$$\begin{aligned}
&I(x, Odd(a,b,c)) \\
&= (\overline{y} \vee Odd(a,b,c)) \wedge (y \vee \overline{Odd(a,b,c)}) \\
&= (\overline{y} \vee Odd(a,b,c)) \wedge (y \vee Even(a,b,c)) \\
&= (\overline{y} \vee ((a \vee b \vee c) \wedge (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{c}) \wedge (\overline{a} \vee \overline{b} \vee c))) \\
&\wedge (y \vee ((\overline{a} \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee c) \wedge (a \vee \overline{b} \vee c) \wedge (a \vee b \vee \overline{c}))) \\
&= (\overline{y} \vee a \vee b \vee c) \wedge (\overline{y} \vee a \vee \overline{b} \vee \overline{c}) \wedge (\overline{y} \vee \overline{a} \vee b \vee \overline{c}) \wedge (\overline{y} \vee \overline{a} \vee \overline{b} \vee c) \\
&\wedge (y \vee \overline{a} \vee \overline{b} \vee \overline{c}) \wedge (y \vee \overline{a} \vee b \vee c) \wedge (y \vee a \vee \overline{b} \vee c) \wedge (y \vee a \vee b \vee \overline{c})
\end{aligned}$$

É possível encontrar outras formas conjuntivas equivalentes para estas expressões, estas contém 16 cláusulas, onde cada cláusula contém 4 variáveis.

Podemos então, começar a somar números maiores. Ao tentar somar dois números de 2 bits (x_1, x_0) e (y_1, y_0),

$$\begin{array}{cccc}
 & c_0 & & \\
 & x_1 & x_0 & \\
+ & y_1 & y_0 & \\
\hline
z_2 & z_1 & z_0 &
\end{array}$$

nos deparamos com um problema de representação. Para obter z_0, basta termos x_0 e y_0, e portanto $z_0 = Odd(x_0, y_0, 0)$. Porém, para obter z_1, iremos precisar da informação dada por $Atle2(x_0, y_0, 0)$, o que nos dará a sub-expressão $z_1 = Odd(x_1, y_1, Atle2(x_0, y_0, 0))$. Não é difícil perceber que, desta forma, se aumentarmos o número de bits dos números que queremos somar, iremos aumentar demasiadamente o tamanho da expressão final. Para evitar este problema, iremos introduzir uma nova variável, $c_0 = Atle2(x_0, y_0, 0)$, chamada de **Carry**, e com isso vamos trabalhar com

$$I(z_0, Odd(x_0, y_0, 0)) \wedge I(c_0, Atle2(x_0, y_0, 0)) \wedge I(z_1, Odd(x_1, y_1, c_0))$$

para concluir, adicionamos $\wedge I(z_2, Atle2(x_1, z_1, c_0))$.

Para generalizar, iremos adicionar no lugar desta última expressão a seguinte expressão equivalente

$$I(c_1, Atle2(x_1, y_1, c_0)) \wedge I(z_2, Odd(0, 0, c_1))$$

Portanto, observando o diagrama para a soma de dois números de n bits

$$\begin{array}{cccccccc} & c_{n-1} & c_{n-2} & c_{n-3} & \dots & c_0 & 0 \\ & 0 & x_{n-1} & x_{n-2} & \dots & x_1 & x_0 \\ + & 0 & y_{n-1} & y_{n-2} & \dots & y_1 & y_0 \\ \hline & z_n & z_{n-1} & z_{n-2} & \dots & z_1 & z_0 \end{array}$$

podemos construir as equações a partir de uma fórmula geral:

$$\begin{cases} c_{-1} = x_n = y_n = 0 \\ z_i = Odd(x_i, y_i, c_{i-1}) \\ c_i = Atle2(x_i, y_i, c_{i-1}) \end{cases}$$

Então, se $A = (x_{n-1}, \dots, x_0), B = (y_{n-1}, \dots, y_0), Z = (z_n, \dots, z_0)$, temos

$$S(A, B, Z) = (I(c_{-1}, 0) \wedge I(x_n, 0) \wedge I(y_n, 0)) \wedge$$

$$\bigwedge_{i=0}^{n} (I(z_i, Odd(x_i, y_i, c_{i-1})) \wedge I(c_i, Atle2(x_i, y_i, c_{i-1})))$$

4.2.4 Produtos

Como fizemos anteriormente, vamos introduzir uma função que multiplique bits e utilizá-la para números maiores. Neste caso precisaremos de uma função $Mult$ que multiplique dois bits, ou seja, $Mult(a, b) = 1 \Leftrightarrow a * b = 1$.

Temos que $a * b = 1 \Leftrightarrow a = b = 1$ e, portanto,

$$Mult(a, b) = (\overline{a} \vee b) \wedge (a \vee \overline{b}) \wedge (a \vee b)$$

A sub-expressão que será inclusa na expressão final é $I(z, Mult(a, b))$, que em forma conjuntiva fica

$$\begin{aligned} & I(z, Mult(a, b)) \\ &= (\overline{z} \vee Mult(a, b)) \wedge (z \vee \overline{Mult(a, b)}) \\ &= (\overline{z} \vee ((\overline{a} \vee b) \wedge (a \vee \overline{b}) \wedge (a \vee b))) \wedge (z \vee (\overline{a} \vee \overline{b})) \\ &= (\overline{z} \vee \overline{a} \vee b) \wedge (\overline{z} \vee a \vee \overline{b}) \wedge (\overline{z} \vee a \vee b) \wedge (z \vee \overline{a} \vee \overline{b}) \\ &= (\overline{z} \vee b) \wedge (\overline{z} \vee a) \wedge (z \vee \overline{a} \vee \overline{b}) \end{aligned}$$

Assim, dados dois números de n bits $(x_{n-1}, \ldots, x_1, x_0), (y_{n-1}, \ldots, y_1, y_0)$, queremos respeitar o diagrama de multiplicação escolar,

$$\begin{array}{cccccccc} & & & & x_{n-1} & \cdots & x_1 & x_0 \\ \times & & & & y_{n-1} & \cdots & y_1 & y_0 \\ \hline & & & & x_{n-1}y_0 & \cdots & x_1y_0 & x_0y_0 \\ & & & x_{n-1}y_1 & \cdots & x_1y_1 & x_0y_1 & \\ & & \cdots & \cdots & \cdots & \cdots & & \\ & x_{n-1}y_{n-1} & \cdots & x_1y_{n-1} & x_0y_{n-1} & & & \end{array}$$

portanto, vamos criar variáveis $z_{k,i,j}$ onde $z_{0,i,j} = Mult(x_{j-i}, y_i)$.

Obtemos o seguinte diagrama

$$\begin{array}{cccccccc} & & & & x_{n-1} & \cdots & x_1 & x_0 \\ \times & & & & y_{n-1} & \cdots & y_1 & y_0 \\ \hline & & & & z_{0,0,n-1} & \cdots & z_{0,0,1} & z_{0,0,0} \\ & & & z_{0,1,n} & \cdots & z_{0,1,2} & z_{0,1,1} & \\ & & \cdots & \cdots & \cdots & \cdots & & \\ & z_{0,n-1,2n-2} & \cdots & z_{0,n-1,n} & z_{0,n-1,n-1} & & & \end{array}$$

Observe que i e j indexam linhas e colunas do diagrama. Iremos preencher os espaços em branco com variáveis nulas a fim de facilitar a indexação para as somas. O índice $k = 1$ será usado para indexar os resultados das somas. Portanto, definimos

$$z_{0,i,j} = \begin{cases} Mult(x_{j-i}, y_i) & \text{se } i \leq j \leq n - 1 + i \\ 0 & \text{se } j < i \text{ ou } j > n - 1 + i \end{cases}$$

Assim, obtemos uma matriz de números que devem ser somados a fim de obter o resultado final.

$$\begin{matrix} z_{0,0,2n-2} & \cdots & z_{0,0,0} \\ \vdots & \ddots & \vdots \\ z_{0,n-1,2n-2} & \cdots & z_{0,n-1,0} \end{matrix}$$

Defina $Z_{k,i} = (z_{k,i,2n-2}, \ldots, z_{k,i,0})$, que, para $k = 0$, é o número dado pela $i-$ésima linha da matriz, que também é, por outro lado, $2^i \cdot y_i \cdot (x_{n-1}, \ldots, x_0)$.

Faremos a seguinte lista de somas

$$\begin{cases} S(Z_{0,0}, Z_{0,1}, Z_{1,1}) & \\ S(Z_{1,i}, Z_{0,i+1}, Z_{1,i+1}) & i = 1, \ldots, n-2 \end{cases} \qquad \left(\begin{array}{l} Z_{1,1} = Z_{0,0} + Z_{0,1} \\ Z_{1,i+1} = Z_{1,i} + Z_{0,i+1} \end{array} \right)$$

Vamos ter que $Z_{1,i+1}$ é a soma de $Z_{1,i}$ com $Z_{0,i+1}$, e com isso temos que $Z_{1,n-1} = \sum_{i=0}^{n-1} Z_{0,i}$. Portanto $Z_{1,n-1} = x \cdot y$.

Assim, obtemos uma expressão booleana que é satisfeita se e somente $Z_{1,n-1} = x \cdot y$. Portanto, dado um número $Z^* = (z^*_{2n-2}, \ldots, z^*_0)$, podemos forçar que $Z_{1,n-1}$ receba esses valores anexando $z^{z^*_j}_{1,n-1,j}, j = 0, \ldots, 2n-2$ à expressão.

Observe que

$$\bigwedge_{j=0}^{2n-2} z^{z^*_j}_{1,n-1,j} = 1 \qquad \Leftrightarrow \qquad z_{1,n-1,j} = z^*_j \quad j = 1, \ldots, 2n-2$$

Exemplo. *Vamos encontrar a expressão booleana para fatorar* $6 = 110$.

$$\begin{array}{r} X \\ Y \\ \hline Z_{0,0} \\ Z_{0,1} \\ \hline Z_{1,0} \end{array} \qquad \Leftrightarrow \qquad \begin{array}{ccc} & x_1 & x_0 \\ & y_1 & y_0 \\ \hline z_{0,0,2} & z_{0,0,1} & z_{0,0,0} \\ z_{0,1,2} & z_{0,1,1} & z_{0,1,0} \\ \hline z_{1,0,2} & z_{1,0,1} & z_{1,0,0} \end{array}$$

A equação, resumidamente, é

$$E_6(x_0, x_1, y_0, y_1, z_{0,0,0}, z_{0,0,1}, z_{0,0,2}, z_{0,1,0}, z_{0,1,1}, z_{0,1,2}, z_{1,0,0}, z_{1,0,1}, z_{1,0,2}) =$$

$$= I(z_{0,0,0}, Mult(x_0, y_0)) \wedge I(z_{0,0,1}, Mult(x_1, y_0)) \wedge I(z_{0,0,2}, 0) \wedge$$

$$\wedge I(z_{0,1,0}, 0) \wedge I(z_{0,1,1}, Mult(x_0, y_1)) \wedge I(z_{0,1,2}, Mult(x_1, y_1)) \wedge$$

$$\wedge S(Z_{0,0}, Z_{0,1}, Z_{1,0})$$

$$\wedge z_{1,0,2} \wedge z_{1,0,1} \wedge \overline{z_{1,0,0}}$$

Exemplo. *Vamos encontrar a expressão para fatorar* $15 = 01111$.

As variáveis respeitam o seguinte diagrama:

X			x_2	x_1	x_0
Y			y_2	y_1	y_0
$Z_{0,0}$	$\Leftrightarrow$	$z_{0,0,4}$	$z_{0,0,3}$	$z_{0,0,2}$	$z_{0,0,1}$	$z_{0,0,0}$
$Z_{0,1}$		$z_{0,1,4}$	$z_{0,1,3}$	$z_{0,1,2}$	$z_{0,1,1}$	$z_{0,1,0}$
$Z_{0,2}$		$z_{0,2,4}$	$z_{0,2,3}$	$z_{0,2,2}$	$z_{0,2,1}$	$z_{0,2,0}$
$Z_{1,1}$		$z_{1,1,4}$	$z_{1,1,3}$	$z_{1,1,2}$	$z_{1,1,1}$	$z_{1,1,0}$

As somas parciais respeitam o seguinte diagrama:

$$\left.\begin{matrix} Z_{0,0} \\ Z_{0,1} \\ Z_{0,2} \end{matrix}\right\} \; Z_{1,0} \; \Bigg\} \; Z_{1,1}$$

Portanto, a equação para fatorar 15 *é*

$$E_{15}(x_0, x_1, x_2, y_0, y_1, y_2, z_{0,0,4}, z_{0,0,3}, z_{0,0,2}, z_{0,0,1}, z_{0,0,0},$$

$$z_{0,1,4}, z_{0,1,3}, z_{0,1,2}, z_{0,1,1}, z_{0,1,0}, z_{0,2,4}, z_{0,2,3}, z_{0,2,2}, z_{0,2,1}, z_{0,2,0},$$

$$z_{1,0,4}, z_{1,0,3}, z_{1,0,2}, z_{1,0,1}, z_{1,0,0}, z_{1,1,4}, z_{1,1,3}, z_{1,1,2}, z_{1,1,1}, z_{1,1,0}) =$$

$$I(z_{0,0,0}, Mult(x_0, y_0)) \wedge I(z_{0,0,1}, Mult(x_1, y_0)) \wedge I(z_{0,0,2}, Mult(x_2, y_0)) \wedge$$

$$\wedge I(z_{0,0,3}, 0) \wedge I(z_{0,0,4}, 0) \wedge I(z_{0,1,0}, 0) \wedge$$

$$\wedge I(z_{0,1,1}, Mult(x_0, y_1)) \wedge I(z_{0,1,2}, Mult(x_1, y_1)) \wedge I(z_{0,1,3}, Mult(x_2, y_1) \wedge$$

$$\wedge I(z_{0,1,4}, 0) \wedge I(z_{0,2,0}, 0) \wedge I(z_{0,2,1}, 0) \wedge$$

$$\wedge I(z_{0,2,2}, Mult(x_0, y_2)) \wedge I(z_{0,2,3}, Mult(x_1, y_2)) \wedge I(z_{0,2,4}, Mult(x_2, y_2)) \wedge$$

$$\wedge S(Z_{0,0}, Z_{0,1}, Z_{1,0}) \wedge S(Z_{1,0}, Z_{0,2}, Z_{1,1}) \wedge$$

$$\wedge \overline{z_{1,1,4}} \wedge z_{1,1,3} \wedge z_{1,1,2} \wedge z_{1,1,1} \wedge z_{1,1,0}$$

Capítulo 5

Programação 0-1

Programação Inteira é um problema bastante importante na área de Combinatória e tem várias aplicações na indústria. Em termos de classes de complexidade, Programação Inteira é um programa $\mathcal{NP}$-Completo.

Existem vários algoritmos, tanto comerciais quanto livres, eficientes para resolver problemas de programação inteira, como por exemplo o *SCIP* [1], o *GLPK* [10], porém, nenhum deles é de tempo polinomial. Estes algoritmos utilizam, mais geralmente, duas técnicas combinadas: o *Branch and Bound* e os *Cortes de Gomory*.

Dada uma matriz A inteira $m \times n$, um vetor c inteiro com n entradas e um vetor b inteiro com m entradas, podemos definir um problema de programação inteira por

$$\begin{aligned} \text{maximizar} \quad & c^T x \\ \text{sujeito a} \quad & Ax \leq b \\ \text{e} \quad & x \text{ inteiro} \end{aligned}$$

Observando c e x como

$$c = (c_1, \ldots, c_n), \qquad x = (x_1, \ldots, x_n)$$

dizemos que $c^T x = c_1 x_1 + \ldots + c_n x_n$ é a *função objetivo*, a função que queremos maximizar, onde $Ax \leq b$ e o "x inteiro" são as restrições. Podemos ver A como um vetor de linhas a_i e b como uma coluna

$$A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}, \qquad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

e teremos para cada $i \in \{1, \ldots, m\}$ uma restrição $a_{i1} x_1 + \ldots + a_{in} x_n \leq b_i$. Além disso, "$x$ inteiro" quer dizer que x_j é inteiro para todo $j \in (1, \ldots n)$.

Um problema de Programação 0-1 é um problema de Programação Inteira onde x é um vetor binário.

5.1 A Redução do Problema de Satisfabilidade para o Problema de Programação 0-1

Este tópico apresenta a segunda parte da redução proposta nesta dissertação. A redução aqui apresentada é uma redução bastante tradicional e simples.

Queremos, à partir de uma expressão booleana em forma conjuntiva, obter um sistema linear de desigualdades tal que o conjunto de soluções inteiras do sistema linear esteja em bijeção com o conjunto de soluções da expressão booleana. Aqui usamos o termo *solução da expressão booleana* no sentido de conjunto de atribuições que satisfazem a expressão.

Seja γ uma aplicação tal que, para A e B expressões booleanas,

- $\gamma(x) = x$
- $\gamma(\overline{x}) = (1 - x)$
- $\gamma(A \vee B) = \gamma(A) + \gamma(B)$
- $\gamma(A \wedge B) = \gamma(A)\gamma(B)$

Observe que estamos construindo as equações lineares com as mesmas variáveis das equações booleanas, a fim de facilitar o entendimento. Portanto, estamos usando $x \in \{0, 1\}$.

Vamos, primeiramente, apresentar um exemplo:

Exemplo. *Seja* $E = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_3} \vee x_4)$.

Temos que

$$\begin{aligned}
\gamma(E) &= \gamma[(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_3} \vee x_4)] \\
&= \gamma(x_1 \vee x_2 \vee x_3)\gamma(x_2 \vee x_3 \vee \overline{x_4})\gamma(\overline{x_3} \vee x_4) \\
&= [\gamma(x_1) + \gamma(x_2) + \gamma(x_3)][\gamma(x_2) + \gamma(x_3) + \gamma(\overline{x_4})][\gamma(\overline{x_3}) + \gamma(x_4)] \\
&= [x_1 + x_2 + x_3][x_2 + x_3 + (1 - x_4)][(1 - x_3) + x_4]
\end{aligned}$$

Vamos ver que para E ser satisfeita devemos ter $\gamma(E) \geq 1$ e, para isso, iremos resolver o seguinte sistema gerado a partir da expressão obtida acima:

$$\begin{cases}
x_1 + x_2 + x_3 \geq 1 \\
x_2 + x_3 + 1 - x_4 \geq 1 \\
1 - x_3 + x_4 \geq 1
\end{cases}$$

A seguir demonstramos a equivalência desta redução, isto é, mostramos que, usando as mesmas variáveis (como observado acima), o conjunto de pontos que satisfaz a expressão booleana dada é o mesmo conjunto de pontos que satisfaz o sistema de desigualdades obtido.

Considere uma cláusula unitária, digamos $C = x$. Para que C seja satisfeita é necessário e suficiente que $x = 1$. Em termos de desigualdades lineares, queremos satisfazer $x \geq 1$, ou seja, $\gamma(C) \geq 1$. Observe que, como x pertence a $\{0, 1\}$, $x \geq 1$ implica $x = 1$. Por outro lado, se $x < 1$, então, $x = 0$ e C não é satisfeita.

Para $C = \overline{x}$ a construção é análoga. C é satisfeita se e somente se $x = 0$. Queremos satisfazer a desigualdade linear $(1 - x) \geq 1$, ou seja $\gamma(C) \geq 1$.

Observe que para qualquer cláusula unitária C vale $\gamma(C) \geq 0$, uma vez que x pertence a $\{0, 1\}$. Assim, $\sum \gamma(x_i^{a_i}) \geq 0$. Logo, se $\sum \gamma(x_i^{a_i}) \geq 1$, então $\gamma(x_i^{a_i}) \geq 1$ para algum i, digamos $i = i_0$, e, portanto, a cláusula $x_{i_0}^{a_{i_0}}$ é satisfeita. Por outro lado, se $\sum \gamma(x_i^{a_i}) < 1$, então $\sum \gamma(x_i^{a_i}) = 0$ e $\gamma(x_i^{a_i}) = 0$ para todo i.

Seja então, $C = \bigvee x_i^{a_i}$. Para que C seja satisfeita é suficiente que $x_i^{a_i}$ seja satisfeita para pelo menos um i. Logo, pela observação acima, basta que $\gamma(C) = \sum \gamma(x_i^{a_i}) \geq 1$. Observe que se $x_i^{a_i}$ é satisfeita para um único i, então $\gamma(C) = 1$.

Uma expressão booleana E em forma conjuntiva é escrita como $E = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, onde $C_i = x_{i_1} \vee x_{i_2} \vee \cdots \vee c_{i_{p_i}}$ são cláusulas. Para E ser satisfeita, todas as cláusulas C_i devem ser satisfeitas. Em outras palavras, $\gamma(C_i) \geq 1$ para todo i.

Observe que é exatamente isso que fazemos quando exigimos $\gamma(E) \geq 1$. Temos que

$$\gamma(E) = \gamma(\bigwedge C_i) = \prod \gamma(C_i) \geq 1$$

Já vimos que $\gamma(C_i) \geq 0$ para todo i, logo, se $\gamma(E) = 0$, então $\gamma(C_i) = 0$ para pelo menos um i. Por outro lado, é óbvio que se $\gamma(C_i) = 0$ para algum i, então $\gamma(E) = 0$.

Assim, temos que

$$E = 1 \quad \Leftrightarrow \quad \gamma(E) \geq 1 \quad \Leftrightarrow \quad \gamma(C_i) \geq 1 \ \forall i$$

Com isso, iremos obter o seguinte sistema de desigualdades:

$$\begin{cases} \gamma(C_1) & \geq & 1 \\ \gamma(C_2) & \geq & 1 \\ & \vdots & \\ \gamma(C_m) & \geq & 1 \end{cases}$$

Para finalizar, deixamos as variáveis de um único lado das desigualdades e as

constantes de outro. Retornando ao exemplo, teremos:

$$\begin{cases} x_1 + x_2 + x_3 \geq 1 \\ x_2 + x_3 - x_4 \geq 0 \\ - x_3 + x_4 \geq 0 \end{cases}$$

e, portanto, obtemos um sistema de inequações lineares, a instância de um programa inteiro. Com isso podemos usar métodos de programação inteira para resolver o nosso problema.

Capítulo 6

Particularidades da Redução Geral

Este capítulo é dedicado a apresentação de algumas particularidades desta redução, propriedades observadas que podem ser utilizadas para o desenvolvimento de algoritmos mais eficientes.

6.1 Unicidade da Solução

A primeira propriedade é que, para a construção da expressão booleana, o número de variáveis usadas para representar cada fator primo do número a ser fatorado é menor do que o número de variáveis usadas para representar o número a ser fatorado. Ou seja, nenhum dos fatores pode ser igual ao número a ser fatorado. Portanto, nenhuma das soluções da expressão pode gerar $1 \cdot N$ ou $N \cdot 1$.

Podemos também, introduzir dois novos conjuntos de variáveis $S_1 = (s_{1,0}, \ldots, s_{1,n-1})$ e $S_2 = (s_{2,0}, \ldots, s_{2,n-1})$ e a sub-expressão $S(1, S_1, S_2) \wedge S(S_2, X, Y)$. Com isso temos $S_2 = S_1 + 1$ e $Y = X + S_2$, e, portanto, $S_2 \geq 1$ e $Y \geq X + 1 > X$.

Alternativamente, podemos introduzir somente $S = (s_0, \ldots, s_{n-1})$ e uma sub-expressão $S(X, S, Y)$, obtendo $X + S = Y$ e, então, redefinir o primeiro carry desta sub-expressão como 1, obtendo $X + S + 1 = Y$.

Assim, no caso onde $N = p \cdot q$ com $p \neq q$ primos, vamos obter uma expressão booleana com uma única solução. Em termos de Programação $0 - 1$, obtemos um sistema de equações lineares com uma única solução inteira. Se olharmos para o relaxamento do programa inteiro para Programação Linear, vamos obter um politopo com um único ponto inteiro.

6.2 O Paralelismo das Equações

A segunda parte da redução nos fornece, para cada inteiro N a ser fatorado, uma matriz A_N e um vetor b_N tal que dado x^* inteiro satisfazendo $A_N \cdot x^* \geq b_N$,

podemos obter p e q tais que $p \cdot q = N$, assim, fatorando N.

Vamos definir a partir do par (A_N, b_N) o *Politopo de Fatoração de N*

$$\mathbb{P}_N = \{x \in \mathbb{R}^n \| A_N \cdot x \geq b_N\}$$

Quando necessário, daremos mais informações sobre o politopo. Aqui, queremos apresentar o paralelismo de um conjunto importante de hiperplanos que o definem.

Lembre-se do início do texto em que definimos funções *Simétricas*.

Definição. *f é Simétrica se vale*

$$f(x) = f(\overline{x})$$

para todo x.

Vamos ter, portanto, que as funções $I_{Atleast2}(x,a,b,c) = I(x, Atle2(a,b,c))$ e $I_{Odd}(y,a,b,c) = I(y, Odd(a,b,c))$ são simétricas, pois Odd e $Atle2$ são funções *Auto-Duais* - Observe o exemplo dado após a definição de função simétrica - Vamos aplicar a função γ definida na segunda parte da redução para construir as submatrizes relativas a estas funções.

Lembre-se que

$$I_{Atleast2}(x,a,b,c) = (\overline{x} \vee a \vee b \vee c) \wedge (\overline{x} \vee a \vee b \vee \overline{c}) \wedge (\overline{x} \vee a \vee \overline{b} \vee c) \wedge (\overline{x} \vee \overline{a} \vee b \vee c)$$

$$\wedge (x \vee \overline{a} \vee \overline{b} \vee \overline{c}) \wedge (x \vee \overline{a} \vee \overline{b} \vee c) \wedge (x \vee \overline{a} \vee b \vee \overline{c}) \wedge (x \vee a \vee \overline{b} \vee \overline{c})$$

$$= (\overline{x} \vee a \vee b \vee c) \wedge (\overline{x} \vee a \vee b) \wedge (\overline{x} \vee a \vee c) \wedge (\overline{x} \vee b \vee c)$$

$$\wedge (x \vee \overline{a} \vee \overline{b} \vee \overline{c}) \wedge (x \vee \overline{a} \vee \overline{b}) \wedge (x \vee \overline{a} \vee \overline{c}) \wedge (x \vee \overline{b} \vee \overline{c})$$

$$= (\overline{x} \vee a \vee b) \wedge (\overline{x} \vee a \vee c) \wedge (\overline{x} \vee b \vee c)$$

$$\wedge (x \vee \overline{a} \vee \overline{b}) \wedge (x \vee \overline{a} \vee \overline{c}) \wedge (x \vee \overline{b} \vee \overline{c})$$

$$I_{Odd}(y,a,b,c) = (\overline{y} \vee a \vee b \vee c) \wedge (\overline{y} \vee a \vee \overline{b} \vee \overline{c}) \wedge (\overline{y} \vee \overline{a} \vee b \vee \overline{c}) \wedge (\overline{y} \vee \overline{a} \vee \overline{b} \vee c)$$

$$\wedge (y \vee \overline{a} \vee \overline{b} \vee \overline{c}) \wedge (y \vee \overline{a} \vee b \vee c) \wedge (y \vee a \vee \overline{b} \vee c) \wedge (y \vee a \vee b \vee \overline{c})$$

Portanto, temos

$$\gamma(I_{Atleast2}(x,a,b,c)) \geq 1$$
$$\Downarrow$$
$$\begin{pmatrix} -1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \\ -1 & 0 & 1 & 1 \\ 1 & -1 & -1 & 0 \\ 1 & -1 & 0 & -1 \\ 1 & 0 & -1 & -1 \end{pmatrix} \begin{pmatrix} x \\ a \\ b \\ c \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

$$\gamma(I_{Odd}(y,a,b,c)) \geq 1$$
$$\Downarrow$$
$$\begin{pmatrix} -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} y \\ a \\ b \\ c \end{pmatrix} \geq \begin{pmatrix} 0 \\ -2 \\ -2 \\ -2 \\ -2 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Observe que, como essas duas funções são simétricas, os maxterms delas estão em pares $s_a(x)$ e $s_{\overline{a}}(x)$. Com isso temos que as linhas de suas matrizes estão em pares onde uma linha é a negativa da outra. Portanto, podemos reapresentar as desigualdades da seguinte forma

$$\gamma(I_{Atleast2}(x,a,b,c)) \geq 1$$
$$\Downarrow$$
$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \geq \begin{pmatrix} -1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \\ -1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ a \\ b \\ c \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\gamma(I_{Odd}(y,a,b,c)) \geq 1$$
$$\Downarrow$$
$$\begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \end{pmatrix} \geq \begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} y \\ a \\ b \\ c \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Essas duas funções aparecerão sempre juntas, então iremos usar uma única

matriz para representá-las:

$$\mathbb{A} = \begin{pmatrix} -1 & 0 & 1 & 1 & 0 \\ -1 & 0 & 1 & 0 & 1 \\ -1 & 0 & 0 & 1 & 1 \\ 0 & -1 & 1 & 1 & 1 \\ 0 & 1 & -1 & 1 & 1 \\ 0 & 1 & 1 & -1 & 1 \\ 0 & 1 & 1 & 1 & -1 \end{pmatrix}$$

Logo,

$$\gamma(I_{Atleast2}(x,a,b,c) \wedge I_{Odd}(y,a,b,c)) \geq 1$$
$$\Downarrow$$
$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix} \geq \mathbb{A} \cdot \begin{pmatrix} x \\ y \\ a \\ b \\ c \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Assim, temos que o Politopo de Fatoração é limitado em grande parte por hiperplanos paralelos, gerados por estas funções simétricas.

Além disso, esta matriz $\mathbb{A}$ possui linhas emparelhadas. Vamos ver isso aplicando $\mathbb{A}$ a todos os possíveis valores para (a,b,c,x,y)

	a	b	c	$x = Atle2(a,b,c)$	$y = Odd(a,b,c)$
x_0	0	0	0	0	0
x_1	0	0	1	0	1
x_2	0	1	0	0	1
x_3	1	0	0	0	1
x_4	0	1	1	1	0
x_5	1	0	1	1	0
x_6	1	1	0	1	0
x_7	1	1	1	1	1

Temos então

$\mathbb{A} \cdot x_0$	$\mathbb{A} \cdot x_1$	$\mathbb{A} \cdot x_2$	$\mathbb{A} \cdot x_3$	$\mathbb{A} \cdot x_4$	$\mathbb{A} \cdot x_5$	$\mathbb{A} \cdot x_6$	$\mathbb{A} \cdot x_7$
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1
0	0	0	0	2	2	2	2
0	2	2	0	2	0	0	2
0	2	0	2	0	2	0	2
0	0	2	2	0	0	2	2

Aqui podemos observar o emparelhamento das linhas da matriz $\mathbb{A}$. Sempre que a última linha de $\mathbb{A} \cdot x_i$ assume 0, a primeira linha também assume 0 e sempre que a última linha assume 2, a primeira linha assume 1. O mesmo acontece com as linhas 2 e 6 e 3 e 5.

6.3 O Método dos Cortes Paralelos

O problema de Programação Inteira é um problema intimamente ligado ao problema de programação linear. Dado um problema de programação inteira P

$$\text{Max } cx, \quad x \in S = \{x \,|\, Ax \geq b,\ x \geq 0 \text{ inteiro}\}$$

podemos relaxar a condição de integralidade e obter um problema de programação linear P':

$$\text{Max } cx, \quad x \in S' = \{x \,|\, Ax \geq b,\ x \geq 0\}$$

Podemos observar que $S \subset S'$.

Com isso, podemos usar métodos de programação linear para trabalhar com nosso problema.

Suponha que ao resolver P' encontremos $x^* \in S'$ inteiro. Então, $x^* \in S$ e, portanto, x^* é solução de P. Infelizmente, este caso é raro, porém, é possível construir um novo sistema de equações lineares $A'x \geq b'$ tal que se

$$T = \{x \,|\, Ax \geq b\,,\ A'x \geq b'\,,\ x \geq 0\}$$

então

1) $S \subset T \subset S'$ e

2) se x^* é solução de

$$Max\, cx, \quad x \in T$$

então x^* é solução de P.

Cada equação de $A'x = b'$ define um hiperplano que corta o politopo S'. Assim, serão chamadas de *Planos de Corte*. As inequações de $A'x \geq b'$, por sua vez, restringirão S' de forma a não excluir os pontos inteiros de S.

Encontrar esse sistema é uma tarefa de custo exponencial. Portanto, neste trabalho, apresentamos um algoritmo polinomial que gera um conjunto também polinomial de planos de corte.

Considere o programa linear P' acima e A, b os dados do politopo. Suponha A e b inteiros.

Seja $A = \begin{pmatrix} a_0 \\ \vdots \\ a_{m-1} \end{pmatrix}$ e $b = \begin{pmatrix} b_0 \\ \vdots \\ b_{m-1} \end{pmatrix}$, dado $i \in \{0, \ldots, m-1\}$, temos que $a_i x \geq b_i$

Seja x^* o resultado do seguinte programa linear

$$Max\ -a_i x, \quad x \in S'$$

temos que $-a_i x^* \geq -a_i x$ para todo $x \in S'$, logo $a_i x \geq a_i x^*$ para todo $x \in S'$. Assim,

$$a_i x \geq a_i x^* \geq b_i$$

e, portanto, como desejamos manter os pontos inteiros, se x é inteiro

$$\lfloor a_i x \rfloor \geq \lfloor a_i x^* \rfloor \geq \lfloor b_i \rfloor$$

Como a_i, x e b_i são inteiros, temos

$$a_i x \geq \lfloor a_i x^* \rfloor \geq b_i$$

Frequentemente ocorre $a_i x^* > b_i$, neste caso, podemos modificar o limite b_i para $\lfloor a_i x^* \rfloor$.

Daí vem o nome do método. Uma vez que ocorre $a_i x^* > b_i$, iremos inserir uma nova inequação no sistema: $a_i x \geq \lfloor a_i x^* \rfloor$, porém, $a_i x \geq b_i$ se torna uma inequação desnecessária e, portanto, é suficiente fazer a alteração do valor de b_i. Fizemos um corte no politopo paralelo à equação $a_i x \geq b_i$. Dizemos que este corte foi um corte efetivo.

Assim, é importante sabermos quantos cortes paralelos podem ser feitos em um politopo de fatoração.

Vimos que se x é um ponto inteiro, então as componentes de $\mathbb{A}x$ só podem assumir dois valores (0 e 1 ou 0 e 2). Portanto, se x é inteiro, as componentes de Ax referentes às somas também só poderão assumir dois valores (0 e 1 ou 0 e 2 ou 0 e -1 ou 0 e -2). Assim, uma vez que é verificado um corte efetivo em uma dessas equações, o valor de b_i deve ser modificado de forma extrema, isto é, de 0 para 2 ou de -2 para 0 no caso dos possíveis valores serem 0 e 2 em $\mathbb{A}$.

No politopo de fatoração de N temos que todas as equações tem no máximo 4 variáveis binárias, assim, b_i pode assumir no máximo 4 valores diferentes. Porém, vimos pelo paralelismo das facetas que as componentes de $\mathbb{A}x$ só podem assumir dois valores (0 e 1 ou 0 e 2 ou 0 e -1 ou 0 e -2). Assim, uma vez que é verificado um corte efetivo com relação a uma equação $a_i x \geq b_i$ ela nunca mais é candidata ao corte paralelo. Ainda mais, se $a_j = -a_i$, pelo paralelismo, $a_j x \geq b_j$ também não será mais candidata ao corte paralelo.

Além disso, vimos que existe um emparelhamento das equações de $\mathbb{A}$. Se a_k está emparelhada com a_i e for verificado um corte efetivo em uma delas, digamos a_k, então, a_i deve também ser cortada.

6.4 A complexidade da redução

Vamos, primeiramente, calcular o número de variáveis envolvidas no Programa Inteiro final. Sabemos que o número de variáveis não se altera na redução *Satisfabilidade* $\longrightarrow$ *Programação Inteira*, portanto, basta contarmos o número de variáveis na expressão booleana.

Seja $N = (z_n^*, \ldots, z_0^*)$ o número que queremos fatorar com $n+1$ dígitos binários. Vamos estimar o número de dígitos binários dos seus fatores primos p e q. Observe que, se n é impar, então $p, q > 2$, o que implica que $p, q < \frac{N}{2}$ e, portanto, p e q tem, no máximo, n dígitos binários. Podemos melhorar essa estimativa fazendo uma quantidade polinomial de testes.

Então, suponha que $p = q = 2^n - 1 = (1, 1, \ldots, 1)$. Ao multiplicar estes números obtemos o seguinte diagrama:

$$
\begin{array}{cccccc}
 & & & 1 & \cdots & 1 \quad 1 \\
\times & & & 1 & \cdots & 1 \quad 1 \\
\hline
 & & & 1 & \cdots & 1 \quad 1 \\
 & & 1 & \cdots & 1 & 1 \\
 & \cdots & \cdots & \cdots & \cdots & \\
1 & \cdots & 1 & 1 & &
\end{array}
\Bigg\} \; n \text{ linhas}
$$

Onde temos $n^2 + 2n$ entradas diferentes de 0. Esse é um caso extremo. Os espaços em branco no diagrama representam zeros e não precisam ser contados como variáveis.

Agora faremos $n-1$ somas. Por simplicidade, vamos colocar o mesmo número de variáveis para cada soma. Então precisamos de um limite superior. Temos que, se $p = q = 2^n - 1$, então $p \cdot q = 2^{2n} - 2^{n+1} + 1 < 2^{2n}$ e, portanto, $p \cdot q$ tem no máximo $2n$ dígitos binários. Vamos supor então, que em cada soma $S(Z, A, B)$, A, B, Z têm $2n$ dígitos binários. Como A e B são números com variáveis já apresentadas, para cada soma $S(Z, A, B)$ devemos introduzir as variáveis de Z e as variáveis auxiliáres

de Carry, que vêm em mesma quantidade. Logo, para cada soma $S(Z, A, B)$ adicionamos $2 \cdot 2n$ variáveis. Portanto, para o processo de somas iremos adicionar $(n-1) \cdot 4n = 4n^2 - 4n$ variáveis.

Assim, a expressão booleana final envolve $n^2+2n+4n^2-4n = 5n^2-2n$ variáveis. No caso em que forçamos a unicidade do ponto inteiro, devemos introduzir mais uma ou duas somas e, portanto, $4n$ ou $8n$ variáveis.

Devemos então, calcular o número de cláusulas presentes na expressão final. Para cada multiplicação desejada são incluídas 3 cláusulas na expressão. Para cada soma de 3 bits são incluídas 16 cláusulas. Cada soma de números de $2n$ dígitos envolve $2n$ somas de 3 bits e, portanto, para estas, incluímos $2n \cdot 16 = 32n$ cláusulas. Iremos fazer também algumas igualdades do tipo $z = 0$ ou $z = 1$ que podem ser representadas por cláusulas do tipo $\overline{z}$ ou z, assim, somente adicionamos uma cláusula para cada uma dessas igualdades.

É estimado que cada fator tem n dígitos binários e, portanto, são feitas n^2 multiplicações. Então, fazemos $(n-1)$ somas de números de $2n$ dígitos e, para garantir que a última soma resulte em N, fazemos $2n$ igualdades. Assim, a expressão final contém $n^2 \cdot 3 + (n-1) \cdot 32n + 2n = 35n^2 + 30n$ cláusulas. No caso onde forçamos a unicidade do ponto inteiro, iremos adicionar $32n$ ou $64n$ cláusulas extras.

Como cada cláusula gera uma única equação no sistema final, o número de equações é igual ao número de cláusulas. Além disso, cada cláusula envolve no máximo 4 variáveis e, assim, cada linha da matriz terá no máximo 4 entradas não nulas. Com isso, a matriz final terá no máximo $4(35n^2 + 30n)$ entradas não nulas.

Portanto, concluímos que a redução proposta é de ordem $O(n^2)$.

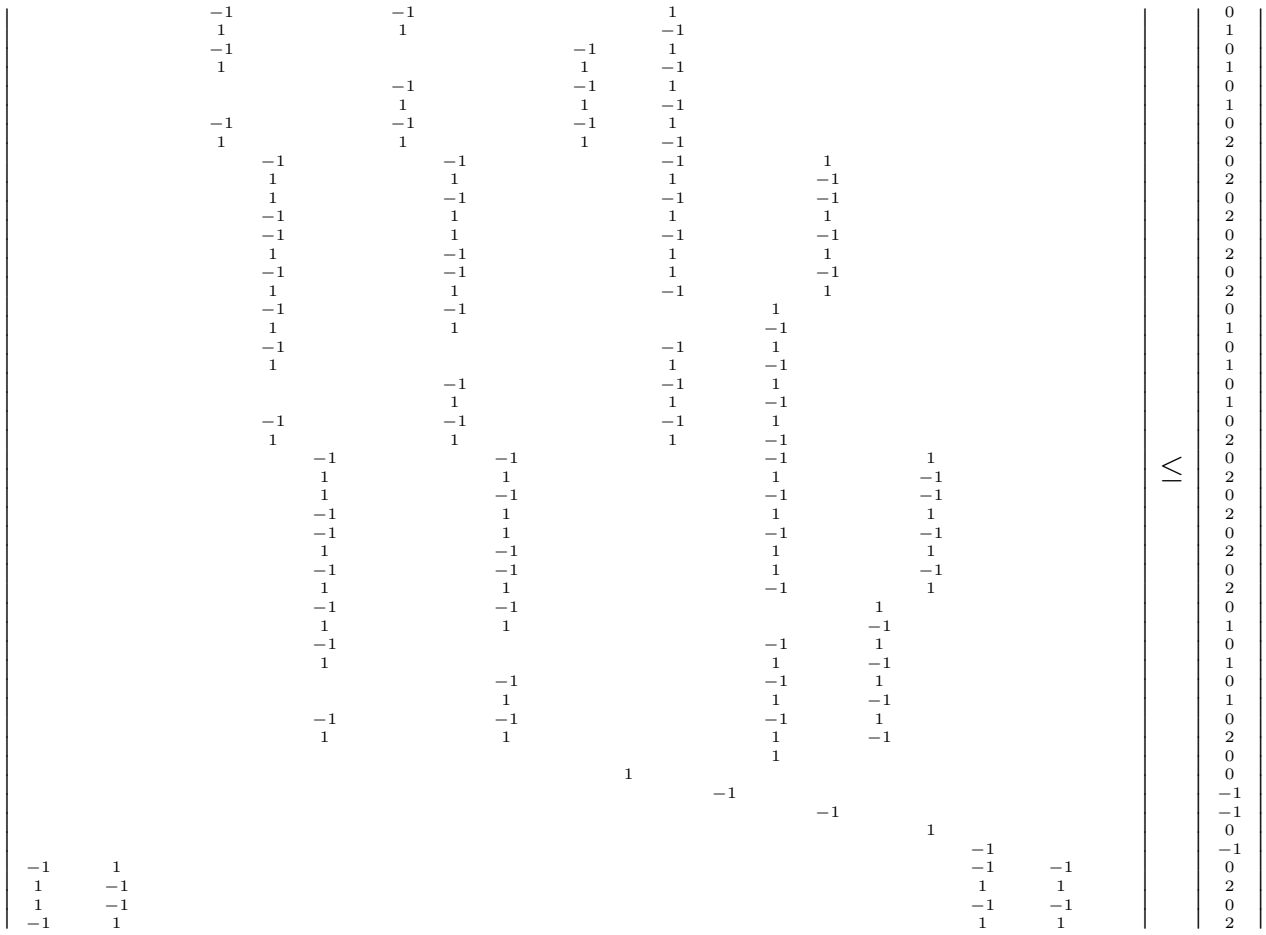
Observamos também que o algoritmo introduz uma submatriz para cada novo par de variáveis introduzido. Logo, o número de passos dado é da mesma ordem do número de variáveis: $O(n^2)$.

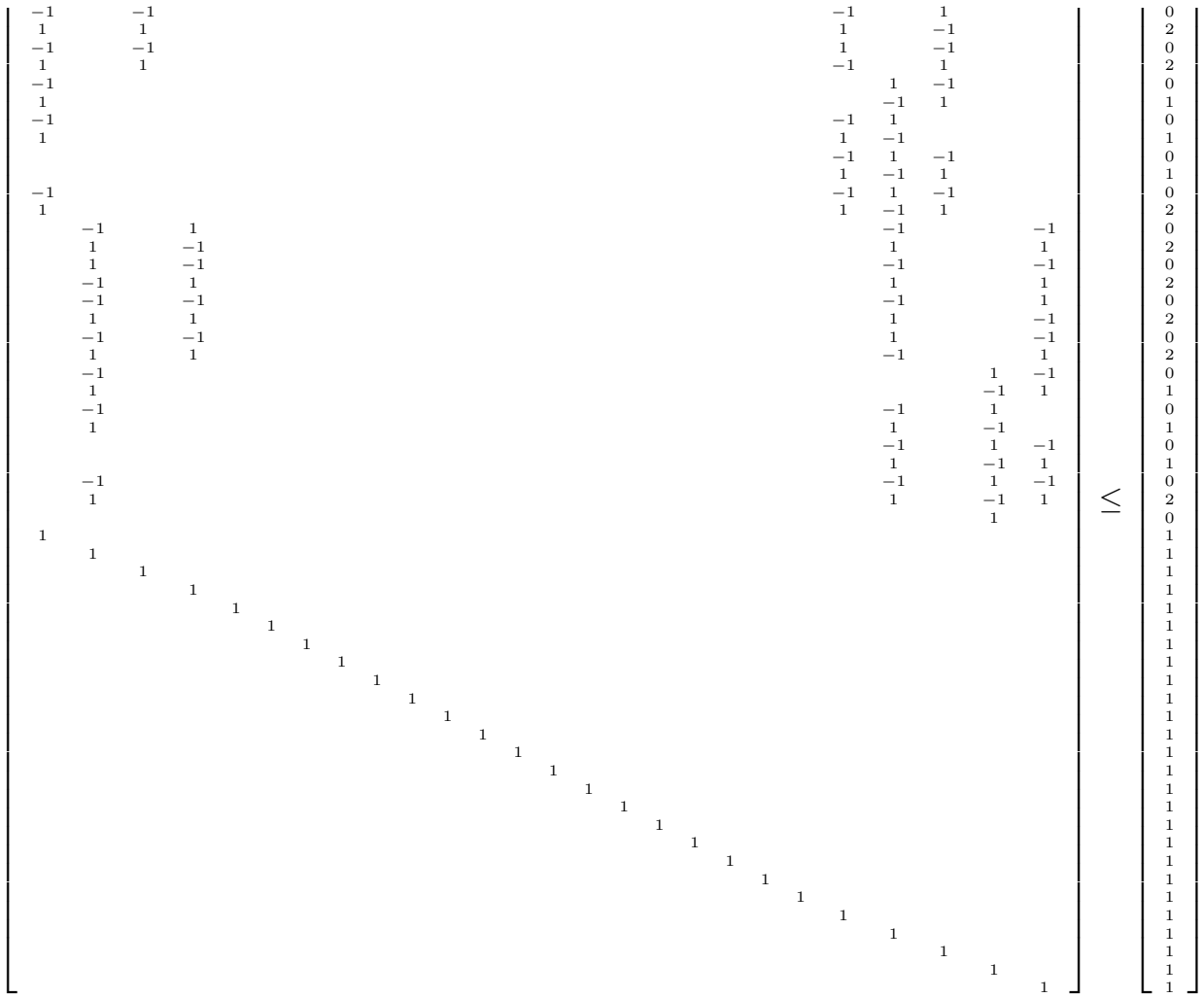
Apêndice

O Politopo de Fatoração de 6

Devido à extensão da matriz, apresentamos apenas o exemplo do politopo de fatoração P_6.Abaixo está a matriz A_6 do politopo, a desigualdade e o vetor b_6,que são suficientes para representar o politopo.Já observamos que cada uma das equações do sistema linear envolve no máximo 4 variáveis e,portanto, omitimos os numerosos zeros presentes na matriz.Omitimos também o vetor de variáveis para uma melhor exposição da matriz.

$$
\begin{bmatrix}
-1 & & & & 1 & & & & & & & & & & & \\
 & & -1 & & 1 & & & & & & & & & & & \\
1 & & 1 & & -1 & & & & & & & & & & & \\
 & -1 & & & & 1 & & & & & & & & & & \\
 & & -1 & & & 1 & & & & & & & & & & \\
 & 1 & 1 & & & -1 & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & \\
 & & & & & & & & 1 & & & & & & & \\
-1 & & & & & & & & & 1 & & & & & & \\
 & & & -1 & & & & & & 1 & & & & & & \\
1 & & & 1 & & & & & & -1 & & & & & & \\
 & -1 & & & & & & & & & 1 & & & & & \\
 & & & -1 & & & & & & & 1 & & & & & \\
 & 1 & & 1 & & & & & & & -1 & & & & & \\
 & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & 1 & & & \\
 & & & & -1 & & & & -1 & & & & -1 & & 1 & \\
 & & & & 1 & & & & 1 & & & & 1 & & -1 & \\
 & & & & 1 & & & & -1 & & & & -1 & & -1 & \\
 & & & & -1 & & & & 1 & & & & 1 & & 1 & \\
 & & & & -1 & & & & 1 & & & & -1 & & -1 & \\
 & & & & 1 & & & & -1 & & & & 1 & & 1 & \\
 & & & & -1 & & & & -1 & & & & 1 & & -1 & \\
 & & & & 1 & & & & 1 & & & & -1 & & 1 & \\
 & & & & -1 & & & & -1 & & & & & 1 & & \\
 & & & & 1 & & & & 1 & & & & & -1 & & \\
 & & & & -1 & & & & & & & & -1 & 1 & & \\
 & & & & 1 & & & & & & & & 1 & -1 & & \\
 & & & & & & & & -1 & & & & -1 & 1 & & \\
 & & & & & & & & 1 & & & & 1 & -1 & & \\
 & & & & -1 & & & & -1 & & & & -1 & 1 & & \\
 & & & & 1 & & & & 1 & & & & 1 & -1 & & \\
 & & & & & -1 & & & & -1 & & & & -1 & & 1 \\
 & & & & & 1 & & & & 1 & & & & 1 & & -1 \\
 & & & & & 1 & & & & -1 & & & & -1 & & -1 \\
 & & & & & -1 & & & & 1 & & & & 1 & & 1 \\
 & & & & & -1 & & & & 1 & & & & -1 & & -1 \\
 & & & & & 1 & & & & -1 & & & & 1 & & 1 \\
 & & & & & -1 & & & & -1 & & & & 1 & & -1 \\
 & & & & & 1 & & & & 1 & & & & -1 & & 1
\end{bmatrix}
\leq
\begin{bmatrix}
0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2
\end{bmatrix}
$$





Referências Bibliográficas

[1] T. Achterber, *Scip - a framework to integrate constraint and mixed integer programming*, Mathematical Programming Computation **1** (2005), no. 1, 1–41.

[2] M Agrawal, N Kayal, and N Saxena, *Primes is in p*, The Annals of Mathematics Second Series **160** (2004), no. 2, 781–793.

[3] R. P. Brent, *Some integer factorization algorithms using elliptic curves*, Australian National University (1985).

[4] S. Cook, *The complexity of theorem-proving procedures*, Proceedings of the 3rd Symposium on the Theory of Computing (1971), 151–158.

[5] R. S. Garfinkel and G. L Nemhauser, *Integer programming*, Wiley-Interscience, New York; London; Sydney; Toronto, 1972.

[6] Jr. H. W. Lenstra, *Elliptic curve factorization*, personal communication via Samuel Wagstaff Jr. (1985).

[7] Juris Hartmanis and Richard E. Stearns, *On the computational complexity of algorithms*, Trans. American Mathematical Society (1965), no. 117, 285–306.

[8] Donald E. Knuth, *The art of computer programming*, 2 ed., vol. 2, Addison Wesley, 1998.

[9] L. Levin, *Universal search problems*, Problemy Peredachi Informatsii **9** (1973), no. 3.

[10] A. Makhorin, *"glpk – gnu linear programming toolkit"*, 2001.

[11] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **1** (1978), no. 2.

[12] A. Schrijver, W.J. Cook, and W.H. Cunningham, *Combinatorial optimization*, John Wiley Professional, 1997.

[13] S. Singh, *The code book*, Anchor Books, 2000.

[14] I. Wegener, *The complexity of boolean functions*, Wiley, Teubner; Chichester; New York; Brisbane; Toronto; Singapore, 1987.

[15] J. Eldon Whitessit, *Boolean algebra and its applications*, Addison-Wesley, Massachusetts, 1961.

Este volume foi tipografado em LaTeXna classe UFPEThesis (www.cin.ufpe.br/ paguso/ufpethesis).