PROBLEMA DA SATISFABILIDADE BOOLEANA

Diego S. Costa, Talles B. de Assunção

Universidade Federal de Roraima (UFRR)
Boa Vista – RR – Brasil

Resumo. *Este artigo descreve o problema de Satisfabilidade Booleana (SAT – Boolean Satisfiability problem), o primeiro algoritmo a ser classificado como NP-completo, que verifica se existe alguma solução para uma expressão booleana ser verdadeira. São apresentados algoritmos para sua resolução, complexidade e análise dos tempos de execução.*

1. O problema SAT

O problema SAT (Boolean Satisfiability problem) foi o primeiro problema a ser identificado como NP-completo. O problema consiste em, dada uma expressão booleana formada por conectivos AND (∧), OR (∨) e NOT (¬), com n variáveis, verificar se existe alguma solução que torne essa expressão verdadeira. Caso exista, essa expressão e dita como satisfazível.

Exemplos:

Dada a expressão booleana com variáveis x1, x2, x3 e x4, existem valores para x1, x2, x3 e x4 que torne essa expressão verdadeira?

$(x1 \vee x2) \wedge (\neg x1 \wedge x3) \wedge (\neg x1 \wedge x3 \wedge \neg x4) \wedge (x2 \vee x4)$

Se existir alguma atribuição para x1, x2, x3 e x4 que torne essa expressão verdadeira ela é considerada satisfazível. No caso, é possível notar que a expressão tem solução com x1 = 0, x2 = 1, x3 = 1 e x4 = 0.

Dada a expressão booleana com variáveis x1, x2, x3 e x4, existem valores para x1, x2, x3 e x4 que torne essa expressão verdadeira?

$(x1 \wedge \neg x1) \vee (x2 \wedge \neg x2) \vee (x3 \wedge \neg x3) \vee (x4 \wedge \neg x4)$

Neste caso, é possível notar que não é possível expressão ser verdadeira com nenhum valor para x1, x2, x3 e x4, então é dito que a expressão é insatisfazível.

2. Versão exata do SAT

O algoritmo da versão exata do SAT, disponível no arquivo sat-exata.c, busca uma solução por meio da força bruta, gerando todas as possíveis combinações de valores para as variáveis da expressão booleana de maneira recursiva, caso encontre uma, encerra a busca e exibe os valores das variáveis que torna a expressão verdadeira. Dada uma expressão booleana para a verificação, é criado um vetor com n posições, onde cada posição é uma variável da expressão, o algoritmo percorre o vetor de trás para frente atribuindo 0 em cada posição. Chegando na primeira posição, nela é atribuído o valor 0 e verifica se a expressão é verdadeira, caso não seja, o valor atribuído agora é 1 e verifica novamente, se também não for, volta uma posição e atribui 1 nela e percorre o vetor novamente.

Podemos identificar que o pior caso desse algoritmo será se a única solução para a expressão ser verdadeira se todas as variáveis possuírem o valor, pois será a última combinação gerada pelo algoritmo. Outro pior caso também é se não existir solução, pois todas as combinações terão que ser verificadas.

```
int forca_bruta(int vet[], int pos_atual){
    if(pos_atual==0){
        vet[0] = 0;
        if(expressao(vet)==1){
            return 1;
        }

        vet[0] = 1;
        if(expressao(vet)==1){
            return 1;
        }
        return 0;
    }
    else{
        vet[pos_atual] = 0;
        if(forca_bruta(vet,pos_atual-1)==1){
            return 1;
        }
        vet[pos_atual] = 1;
        if(forca_bruta(vet,pos_atual-1)==1){
            return 1;
        }
        return 0;
    }
}
```

Figura 1. Função forca_bruta()

Esse algoritmo possui complexidade $O(2^n)$, cada atribuição de valor e if() da função possui custo 1, assim é possível notar que com uma variável no pior caso, o custo seria O(4), e criar o sistema para calcular a complexidade:

$$\begin{cases} 0 & \text{, para } n = 0 \\ 2T(n-1) + 4 & \text{, para } n > 0 \end{cases}$$

$2T(n-1) + 4$, para $n=1$

$2(2T(n-2) + 4) + 4 = 4T(n-2) + 12$, para $n=2$

$4(2T(n-3) + 4) + 12 = 8T(n-3) + 28$, para $n=3$

$8(2T(n-4) + 4) + 28 = 16T(n-4) + 60$, para $n=4$

$2^kT(n-k) + 2^{k+2} – 4$

Com $k = n$,

$2^nT(n-n) + 2^{n+2} – 4 = 2^nT(0) + 2^{n+2} – 4 = 2^{n+2} – 4$

Complexidade $O(2^n)$.

3. Algoritmo Zchaff

O algoritmo Zchaff foi desenvolvido por Zhaohui Fu, Yogesh Marhajan e Sharad Malik da Universidade de Princeton, Estados Unidos. Essa solução implementa o algoritmo de Chaff, utilizando a estratégia de decisão VSIDS (Variable State Independent Decaying Sum), que torna possível manter o registro de cada literal de cada variável. Podemos citar também outras estratégias implementas por esse algoritmo, como: propagação de restrição booleana com dois literais vigiados, aprendizado de clausula por conflito e backtrack não cronológico.

Esse algoritmo tem como heurística resolver o conflito de clausulas mais recente e com o aprendizado de clausulas pequenas é possível reduzir periodicamente o número de clausulas excluindo-as, sua inutilidade é decidida estatisticamente pela inferência da aprendizagem de clausulas pequenas e pelo seu tamanho, dependendo desses dois fatores a clausula é excluída, pois ela apenas faria perder mais tempo de computação. O algoritmo também conta com reinícios frequentes para melhorar a busca local.

3.1. Ambiente de testes

Os testes foram realizados em ambiente Linux. Computador com Ubuntu 16.04LTS 64 bits, processador i7 de 4ª geração, 8gb de memória RAM. Durante os testes o computador encontrava-se off-line e executava apenas um dos algoritmos por vez e os tempos de execução foram medidos utilizando a função *time* do Linux.

As entradas para os algoritmos foram fórmulas booleanas com diferentes números de clausulas e com 2, 3, 30, 35 e 36 variáveis, cada fórmula foi posta em um arquivo que servia de entrada para o algoritmo zchaff, o algoritmo de versão exata implementado por nós precisava que as fórmulas fossem colocadas diretamente no trecho de código. Como o tempo de execução da versão exata do algoritmo do problema SAT começou a cresce exponencialmente, paramos os experimentos com 36 variáveis que já estavam levando 14 minutos para dar uma resposta, porém ainda fizemos mais um experimento de 1006 variáveis e 33592 clausulas com o algoritmo Zchaff que se mostrou muito eficiente encontrando uma resposta se a fórmula era sat ou unsat em apenas 23 segundos.

4. Resultados e estatísticas

O tempo de execução a cada entrada foi expressado em um gráfico para melhor exibição dos resultados obtidos com os experimentos.

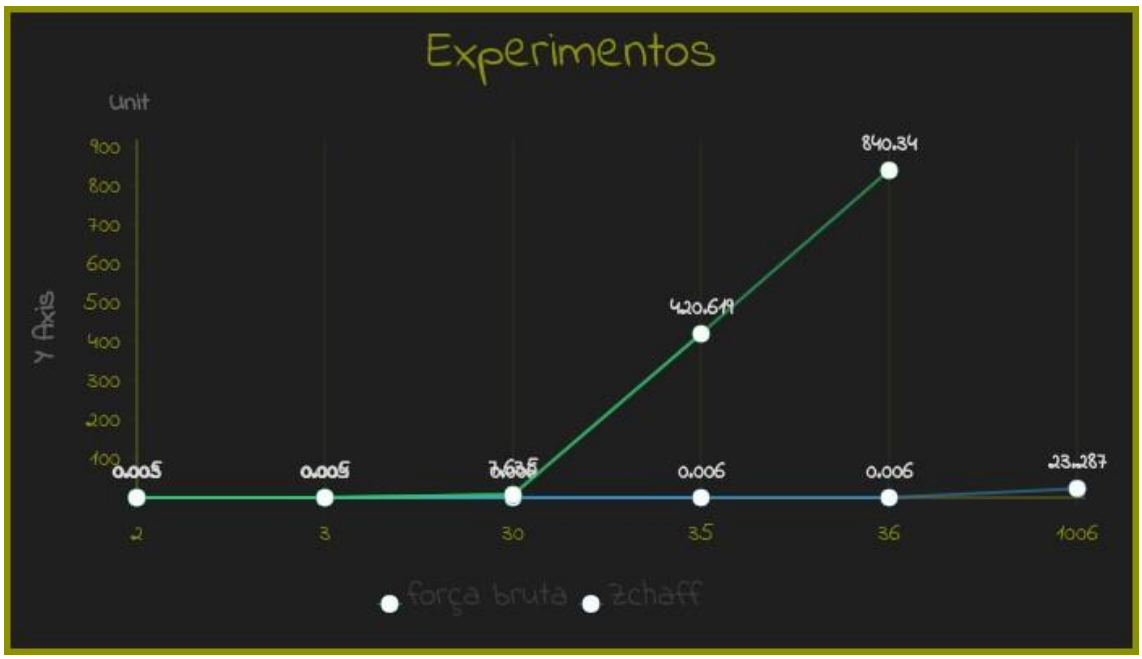

Figura 2. Resultados dos experimentos

Como é possível observar, a partir de 30 variáveis o algoritmo da versão exata começa a ter um aumento exponencial, tornando-se inviável de ser utilizado, enquanto o algoritmo Zchaff decide uma resposta na casa de segundos com mais de 1000 variáveis.

A tabela abaixo contém os dados dos experimentos. As colunas contêm o tempo em segundos para a resposta dos algoritmos de cada linha, a primeira linha representa o número de variáveis da fórmula:

	2	3	30	35	36	1006
Zchaff	0.005	0.005	0.006	0.006	0.006	23.287
Algoritmo exato	0.003	0.004	7.635	420.619	840.34	Desconhecido

Tabela 1. Resultados dos experimentos

Referências

Fux, Jacques. Análise de Algoritmos SAT para Resolução de Problemas Multivalorados. Disponível em: https://www.dcc.ufmg.br/pos/cursos/defesas/23M.PDF, acesso em 04 de julho de 2018.

Análise de Algoritmos. Disponível em: https://www.ime.usp.br/~cris/mac5711/slides/aula22.pdf, acesso em 04 de julho de 2018.

Zhaohui Fu, Yogesh Marhajan e Sharad Malik. zChaff SAT Solver. Disponível em: http://fmv.jku.at/sat-race-2006/descriptions/6-zChaff.pdf, acesso em 01 de julho de 2018.