Provadores de teoremas baseados em contagem

Eduardo Menezes de Morais
lenin@linux.ime.usp.br
Orientador: Marcelo Finger

2 de fevereiro de 2009

Sumário

A Exceção e a Regra
“Estranhem o que não for estranho.
Tomem por inexplicável o habitual.
Sintam-se perplexos ante o cotidiano.
Tratem de achar um remédio para o abuso
Mas não se esqueçam de que o abuso é sempre a regra.”
Bertold Brecht

Sobre A Violência
“A corrente impetuosa é chamada de violenta
Mas o leito do rio que a contem
Ninguém chama de violento.

A tempestade que faz dobrar as bétulas
é tida como violenta
E a tempestade que faz dobrar
Os dorsos dos operários na rua?”
Bertold Brecht

1 Introdução

A idéia desse trabalho surgiu tentando imaginar maneiras alternativas de se provar teoremas.

Nesse trabalho foram estudados e produzidos provadores de teoremas na lógica proposicional baseados em métodos de contagem.

Provar teoremas eficientemente na lógica proposicional é um problema muito útil e ao mesmo tempo muito difícil.

Muito útil pois a lógica proposicional é uma lógica consistente e completa, conforme demonstrado por Bertrand Russel, e além de ser usado diretamente em diversas áreas da computação, o Teorema de Cook-Levin mostra como uma execução de uma máquina de Turing não-determinística pode ser reduzida à uma fórmula na lógica proposicional.

Por esse último motivo também que é difícil: todos os problemas resolvidos com máquinas de Turing não-determinísticas podem ser reduzidos a esse problema. Em outras palavras, ele é um problema (Co-)NP-Completo.

Por isso é importante se estudar diversos métodos e idéias diferentes para provar teoremas. Todos os algoritmos apresentados foram implementados e seu código vai em anexo.

2 Conceitos e tecnologias estudadas

Começaremos revendo alguns conceitos e propriedades conhecidos e utilizados no trabalho. Para aqueles já familiarizados com a área, não existe problema em pular diretamente para a seção 3.

2.1 A lógica proposicional

A lógica proposicional é um sistema formal onde fórmulas são formadas combinando *variáveis* e *conectivos lógicos*. As variáveis podem assumir dois valores, *verdadeiro*(1) ou *falso*(0), e podem ser combinadas utilizando os conectivos binários de *conjunção* ("E", símbolo $\wedge$), *disjunção* ("OU", símbolo $\vee$), *implicação* (símbolo, $\rightarrow$) e o conectivo unário *negação* ("NÃO", símbolo $\neg$).

Exemplo 1.

$$(A \vee B) \wedge \neg C$$

é um exemplo de fórmula na lógica proposicional.

$$(Chove \wedge (Chove \rightarrow GramaMolhada)) \rightarrow GramaMolhada$$

é outro exemplo de fórmula. Essa última fórmula pode ser vista como a afirmação "Choveu e quando chove molha a grama. Portanto a grama está molhada".

Uma atribuição de valores *verdadeiro* ou *falso* às variáveis é chamada uma **interpretação** da fórmula. Como cada variável pode receber um valor verdadeiro ou falso fica claro que se temos n variáveis, temos 2^n possíveis interpretações.

Após atribuir valores às variáveis, podemos calcular se para aquela interpretação a fórmula é verdadeira segundo essas propriedades dos conectivos lógicos expostas nas tabela 1.

Interpretações que tornam a fórmula verdadeira, dizemos que *satisfazem*

	$\wedge$
$0 \wedge 0$	0
$0 \wedge 1$	0
$1 \wedge 0$	0
$1 \wedge 1$	1

	$\vee$
$0 \vee 0$	0
$0 \vee 1$	1
$1 \vee 0$	1
$1 \vee 1$	1

	$\rightarrow$
$0 \rightarrow 0$	1
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

	$\neg$
0	1
1	0

Tabela 1: Resultados dos conectivos lógicos

a fórmula. Dependendo de quantas interpretações satisfazem uma fórmula, classificamos ela pelo seguinte parâmetro:

- Se **todas** as interpretações satisfazem uma fórmula, ela é uma tautologia (ou teorema)
- Se **algumas** interpretações satisfazem uma fórmula, ela é satisfatível
- Se **nenhuma** interpretação satisfaz uma fórmula, ela é insatisfatível

Definição 2. *Chamamos de **literal** uma variável (A) ou sua negação ($\neg A$).*

Uma última propriedade interessante de se falar sobre a lógica proposicional é que existe uma transformação que torna todas as fórmulas em disjunções de conjunções, com três literais de por conjunção, conservando a propriedade de tautologia.

Explicando melhor: para todas as possíveis fórmulas na fórmula proposicional existem fórmulas equivalentes que são disjunções de conjunções com três literais. Essa equivalência é no que diz respeito à tautologia, isto é, se a fórmula original é uma tautologia, a transformação também é uma tautologia. Essa nova forma é chamada 3DNF (*Disjuntive Normal Form*)[5].

2.2 O problema

Neste trabalho foram estudados *provadores de teoremas*, isto é, algoritmos que decidem se uma fórmula na lógica proposicional é ou não um teorema (tautologia).

O problema de decidir se uma fórmula 3DNF é um teorema é o problema complementar do famoso problema *3SAT* (problema da satisfatibilidade booleana), o primeiro problema provado ser NP-Completo [1, 7]. Isso faz com que esse problema seja Co-NP-Completo (i.e., dado um contra-exemplo, é rápido verificar se de fato ele mostra que não é um teorema).

Por ser complementar do problema *3SAT*, podemos resolver esse problema simplesmente negando a fórmula e resolvendo o problema da Satisfatibilidade. Se a negação da fórmula for insatisfatível, não existe contra-exemplo para a fórmula original e ela é uma tautologia.

As abordagens mais tradicionais para a resolução desse tipo de problema são variantes do algoritmo de Davis-Putnam-Logemann-Loveland, baseado em backtrack. Também existem outras abordagens populares que não resolvem sempre o problema, mas o resolvem com alta probabilidade, como algoritmos genéticos.

2.3 A complexidade de contar

L.G. Valiant em [8] define a classe de complexidade $\#P$ como a classe de complexidade dos problemas que envolvem contar o número de caminhos que levam um máquina NP a aceitar uma entrada, ou seja, os problemas de contagem associados ao problemas de decisão em NP. Além disso também se chamou o problema de dizer quantas interpretações satisfazem uma fórmula proposicional de $\#SAT$.

Faz pouco sentido dizer que $NP \subset \#P$, pois eles contém problemas diferentes. Mas temos que se $\#P$ puder ser resolvido eficientemente (em tempo polinomial), NP também pode. Porém mesmo que $P = NP$, pode ser que $\#P$ não possa ser resolvido eficientemente. (É como se pudéssemos dizer que "$NP \subset \#P$"). Além disso, temos que alguns problemas fáceis em P correspondem a problemas difíceis em $\#P$, como por exemplo o $2SAT$. Isso levou a certos autores ([2]) a afirmar que problemas $\#P\text{-}Completos$ são ainda mais difíceis que problemas $NP\text{-}Completos$.

2.4 Transição de fase

Diversos artigos [4] apontam a existência de uma "transição de fase" em relação à proporção Conjunções/variáveis.

Segundo [4] existe grande variação no tempo necessário para provar a propriedade de satisfatibilidade numa fórmula dependendo da relação disjunções/variáveis. Testes mostram que para o $3SAT$ formulas com uma relação disjunções/variáveis numa certa faixa são os mais difíceis de se resolver (ver figura 1).

Em nenhum dos métodos implementados se observa esse comportamento.

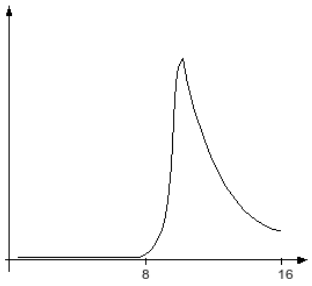


Figura 1: Baseado nos dados do artigo de Ian P. Gent e Toby Walsh

Mais detalhes podem ser vistos na seção 5.2

3 Atividades realizadas

Foram pesquisados e implementados algoritmos de prova de Teorema baseados na contagem de interpretações que satisfazem uma fórmula (ou seja, algoritmos que resolvem o problema $\#SAT$). Dos diversos algoritmos encontrados foram escolhidos três que serviram como base para outros ou que tem certas características que valiam a pena serem destacadas.

Além disso desenvolveu-se um novo algoritmo baseado em contagem. Esse algoritmo foi desenvolvido independentemente baseado nas próprias idéias do autor. Porém a leitura do artigo de O. Dubois [3] mostrou que grande parte das idéias já haviam sido desenvolvidas por Dubois, apesar da abordagem ser bastante diferente. Então foram adicionados ao novo algoritmo alguns conceitos apresentados por Dubois, mas ele foi mantido como um algoritmo a parte devido a sua abordagem diferenciada.

Todos os algoritmos aqui apresentados foram implementados em java. Criou-se um arcabouço de funções e estruturas de dados comuns de maneira que os diferentes métodos agissem como *plugins*, implementando uma mesma interface e usando as mesmas estruturas de dados.

Após a implementação, os algoritmos foram rodados diversas vezes e dados foram coletados. Chegou-se assim a diversas conclusões, principalmente

sobre a chamada *transição de fase* da satisfatibilidade [4].

Algo que deve ser salientado aqui é que a implementação do novo método ainda tem algumas limitações que (esperamos) serão superadas logo.

4 Funcionamento dos algoritmos

Nessa seção serão apresentados os algoritmos estudados com uma pequena explicação de cada um. Para maiores detalhes consulte artigos específicos de cada método.

4.1 Tabela-Verdade

O primeiro método que podemos imaginar para contar o número de interpretações é enumerando-as. Verificar todas as interpretações e ver quais satisfazem a fórmula. É o famoso método da tabela-verdade. Ver tabela 2 para um exemplo.

a	b	c	$(a \wedge \neg b) \vee \neg c$
F	F	F	V
F	F	V	F
F	V	F	V
F	V	V	F
V	F	F	V
V	F	V	V
V	V	F	V
V	V	V	F

Tabela 2: Exemplo de tabela-verdade

4.2 Algoritmo de Iwama

Esse algoritmo foi pensado por Kazuo Iwama em 1987 e descrito em [6]. Outros algoritmos interessantes se originaram dele ([2] por exemplo).

Iwama apontava em seu artigo que esse algoritmo é o "dual" dos algoritmos de backtrack do ponto de vista da classe de fórmulas eficientemente resolvidas por ele.

A idéia principal do algoritmo é descrita a seguir:

Idéia. *Se temos n variáveis e apenas i literais em uma conjunção, essa conjunção valida 2^{n-i} interpretações.*

Ex.: $a \wedge b$ é valido para as interpretações $a = V, b = V, c = V$ e $a = V, b = V, c = F$

Isso ocorre porque para satisfazer uma conjunção que contém os literais $x_1, x_2, \ldots, x_i$ precisamos apenas escolher uma valoração *verdadeiro* para os literais não-negados e *falso* para os literais negados. As $n - i$ variáveis $x_{i+1}, \ldots, x_n$ podem ter qualquer valoração. Portanto para satisfazer essa dada conjunção temos 2^{n-i} interpretações.

Poderíamos pensar em calcular o número de interpretações que satisfaz uma fórmula simplesmente somando quantas interpretações satisfazem cada conjunção (lembrando que a fórmula está na DNF). Porém há um problema:

Problema. *Algumas conjunções validam a mesma interpretação.*
Ex.: Na fórmula $(a \wedge b) \vee (a \wedge c)$, ambas as conjunções validam a interpretação $a = T, b = T, c = T$

Para esse problema, Iwama encontrou a seguinte solução:

Solução. *Subtrair as interpretações que aparecem duas vezes, somar as que aparecem três vezes, ..., como na teoria dos conjuntos*

$$S = \sum_{\Omega \in C} (-1)^{\|\Omega\|} 2^{(n-\phi(\Omega))}$$

Onde S é total de interpretações, C é o conjunto dos conjuntos de conjunções não-independentes (que podem ser satisfeitas ao mesmo tempo) e $\phi(x)$ é o número de variáveis que aparece em todas as conjunções do conjunto x.

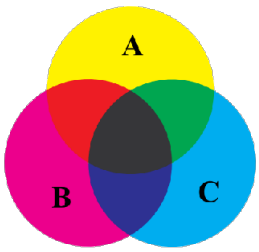


Figura 2: Podemos ver as conjunções como conjuntos que abarcam as interpretações que as satisfazem. Para contar o número total de interpretações que satisfazem a fórmula, precisamos considerar as intersecções

Exemplo 3. *Vamos calcular quantas interpretações satisfazem*

$$(a \wedge b \wedge c) \vee (d \wedge b \wedge \neg e) \vee (\neg b \wedge \neg e) \vee (b \wedge c \wedge d)$$

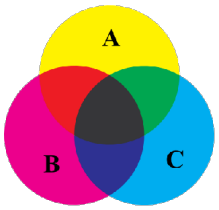
Começamos somando o número de interpretações de cada conjunção:
$(a \wedge b \wedge c)$ *é satisfeita por* $2^{5-3} = 4$ *interpretações.* $S = 4$
$(d \wedge b \wedge \neg e)$ *é satisfeita por* $2^{5-3} = 4$ *interpretações.* $S = 8$
$(\neg b \wedge \neg e)$ *é satisfeita por* $2^{5-2} = 8$ *interpretações.* $S = 16$
$(b \wedge c \wedge d)$ *é satisfeita por* $2^{5-3} = 4$ *interpretações.* $S = 20$
Agora subtraímos as intersecções dois-a-dois:
$(a \wedge b \wedge c)$ *e* $(d \wedge b \wedge \neg e)$ *são satisfeitos por* $2^{5-5} = 1$ *interpretações.* $S = 19$
$(a \wedge b \wedge c)$ *e* $(b \wedge c \wedge d)$ *são satisfeitos por* $2^{5-4} = 2$ *interpretações.* $S = 17$
$(d \wedge b \wedge \neg e)$ *e* $(b \wedge c \wedge d)$ *são satisfeitos por* $2^{5-4} = 2$ *interpretações.* $S = 15$
As demais combinações de conjunções são contraditórias e não podem aparecer juntas.
Finalmente somamos as intersecções três-a-três:
$(a \wedge b \wedge c)$ *e* $(d \wedge b \wedge \neg e)$ $e(b \wedge c \wedge d)$ *são satisfeitos por* $2^{5-5} = 1$ *interpretações.* $S = 16$
Portanto essa fórmula é satisfeita por 16 interpretações.

4.3 Algoritmo de Dubois

Olivier Dubois[3] procurou uma abordagem diferente: ao invés de ficar trabalhando para não contar duas vezes a mesma interpretação, fazer uma transformação na fórmula para cada conjunção ser independente.

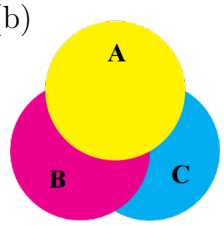
Solução. *A solução de Dubois envolve transformar algo do tipo* $A \vee B$ *em* $A \vee (B \wedge \neg A)$*, transformando tudo de volta à DNF quando necessário.*

(a)



(Interpretações originais)

(b)



(Após modificações)

Figura 3: Representação da transformação feita por Dubois

Exemplo 4. $(a \wedge b \wedge c) \vee (d \wedge e)$ *se torna* $(a \wedge b \wedge c) \vee (d \wedge e \wedge \neg a) \vee (d \wedge e \wedge a \wedge \neg b) \vee (d \wedge e \wedge a \wedge b \wedge \neg c)$
Agora basta somar o número de interpretações de cada conjunção, pois todas as conjunções são independentes. Neste caso: $S = 2^{5-3} + 2^{5-3} + 2^{5-4} + 2^{5-5} = 11$

Para tornar independente de mais de uma conjunção, para cada conjunção criada se aplica o algoritmo de novo.

Isso pode levar a um crescimento exponencial da fórmula algumas vezes. Mas muitas vezes, como os literais se repetem, algumas conjunções somem.

Além disso, Dubois teve uma outra idéia muito interessante descrita a seguir:

Idéia. *Agora temos um algoritmo para tornar conjunções independentes entre si. Dubois propõe um outro método de fazer tudo: tornar a fórmula inteira independente de uma conjunção vazia!*
Sabemos que $A \vee \square \equiv \square$ *onde* $\square$ *é uma conjunção vazia (Verdadeiro)*[1].
Ou seja, tornando $\square$ *independente do resto da fórmula, temos "quantas interpretações que* $\square$ *contribui para deixar tudo verdadeiro que ainda não foram contribuídos pela fórmula". Ou seja, quantas interpretações* ***não*** *satisfazem o resto da fórmula.*
Então fazendo $2^n - ResultadoQueVoceEncontrou$ *temos a resposta que queremos.*

Exemplo 5. *Vamos calcular o número de interpretações que satisfazem a fórmula*

$$(a \wedge d) \vee (a \wedge b \wedge c) \vee (\neg d \wedge b) \vee (\neg a \wedge c \wedge \neg d)$$

Tornando $\square$ *independente de cada conjunção (chamaremos as conjunções de* C_1*,* C_2*,* C_3 *e* C_4*). Temos então que o número de interpretações que*

de C_1	de C_2	de C_3	de C_4	número de soluções
$\square \wedge \neg a$	$\neg a$	$\neg a \wedge d$	$\neg a \wedge d \wedge \neg c$	$2^{4-3} = 2$
$\square \wedge a \wedge \neg d$	$a \wedge \neg d \wedge \neg b$	$\neg a \wedge \neg d \wedge \neg b$	$\neg a \wedge d \wedge c$	$2^{4-3} = 2$
	$a \wedge \neg d \wedge b \wedge \neg c$	$a \wedge \neg d \wedge \neg b$	$\neg a \wedge \neg d \wedge \neg b \wedge \neg c$	$2^{4-4} = 1$
			$a \wedge \neg d \wedge \neg b$	$2^{4-3} = 2$
				Total: 7

satisfazem essa fórmula é $2^4 - 7 = 9$

Essa pequena variação do método original do algoritmo de Dubois é bastante interessante. Um importante uso para ela que Dubois apontou é o fato de que ela é monotônica decrescente em cada iteração enquanto o método de Dubois sem essa variação é monotônico crescente. Isso pode ser interessante quando se está interessado em conseguir um intervalo ou uma aproximação do número de interpretações[2].

Essa variação também foi implementada (sob o nome "DuboisTrue").

[1] No artigo original, Dubois usou *Falso* numa CNF

[2] Uma pequena observação deve ser feita sobre a propriedade decrescente dessa variação.

4.4 Um novo algoritmo

Esse é algoritmo inventado pelo autor, baseado (de início sem saber) no algoritmo de Dubois, parte do seguinte questionamento:

Problema. *A transformação de Dubois para tornar uma fórmula independente pode criar um número exponencial de novas conjunções. Mas no final das contas só precisamos saber o tamanho e quantidade de novas conjunções, não elas explicitamente. É possível calcular isso?*

Solução. *Eis a solução encontrada:*

- *Começamos "fingindo" que todos os literais são diferentes* [3]*;*
- *Usando análise combinatória, sabemos exatamente quantas interpretações todas aquelas conjunções validam sem ter que calcular cada uma!*
- *Então para cada literal repetido tiramos ou adicionamos interpretações;*
- *Finalmente tiramos as variáveis extras adicionadas para "fingir" que todos os literais são diferentes.*

Vamos explicar com maiores detalhes:

Seja A o conjunto de conjunções originais e $\|A\|$ o número de interpretações satisfeitas por elas. Vamos considerar que todas as conjunções possuem três literais para efeitos de explicação, apesar de que tudo poderia ser adaptado para outro número de literais com algumas poucas mudanças.

Ao tornar as conjunções de A independentes de uma nova conjunção $x_1 \wedge x_2 \wedge x_3$, criamos novas conjunções $(\alpha \wedge \neg x_1)$, $(\alpha \wedge x_1 \wedge \neg x_2)$ e $(\alpha \wedge x_1 \wedge x_2 \wedge \neg x_3)$ para todo $\alpha \in A$.

O número de interpretações então se torna $\frac{\|A\|}{2} + \frac{\|A\|}{4} + \frac{\|A\|}{8}$, ou seja $\frac{7}{8}\|A\|$, pois para cada conjunção que existia antes, três novas conjunções são formadas, com um, dois e três literais a mais respectivamente [4].

Portanto para calcular o número de interpretações que uma dada conjunção $y_1 \wedge \cdots \wedge y_m$ contribui, sem considerar as interpretações contribuídas pelas i conjunções anteriores, basta calcular $2^{(n-m)}(\frac{7}{8})^i$.

Poderia se pensar que, como é decrescente e se quer saber se é tautologia, basta rodar uma iteração e ver se ela diminui o número de interpretações para se ter uma resposta. Isso não é verdade. Devido a algumas suposições feitas por Dubois, um resultado 0 que correspondem a 2^n na verdade.

[3] Esse é o pior caso do algoritmo de Dubois

[4] $\frac{7}{8}$ é o "número mágico" quando se tem exatamente três literais. No caso geral se multiplica por $\frac{\sum_{i=0}^{n-1} 2^i}{2^n}$

$2^{(n-m)}$ é o número de conjunções originais contribuídas (o $\|A\|$ acima). Multiplica-se por $\frac{7}{8}$ para cada conjunção que se pretende tornar independente.

Não se esqueça que o número de variáveis aqui dito é um número "falso", considerando-se que nenhum literal se repete!

Exemplo 6. *Vamos calcular o número de interpretações que cada conjunção contribui na fórmula*

$$(a \wedge b \wedge c) \vee (d \wedge e \wedge f) \vee (g \wedge h \wedge i)$$

$(a \wedge b \wedge c)$ *contribui* $2^{9-3} = 64$
$(d \wedge e \wedge f)$ *contribui* $2^{9-3}(\frac{7}{8}) = 56$ *interpretações além das interpretações contribuídas anteriormente.*
$(g \wedge h \wedge i)$ *contribui* $2^{9-3}(\frac{7}{8})^2 = 49$ *interpretações além das interpretações contribuídas anteriormente.*
$Total = 169$
Nesse caso o resultado está correto pois os literais já são todos diferentes.

4.4.1 Conflitos

Agora temos que ver a questão dos literais se repetindo. O problema é que quando os literais não são todos diferentes, algumas das conjunções criadas para garantir independência podem ser contraditórias ou ter menos literais do que o normal. Quando isso acontece chamamos **conflito**.

Se dois literais contraditórios aparecem juntos, precisamos eliminar as conjunções onde eles aparecem; se dois literais iguais aparecem juntos, precisamos duplicar o número de interpretações contribuídas por suas conjunções. Em ambos os casos precisamos pegar um subconjunto das conjunções e somar ou subtrair ao total.

Para pegar esse subconjunto utilizamos a mesma idéia utilizada anteriormente, mas ao invés de considerar as três possibilidades ($(\alpha \wedge \neg x_1)$, $(\alpha \wedge x_1 \wedge \neg x_2)$ e $(\alpha \wedge x_1 \wedge x_2 \wedge \neg x_3)$), consideramos apenas a que nos interessa.

Exemplo 7. *Na i-ésima conjunção, temos que podem aparecer juntos numa conjunção* $a \wedge b$ *e* $\neg a \wedge c \wedge d$. *Então calculamos os casos em que eles aparecem usando:*

$$2^{n-m} \left(\frac{7}{8}\right)^{i-2} \left(\frac{1}{4}\right) \left(\frac{1}{8}\right)$$

Ao invés de reconstruir tudo só para mudar um detalhe, podemos "voltar" um passo e multiplicar pelo que queremos agora com a seguinte operação:

$X\left(\frac{8}{7}\right)\left(\frac{1}{2^n}\right)$. Multiplicar por $\frac{8}{7}$ cancela a multiplicação por $\frac{7}{8}$.

Isso tudo também funciona se estivermos tornando a fórmula independente de $\square$.

4.4.2 Conflitos conflitantes

O maior problema é quando uma mesma conjunção se envolve em mais de um conflito. Nesse caso não se pode simplesmente "voltar a trás e pegar o caso que queremos" como feito anteriormente. Tem que se considerar o que já foi eliminado/somado anteriormente. Caso contrario pode-se acabar somando/subtraindo duas vezes a mesma interpretação.

Normalmente isso envolve fazer uma "bifurcação", considerando o outro conflito. Ou seja, vamos considerar separadamente a situação em que aparece o outro conflito e que não aparece.

Essa bifurcação pode ser calculada de maneira simples, multiplicando por algo do tipo $(1 \pm \left(\frac{8}{7}\right)\left(\frac{1}{2}\right))$. Ou seja, usando a construção $(1 \pm x)$ podemos considerar os dos casos.

É importante perceber que esse x pode conter ele mesmo bifurcações no caso do conflito considerado conflitar com um terceiro.

Um exemplo completo desse novo método pode esclarecer muitas coisas.

Exemplo 8. *Vamos tornar* $(a \wedge \neg b \wedge c) \vee (\neg b \wedge d \wedge e) \vee (\neg d \wedge f)$ *independentes de* $\square$ *e calcular o número de interpretações que satisfazem essa fórmula*
Número "falso" de variáveis : 8
$2^8\left(\frac{7}{8}\right)^2\left(\frac{3}{4}\right) = 7^2 \cdot 3 = 147$
Conflito: (d) *c/* $(\neg b \wedge \neg d)$*:* $-7^2 \cdot 3 \cdot \frac{4}{3} \cdot \frac{1}{2} \cdot \frac{8}{7} \cdot \frac{1}{4} = -28$
Conflito: (d) *c/* $(\neg b \wedge d \wedge \neg e)$*:* $+7^2 \cdot 3 \cdot \frac{4}{3} \cdot \frac{1}{2} \cdot \frac{8}{7} \cdot \frac{1}{8} = +14$
Conflito: $(\neg d \wedge \neg f)$ *c/* $(\neg b \wedge \neg d)$*:* $+7^2 \cdot 3 \cdot \frac{4}{3} \cdot \frac{1}{4} \cdot \frac{8}{7} \cdot \frac{1}{4} = +14$
Conflito: $(\neg d \wedge \neg f)$ *c/* $(\neg b \wedge d \wedge \neg e)$*:* $-7^2 \cdot 3 \cdot \frac{4}{3} \cdot \frac{1}{4} \cdot \frac{8}{7} \cdot \frac{1}{8} = -7$
Conflito: $(a \wedge b)$ *c/* (b)*:* $+7^2 \cdot 3 \cdot \frac{8}{7} \cdot \frac{1}{4} \cdot \frac{8}{7} \cdot \frac{1}{2} = +24$
Conflito: $(a \wedge b)$ *c/* $(\neg b \wedge \neg d)$*:* $-7^2 \cdot 3 \cdot \frac{8}{7} \cdot \frac{1}{4} \cdot \frac{8}{7} \cdot \frac{1}{4} \cdot (1 - \frac{4}{3} \cdot \frac{1}{2} + \frac{4}{3} \cdot \frac{1}{4}) = -8$
Conflito: $(a \wedge b)$ *c/* $(\neg b \wedge d \wedge \neg e)$*:* $-7^2 \cdot 3 \cdot \frac{8}{7} \cdot \frac{1}{4} \cdot \frac{8}{7} \cdot \frac{1}{8} \cdot (1 - \frac{4}{3} \cdot \frac{1}{2} + \frac{4}{3} \cdot \frac{1}{4}) = -8$
Conflito: $(a \wedge \neg b \wedge \neg c)$ *c/* (b)*:* $-7^2 \cdot 3 \cdot \frac{8}{7} \cdot \frac{1}{8} \cdot \frac{8}{7} \cdot \frac{1}{2} = -12$
Conflito: $(a \wedge \neg b \wedge \neg c)$ *c/* $(\neg b \wedge \neg d)$*:* $+7^2 \cdot 3 \cdot \frac{8}{7} \cdot \frac{1}{8} \cdot \frac{8}{7} \cdot \frac{1}{4} \cdot (1 - \frac{4}{3} \cdot \frac{1}{2} + \frac{4}{3} \cdot \frac{1}{4}) = +4$
Conflito: $(a \wedge \neg b \wedge \neg c)$ *c/* $(\neg b \wedge d \wedge \neg e)$*:* $+7^2 \cdot 3 \cdot \frac{8}{7} \cdot \frac{1}{8} \cdot \frac{8}{7} \cdot \frac{1}{8} \cdot (1 - \frac{4}{3} \cdot \frac{1}{2} + \frac{4}{3} \cdot \frac{1}{4}) = +4$
Total=144
Tínhamos fingido que o número de variáveis era 8, quando na verdade é 6.
$Total = 144 \div 4 = 36$
Como fizemos o algoritmo da independência com $\square$*, temos que a resposta é*

$$2^6 - 36 = 28$$

que de fato é a resposta correta!

5 Resultados

5.1 Eficiência dos algoritmos

Para testar a eficiência dos algoritmos, todos foram expostos a uma bateria de testes que envolvia executar o algoritmo para fórmulas 3-DNF geradas aleatoriamente com um número de variáveis entre 3 e 30 (usando incrementos de 3) e com um número aleatório de conjunções, podendo ser até 16 vezes o número de variáveis. Todos foram repetidos duzentas vezes para cada algoritmo.

A figura 4 mostra os resultados em termos de tempo de execução.

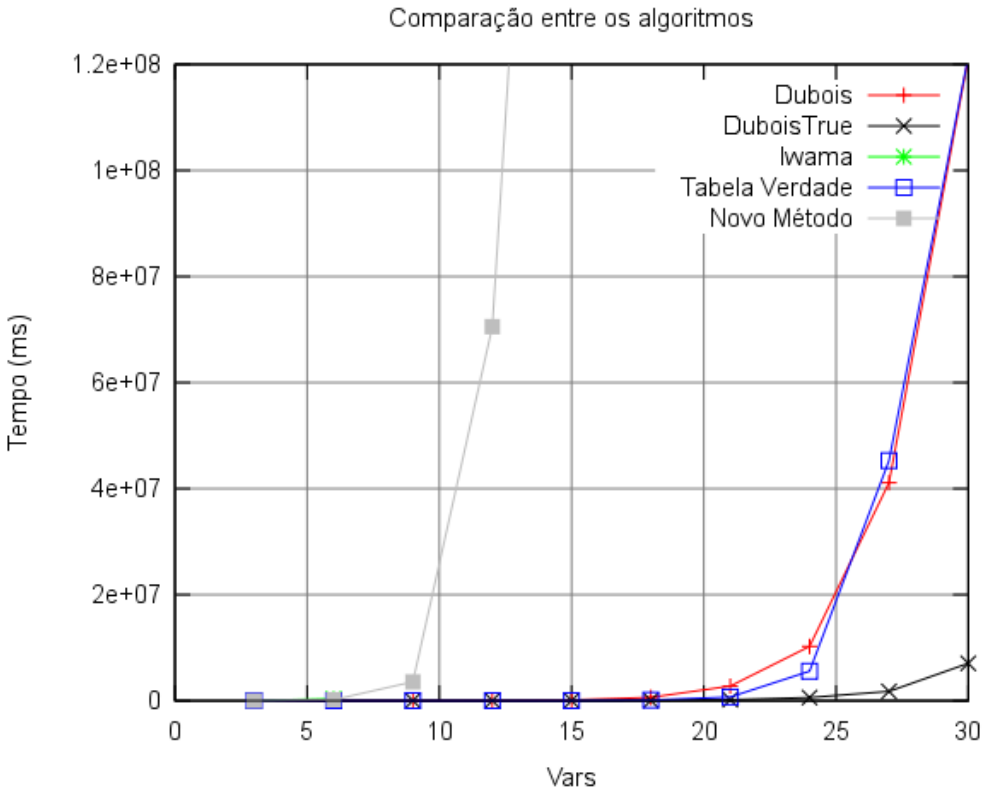


Figura 4: A eficiência dos algoritmos considerados

Alguns comentários sobre a eficiência de cada algoritmo:

5.1.1 A Tabela-verdade

A Tabela-verdade se destaca por seu baixo uso de memória (constante para fórmulas com o mesmo número de variáveis) e pelo tempo necessário crescer rapidamente. Porém, comparado com alguns métodos, percebe-se que a Tabela-verdade não é tão ruim como se poderia imaginar, superando o algoritmo de Dubois para n pequeno.
Sua "melhor" característica parece ser o fato de seu tempo de execução se manter quase constante quando aumenta o número de conjunções e se mantem o mesmo número de variáveis (ver seção 5.2), tornando a Tabela-verdade recomendável para um número muito grande de conjunções para poucas variáveis.

5.1.2 O método de Iwama

O método de Iwama se mostrou bastante decepcionante pois não chegou nem mesmo a completar todos os testes devido a um consumo excessivo de memória. Em alguns minutos o algoritmo consumiu todos os 1Gb de heap da JVM e saiu. O que causou esse excessivo consumo de memória foi a geração do conjunto de conjunções independentes, mesmo eliminando os conjuntos anteriores. Com um número muito grande de conjunções, o consume de memória é demasiado grande.
Não foi possível testar muito bem o tempo do algoritmo devido a este problema. Por isso os resultados dos gráficos referentes a esse método parecem "incompletos".

5.1.3 O método de Dubois

O método de Dubois foi o melhor dos algoritmos implementados, sendo que a versão "DuboisTrue" (usando a técnica de tornar as conjunções independentes da fórmula vazia) foi ainda melhor.
Aparentemente o método "DuboisTrue" não tem perda significativa de performance quando se aumenta a razão conjunções/variáveis (ver figura 5d). Isso se explica pelo seguinte fato: [4] explica como quando a razão conjunções/variáveis é muito grande, a fórmula tende a ser uma tautologia, pois existem muitas interpretações que cada conjunção "contribui". Após processar o número de conjunções suficiente para se provar uma tautologia, as demais são rápidas, pois calcular a independência se torna trivial.

5.1.4 O novo método

O novo método apresentado aqui teve um desempenho decepcionante. Uma análise mais detalhada mostra que foi devido ao *overhead* de procurar e arrumar "conflitos conflitantes" que causou essa ineficiência.
Porém, deve-se perceber que a implementação feita do novo método é bastante ingênua, e não se fez nenhuma otimização que poderia levar a um melhor resultado.

5.2 Transição de fase

Todos os testes mostram que a transição de fase indicada na subseção 2.4 parece não ocorrer com algoritmos baseados em contagem. Esse é um fato surpreendente e não é muito bem documentado na literatura atual!

Observando os gráficos gerados pelos métodos implementados não observa-se esse tipo de comportamento, mostrando que esse métodos podem ser alternativas viáveis para essa classe de problemas (ver figuras 5a, 5b, 5c, 5d e 5e).

Todos os gráficos, exceto o da Tabela Verdade, estão acompanhados de uma aproximação aos dados experimentais. Foram testados diversas funções até encontrar a que melhor se aproxima O novo método e o algoritmo de Iwama são aproximados por uma função exponencial, enquanto os algoritmos de Dubois são aproximados por uma função quadrática.

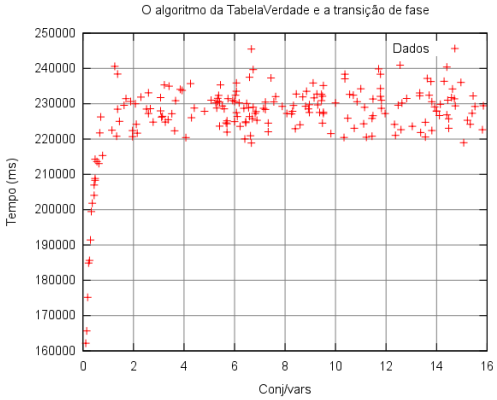
Iwama[5]	2^x
Dubois	$3106,54 \cdot x^2 - 8657,55 \cdot x + 10498.4$
DuboisTrue	$-11.5635 \cdot x^2 + 1370.89 \cdot x - 1262.2$
Novo Método	$10 \cdot 2^{1,1 \cdot x}$

Tabela 3: Aproximações aos dados experimentais

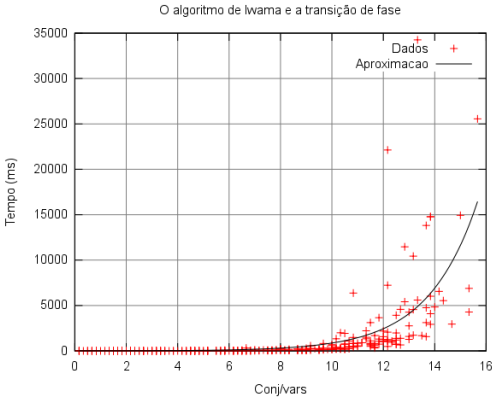
6 Conclusão

Algoritmos de prova de teorema baseados em contagem são de fato uma alternativa viável a ser seguida pois eles parecem não ser afetados pela transição de fase além de outras propriedades. Saber o número de interpretações que satisfazem uma fórmula pode ser útil como heurística em certas áreas como

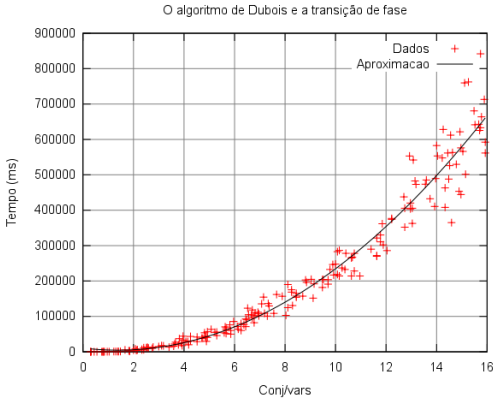
[5]Os dados do algoritmo de Iwama foram calculados com uma quantidade pequena de variáveis, de devido a impossibilidade de usar mais variáveis



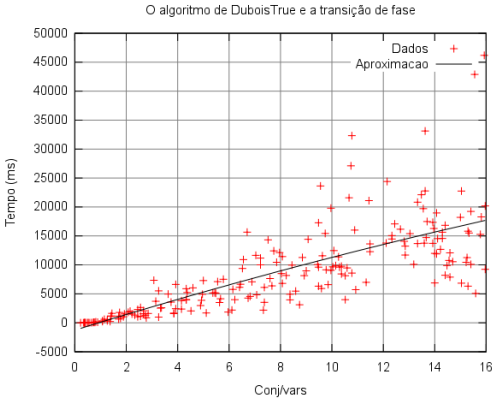
(a) Tabela-verdade



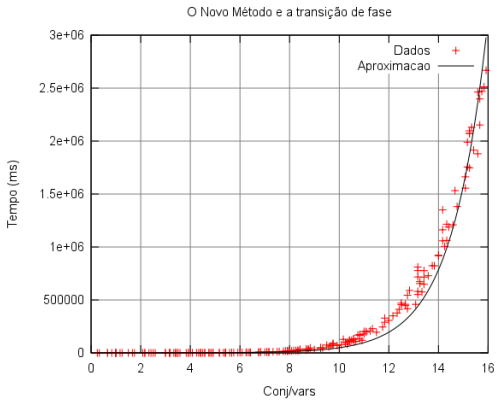
(b) Algoritmo de Iwama



(c) Algoritmo de Dubois



(d) DuboisTrue



(e) Novo Método

Figura 5: A transição de fase e os algoritmos de contagem

planejamento.

Ainda existe muito trabalho a ser feito nessa área. As implementações são bastante cruas, mas como foi dito na Introdução, elas não são o foco desse trabalho.

Em particular o novo método apresentado aqui tem muito o que ser otimizado. As contas feitas em cada iteração tendem a se repetir bastante, o que apontaria para uma possibilidade de otimização.

O fato é que o problema SAT deve ser encarado de maneiras diferentes e criativas para que se possa avançar e sempre que possível, deve-se questionar as abordagens mais usadas.

Referências

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[2] Ştefan Andrei. Counting for satisfiability by inverting resolution. *Artificial Intelligence Review*, 22(4):339–366, 2004.

[3] Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theor. Comput. Sci.*, 81(1):49–64, 1991.

[4] Ian P. Gent and Toby Walsh. The sat phase transition. In *Proceedings of 11th ECAI*, pages 105–109, 1994.

[5] John Harrison. Formal verification methods 1: Propositional logic. Slides de aulas, Julho 2003.

[6] Kazuo Iwama. Cnf satisfiability test by counting and polynomial average time. *SIAM J. Comput.*, 18(2):385–391, 1989.

[7] B. A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *IEEE Ann. Hist. Comput.*, 6(4):384–400, 1984.

[8] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[9] Wikipedia. Boolean satisfiability problem — Wikipedia, the free encyclopedia, 2008. [Online; accessado 16 de junho de 2008].

[10] Wikipedia. Sharp-p — Wikipedia, the free encyclopedia, 2008. [Online; accessado 29 de novembro de 2008].

7 Parte subjetiva

7.1 Sobre a elaboração desse trabalho e uma balanço

Dois momentos distintos podem ser destacadas sobre a experiência de elaboração desse trabalho:
Num primeiro momento, fiquei muito empolgado com o trabalho. Gostei das coisas que pesquisava, em grande parte devido ao algoritmo original que estava criando, mas também por gostar bastante de lógica. Li muito sobre

lógica, inclusive muitas coisas que não tem nada a ver diretamente com o trabalho. Esperava um trabalho muito bom.
Num segundo momento me senti decepcionado. Primeiramente ao concluir que meu algoritmo não era original e que Dubois havia pensando naquilo antes (apesar de ter superado isso ao perceber as diferenças de nossas abordagens). Depois por perceber que muitas coisas que gostaria de fazer e pesquisar (alguns outros algoritmos, consertar os erros na implementação do novo algoritmo, ...) não seriam possíveis devido a falta de tempo.
Acredito realmente que esse é um assunto fascinante mas creio que talvez essa monografia esteja aquém do possível. Isso se deu um pouco por ter deixado as coisas para a última hora, um pouco por não ter sido realista nas espectativas.

7.2 Sobre o as disciplinas e o curso

Ao contrário de que algumas pessoas, sempre fui a favor de um curso bastante teórico. É importante que se tenha um conhecimento profundo na área e não se perca estudando tecnologias e linguagens que serão obsoletas em alguns anos. A matemática é essencial no curso (apesar de eu não ser muito bom nela...) e é bom que o curso do IME foque bastante nessas áreas.
As disciplinas que mais contribuíram para esse trabalho foram:

- MAC 211 - Laboratório de Programação
  Por ensinar ferramentas essenciais para um dia-a-dia de programação
- MAC 239 - Métodos Formais de Programação
  Ensinou o básico de lógica formal
- MAC 329 - Álgebra Booleana e Aplicações
  Aprofundou o conhecimento de lógica proposicional
- MAE 312 - Introdução aos Processos Estocásticos
  Esclareceu algumas questões quanto à analise combinatória e teoria dos conjuntos que foram utilizados nos algoritmos
- MAC 414 - Linguagens Formais e Autômatos
  Esclareceu questões referentes a computação de respostas de problemas e seus limites
- MAC 425 - Introdução à Inteligência Artificial
  Mais lógica

Essas disciplinas apontadas não correspondem necessariamente as minhas favoritas, mas sim às mais relevantes ao trabalho.
Uma última observação quanto ao curso: concordo com as posições do professor Carlinhos sobre trabalho durante o curso. De fato atrapalha muito e pude sentir isso na pele. Seria muito melhor se todos os estudantes pudessem se dedicar unicamente à faculdade, porém isso não é a realidade. Infelizmente em muitos casos essa posição atrapalha imensamente um estudante que tem dificuldades de se sustentar sem trabalhar. Acredito que seja necessário um curso noturno de Bacharelado em Ciências da Computação, talvez com duração de 5 anos para permitir aos estudantes que precisem trabalhar possam cumprir com qualidade o curso. Isso deve ser pensado seriamente.

7.3 Sobre o futuro do projeto

Pretendo continuar estudando a área e melhorando meu algoritmo. Continuo estudando lógica (atualmente estou lendo um livro do profº Newton da Costa chamado "Ensaios sobre os fundamentos da lógica". Excelente), e a dois dias antes da entrega desse trabalho tive uma idéia de modificações que poderiam ser feitas no novo algoritmo. Portanto espero continuar esses estudos, mesmo que seja por *hobbie*.