Prova da NP-completude do problema 3SAT

Gabriel Manzoni Moreira, gmmoreira@inf.ufrgs.br
Rafael Thomazi Gonzalez, rtgonzalez@inf.ufrgs.br

Sumário

1. Introdução

Na teoria da complexidade, os problemas de decisão podem ser divididos em classes conforme suas características. A classe P contem os problemas que podem ser resolvidos em uma máquina de Turing determinística em tempo polinomial. A classe NP (Nom-deterministic Poliomial time) engloba os problemas que são resolvidos em tempo polinomial em uma máquina de Turing não-determinística. Além disso, a classe NP pode ser caracterizada como o conjunto de problemas que podem ser verificados em uma máquina de Turing determinística em tempo polinomial. Por fim, os problemas NP-Completos são um subconjunto da classe NP tendo como característica o fato de que caso uma solução em tempo polinomial seja encontrada para um deles, então todos os outros problemas NP-Completos também poderão ser resolvidos em tempo polinomial.

Este trabalho tem como objetivo descrever o problema 3SAT e apresentar a prova de sua NP-completude. Para isso, inicialmente é feita a caracterização do problema e, em seguida é feita a descrição da prova. A prova será feita reduzindo o problema SAT para o problema 3SAT e apresentando um algoritmo de verificação para o 3SAT em tempo polinomial.

2. Caracterização do problema

Uma fórmula CNF (forma normal conjuntiva) é uma conjunção de i cláusulas C, cada uma sendo uma disjunção de termos. Uma fórmula 3CNF é uma conjunção de i cláusulas C, cada uma sendo uma disjunção de exatamente três termos.

O problema 3SAT é o problema da satisfazibilidade Booleana que pode ser

representado como o conjunto de todas fórmulas 3CNF satisfazíveis. O Problema de satisfazibilidade Booleana é o problema de determinar se existe uma determinada valoração para as variáveis de uma determinada fórmula booleana tal que a valoração satisfaça esta fórmula em questão. Sendo assim o 3SAT é uma conjunção de i cláusulas C, cada uma sendo uma disjunção de exatamente três termos.

O problema 3SAT é uma derivação do problema original SAT, este por sua vez possui os mesmos princípios, porém no SAT cada cláusula pode ter um número variável de termos, indo de cláusulas com apenas um termo, até com n termos. Para ficar mais fácil de entender segue um exemplo de uma fórmula SAT: (a) ^ (a V b) ^ (a V b V c) ^ (!a V !b V d V e V !f). E um exemplo de uma fórmula 3SAT: (a V b V c) ^ (a V !b V c) ^ (a V b V !c).

3. Prova da NP-Completude

Essa prova se divide em duas partes. Na primeira, será provado que 3SAT pertence a classe dos problemas NP, ou seja existe um algoritmo de verificação em tempo polinomial. Em seguida, usaremos um problema da classe NP-Completo e reduziremos ele ao problema do 3SAT em tempo polinomial.

3.1. Primeira Parte: $3SAT \in NP$

A verificação da satisfabilidade de uma expressão booleana é feita em tempo polinomial, pois dado um conjunto de valores-verdade atribuíveis a cada variável da expressão, basta realizar uma varredura pelos literais substituindo os mesmos pelos seus valores-verdade. No fim verificamos se a expressão foi satisfeita ou não computando o valor-verdade de cada cláusula.

Assim, como a verificação da satisfabilidade da expressão booleana pode ser feita em tempo polinomial, o problema 3SAT pertence a classe NP.

3.1.1. Algoritmo de verificação

Entradas: Uma fórmula na 3CNF com *n* cláusulas e um vetor (certificado) com o valor-verdade de cada variável da fórmula.

```
1.  function verificaSatisfazibilidade( f3SAT[n], certif )
2.  {
3.  for ( i = 0; i < n; i++ ) {
4.     if( certif[f3SAT[i].v1]==false && certif[f3SAT[i].v2]==false &&
5.     certif[f3SAT[i].v3]==false )
6.  {
7.      return false;
8.  }
9.  }
10. return true;
11. }
```

3.1.2. Análise da Complexidade

Cp[verificaSatisfazibilidade] = Cp[3..10]
Cp[3..10] = Cp[3..9] + Cp[10]
Cp[10] = O(1)
Cp[3..9] = $\sum_{i=0}^{n-1} Cp[4..8]$

Cp[4..8] = O(1)

$$\sum_{i=0}^{n-1} O(1) = O(n)$$

Cp[3..9] =

Cp[3..10] = O(n) + O(1) = O(n)

Cp[verificaSatisfazibilidade] = O(n)

Como o algoritmo de verificação tem complexidade linear para o número de cláusulas da expressão provamos que o problema 3SAT NP.

3.2. Segunda Parte: $([?][?][?]L \in NP-Completo)\,(L \leq p\,3SAT)$

Para concluir a prova de que o problema do 3SAT é um problema NP-Completo, devemos provar que existe algum problema NP-Completo que pode ser reduzido ao problema do 3SAT.

Será usado o problema SAT que, pelo Teorema de Cook, é provado que pertence à classe dos problemas NP-Completo.

3.2.1. Redução de SAT para 3SAT

A redução de uma instância pertencente ao problema SAT para outra pertencente ao problema 3SAT é feita substituindo cada cláusula *Ci* da expressão *fSAT* por *Ci'* na expressão *f3SAT*, de tal forma que:

- Se *Ci* = X_1, então coloca-se a seguinte expressão em *f3SAT*:
  Ci' = $(X_1 \vee Y \vee Y_1) \wedge (X_1 \vee \neg Y \vee Y_1) \wedge (X_1 \vee Y \vee \neg Y_1) \wedge (X_1 \vee \neg Y \vee \neg Y_1)$

- Se *Ci* = $(X_1 \vee X_2)$, então coloca-se a seguinte expressão em *f3SAT*:
  Ci' = $(X_1 \vee X_2 \vee Y) \wedge (X_1 \vee X_2 \vee \neg Y)$

- Se *Ci* = $(X_1 \vee X_2 \vee X_3)$, então copia-se *Ci* para *f3SAT*;

- Se *Ci* = $(X_1 \vee X_2 \vee ... \vee X_k)$ e k > 3, então coloca-se a seguinte expressão em *f3SAT*:
  Ci' = $(X_1 \vee X_2 \vee Y) \wedge (\neg Y \vee X_3 \vee Y_1) \wedge (\neg Y_1 \vee X_4 \vee Y_2) \wedge ... \wedge (\neg Y_{(k-4)} \vee X_{(k-2)} \vee Y_{(k-3)}) \wedge (\neg Y_{(k-3)} \vee X_{(k-1)} \vee X_k)$

Resumidamente, a redução é baseada no mapeamento de *fSAT* para *f3SAT*, as cláusulas de *fSAT* com um e dois termos são transformadas em cláusulas com três termos, e cláusulas com quatro ou mais termos são quebradas em cláusulas de exatamente três termos.

3.2.2. Equivalência entre SAT e 3SAT

Deve-se mostrar que tais substituições não alteram a satisfazibilidade da expressão. Para isso será provado que f3SAT é satisfazívelfSAT é satisfazível.

Para as cláusulas com um e dois elementos a prova é trivial.

Para cláusulas com um termo temos que X_1 deve possuir o mesmo valor verdade que $(X_1 \vee Y \vee Y_1) \wedge (X_1 \vee \neg Y \vee Y_1) \wedge (X_1 \vee Y \vee \neg Y_1) \wedge (X_1 \vee \neg Y \vee \neg Y_1)$. Isto é percebido facilmente, pois se X_1 for Verdade, teremos um termo verdadeiro em todas as cláusulas, o que basta para tornara a expressão verdadeira. Agora, se X_1 for Falso, podemos notar que existem todas combinações de Y e Y_1 nas cláusulas, logo para qualquer combinação de valores verdade de Y e Y_1, sempre existirá uma cláusula que será falsa, tornando a expressão falsa.

Para cláusulas com dois elementos, podemos provar usando propriedades da lógica básica. Partiremos de $(X_1 \vee X_2)$ e chegaremos em $(X_1 \vee X_2 \vee Y) \wedge (X_1 \vee X_2 \vee \neg Y)$.

1. $(X_1 \vee X_2)$

2. (X_1 v X_2 v Y): por introdução da disjunção da linha 1

3. (X_1 v X_2 v !Y): por introdução da disjunção da linha 1

4. (X_1 v X_2 v Y) ^ (X1 v X2 v !Y): por união da linha 2 com a linha 3.

Como partimos de (X_1 v X_2) e chegaremos em (X_1 v X_2 v Y) ^ (X_1 v X_2 v !Y) é provado então que as duas expressões possuem o mesmo valor verdade.

Para cláusulas com três elementos, basta copiar a cláusula, pois está ja é um 3SAT. E para cláusulas com 4 ou mais elementos, segue a prova abaixo.

3.2.2.1. *f3SAT é satisfazível se fSAT é satisfazível*

Esta prova será feita por redução ao absurdo.

Assuma que *f3SAT* é satisfazível e que *fSAT* não seja. Então como *fSAT* = (X_1 v X_2 v .. v X_k) = F, temos X_1 = X_2 = ... = X_k = F.

Como por hipótese *f3SAT* = (X_1 v X_2 v Y) ^ (!Y v X_3 v Y_1) ^ (!Y_1 v X_4 v Y_2) ^ ... ^ (!$Y_{(k-4)}$ v $X_{(k-2)}$ v $Y_{(k-3)}$) ^ (!$Y_{(k-3)}$ v $X_{(k-1)}$ v X_k) é satisfazível, todas as suas cláusulas devem ser satisfazíveis. Assim, para que a primeira cláusula de *f3SAT* seja satisfazível, temos que ter Y = V. Logo, teremos !Y = F na segunda cláusula, obrigando que Y_1 = V para que essa cláusula seja satisfeita. Com isso !Y_1 = F, o que obriga Y_2 = V para que a terceira cláusula seja verdadeira.

Continuando este raciocínio, deve-se ter $Y_{(k-3)}$ = V para a penúltima cláusula ser verdadeira, já que !$Y_{(k-4)}$ = F. Desse modo, a última cláusula será falsa, pois ela possui !$Y_{(k-3)}$ = F e $X_{(k-1)}$ = X_k = F, tornando a espressão *f3SAT* não satisfazível. Isso é um absurdo, pois contraria a hipótese inicial.

Logo, ***f3SAT é satisfazível se fSAT é satisfazível.***

3.2.2.2. *fSAT é satisfazível se f3SAT é satisfazível*

Esta prova será feita por redução ao absurdo.

Assuma que *fSAT* é satisfazível e que *f3SAT* não seja.

Sendo *fSAT* = (X_1 v X_2 v .. v X_k), suponha que X_i, com 1 ≤ i ≤ k, seja o termo de menor índice responsável por tornar *fSAT* satisfazível, ou seja, para todos termos até o i-ésimo é atribuído o valor-verdade F e que ao i-ésimo termo é atribuído o valor-verdade V.

Logo defini-se que os literais de *fSAT3* são atribuídos da seguinte maneira:

Y_j = F, se max(1, i-1)jk-3

Y_j = V, caso contrário.

fSAT3 = (X_1 v X_2 v Y_1) ^ (X_3 v !Y_1 v Y_2) ^ (X_4 v !Y_2 v Y_3) ^ ... ^ (X_{k-2} v !Y_{k-4} v Y_{k-3}) ^ (X_{k-1} v X_k v !Y_{k-3}). A partir da regra, temos dois tipos possíveis de atribuições para os literais devido à função max(1, i-1).

Primeiro suponha que max(1, i-1) = 1, o termo de menor índice que torna a cláusula verdadeira será o primeiro. Então de acordo com a regra de valoramento, para qualquer j pertencente a [1, k-3], Y_j = F.

Isto torna *fSAT3* satisfazível, pois X_1 = V e Y_j = F para qualquer valor de j pertencente a [1, k-3]. Sendo assim se Y_j é F, logo !Y_j é V, e como temos !Y_j em todas cláusulas exceto a primeira, garantimos que todas cláusulas são verdadeiras e a primeira cláusula é verdadeira por X_1 = V de tal forma que torna *fSAT3* satisfazível o que é um absurdo, pois contradiz com a hipótese.

Agora suponha que max(1, i-1) = i-1. Segundo a regra temos que, Y_j = F para qualquer j pertencente a [i-1, m-3] e Y_j = V para qualquer j pertencente a [1, i-2].

Temos que *fSAT3* = (X_1 v X_2 v Y_1) ^ (!Y_1 v X_3 v Y_2) ^ … ^ (!Y_{i-3} v X_{i-1} v Y_{i-2}) ^ (!Y_{i-2} vX_i v Y_{i-1}) ^ (!Y_{i-1} v X_{i+1} v Y_i) ^ … ^ (!Y_{k-4} v X_{k-2} v Y_{k-3}) ^ (!Y_{k-3} v X_{k-1} v X_k) é satisfazível, pois entre [1, i-2] Y_j = V, X_i = V e entre [i-1, k-3] !Y_j = V. Assim *fSAT3* é satisfazível o que é um absurdo, pois

contradiz com a hipótese.

Com isso foi provado que *f3SAT* e *fSAT* são equivalentes. Assim provamos que podemos reduzir um problema NP-Completo para uma instância do problema 3SAT. Logo 3SAT é um problema NP-Completo.

3.2.3. Algoritmo de redução

Entrada: Uma fórmula na CNF com n cláusulas.
Saída: Uma fórmula na 3CNF.

```
1.  function convertSATto3SAT( fSAT[n] )
2.  {
3.  f3SAT = null;
4.  ind = 1;
5.  for (i = 0; i < n; i++)
6.  {
7.  if( fSAT[i].size() == 1 ) {
8.    v = "x" + ind++;
9.    notv = "!" + v;
10.   v1 = "x" + ind++;
11.   notv1 = "!" + v1;
12.   f3SAT.Add( new Clausula(fSAT[i][0], v, v1) );
13.   f3SAT.Add( new Clausula(fSAT[i][0], notv, v1) );
14.   f3SAT.Add( new Clausula(fSAT[i][0], v, notv1) );
15.   f3SAT.Add( new Clausula(fSAT[i][0], notv, notv1) );
16. } else if( fSAT[i].size() == 2 ) {
17.   v = "x" + ind++;
18.   notv = "!" + v;
19.   f3SAT.Add( new Clausula(fSAT[i][0], fSAT[i][1], v) );
20.   f3SAT.Add( new Clausula(fSAT[i][0], fSAT[i][1], notv) );
21. } else if( fSAT[i].size() == 3 ) {
22.   f3SAT.Add( new Clausula(fSAT[i][0], fSAT[i][1], fSAT[i][2]) );
23. } else {
24.   v = "x" + ind++;
25.   f3SAT.Add( new Clausula(fSAT[i][0], fSAT[i][1], v) );
26.   v1 = "x" + ind++;
27.   for( j = 2; j < fSAT[i].size()-2; j++) {
28.       notv = "!" + v;
29.       f3SAT.Add( new Clausula(notv, fSAT[i][j], v1) );
30.       v = v1;
31.       v1 = "x" + ind++;
32.   }
33.   notv = "!" + v;
34.   f3SAT.Add(new Clausula(notv, fSAT[i][fSAT[i].size()-2], fSAT[i]
    [fSAT[i].size()-1]) );
35.   ind--;
36. }
37. }
38. return f3SAT;
39. }
```

3.2.4. Análise da complexidade

Cp[convertSATto3SAT] = Cp[3] + Cp[4] + Cp[5..37] + Cp[38]
Cp[3] = Cp[4] = Cp[38] = O(1)
Cp[5..37] = $\sum_{i=0}^{n-1} Cp[7..36]$
Cp[7..36] = Cp[7] + Cp[16] + Cp[21] + max(Cp[8..15], Cp[17..20], Cp[22], Cp[24..36])
Cp[7] = Cp[16] = Cp[21] = Cp[8..15] = Cp[17..20] = Cp[22] = O(1)
Cp[24..36] = Cp[24..26] + Cp[27..32] + Cp[33..36]
Cp[24..26] = Cp[33..36] = O(1)
Cp[27..32] = $\sum_{j=2}^{m-3} Cp[28..31]$
Cp[28..31] = O(1)
Cp[27..32] = (m-4)*O(1) = O(m-4) = O(m)
Cp[24..36] = O(1) + O(m) + O(1) = O(m)
Cp[7..36] = O(1) + O(1) + O(1) + max(O(1), O(1), O(1), O(m)) = O(m)
Cp[5..37] = n*O(m) = O(n*m)
Cp[convertSATto3SAT] = O(n*m)

1. Conclusão

A importância do problema 3SAT reside no fato de este ser uma instância mais simétrica do problema SAT, tendo em vista que suas cláusulas têm no máximo três variáveis, sendo mais fácil por isso provar propriedades sobre ele e reduzi-lo a outros problemas NP-Completos.

Podemos mostrar a importância da classe de problemas NP-Completos, que caso exista algum algoritmo que em tempo polinomial solucione um problema desta classe, então todos os outros problemas desta classe serão solucionados em tempo polinomial.

1. Bibliografia

http://www.inf.ufrgs.br/~rcpinto/facul/inf05515/npc/index.htm

http://www.dsc.ufcg.edu.br/~abrantes/CursosAnteriores/ATAL031/NPCompleto/NPCompleto.html

http://en.wikipedia.org/wiki/Boolean_satisfiability_problem

TOSCANI, L. V., VELOSO, P. A. S. - Complexidade de Algoritmos