Universidade Federal de Ouro Preto - Campus Morro do Cruzeiro
Departamento de Ciência da Computação - DECOM
BCC241 - Projeto e Análise de Algoritmo
Profº. Drº. Anderson Almeida Ferreira

Documentação do Trabalho Prático

Marcos Paulo Ferreira Rodrigues - 14.2.4341
Thiago Gonçalves Resende - 13.2.4667
Victor Lott Galvão de Oliveira - 15.1.4240

Ouro Preto, Julho de 2019

INTRODUÇÃO

O trabalho prático se dá na implementação de três programas. Sendo eles Satisfabilidade, Clique e Conjunto independente.

O Conjunto Independente será resolvido por branch-and-bound, onde o restante dos problemas serão reduzidos polinomialmente a fim de ser resolvido da mesma forma que o conjunto independente. Nos cabe definir como fazer a redução de forma correta, e ao final também devemos saber interpretar os resultados para ter a resposta certa.

Um exemplo de redução polinomial é no caso de adquirir o menor valor de um vetor. Suponhamos que não sabemos se é possível desenvolver um algoritmo que faça isso, porém sabemos que existe um algoritmo que ordene o vetor de forma ascendente, portanto se ordenarmos o vetor e extrairmos o primeiro elemento dele, estamos assim obtendo o menor valor do vetor inteiro.

Em seguida nós iremos descrever do que consiste cada um dos problemas abordados nesse trabalho prático.

Conjunto Independente

O problema do conjunto independente consiste em, dado um Grafo G, encontrar o maior número de vértices independentes. Isto significa encontrar o maior subconjunto de G tal que não exista aresta entre nenhum par dos vértices.

A entrada do problema se dá por um Grafo G. A imagem abaixo exemplifica um Grafo.

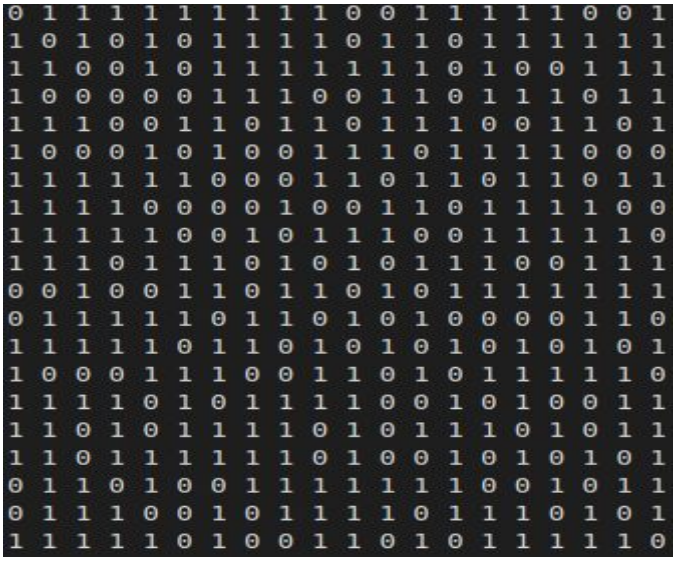

Imagem 1: grafo_20_6.txt

A imagem 1 mostra um grafo de 20 vértices apresentado como uma matriz gerado automaticamente com 60% de probabilidade de existir uma aresta entre cada vértice.

Para a resolução do problema, foi executado os seguintes passos:

1. Fazemos a leitura do grafo a partir de uma função implementada..
2. Executamos o algoritmo implementado do Branch and Bound.
3. Imprimimos a solução.

A solução do problema é uma lista contendo os índices dos vértices que fazem parte do conjunto independente.

A tabela abaixo exibe os resultados dos testes realizados na implementação da solução do conjunto independente:

INSTÂNCIAS (vértice, prob)	**TEMPO (em segundos)**	**MELHOR SOLUÇÃO**	**VÉRTICES**
grafo_10_2.txt	0.000180206	6	0 2 3 7 9 1
grafo_10_4.txt	0.000036397	4	0 3 1 9
grafo_10_6.txt	0.000037534	4	0 3 6 8
grafo_10_8.txt	0.000011418	2	0 1
grafo_20_2.txt	1.16709	9	0 3 4 5 6 7 10 15 19
grafo_20_4.txt	0.0037577	7	3 4 6 7 14 16 18
grafo_20_6.txt	0.000136365	3	1 3 5
grafo_20_8.txt	0.000089785	3	0 10 8
grafo_30_2.txt	190.296	12	1 2 4 6 11 12 13 15 19 24 26 28
grafo_30_4.txt	0.0520036	7	1 4 8 10 11 13 26
grafo_30_6.txt	0.00121473	5	1 2 3 19 20
grafo_30_8.txt	0.000136752	4	2 9 23 27
grafo_40_2.txt	0	0	0
grafo_40_4.txt	0	0	0
grafo_40_6.txt	0	0	0
grafo_40_8.txt	0	0	0

Tabela 1: Resultado da execução do algoritmo para encontrar o Conjunto Independente.

Clique

Um clique em um determinado Grafo G, é um subconjunto de vértices, tal que para cada dois vértices do grafo, existe uma aresta os conectando. Para um caso geral, se extrairmos um subconjunto C de vértices desse grafo, possuímos um clique se todos os vértices de C estão interligados por arestas. Isso se equivale a dizer que um grafo induzido de C é completo.

Devemos ressaltar que o problema do clique, seja para achar um clique máximo, ou todos os cliques de seja qual for o grafo, é NP-Completo. Ele também é intratável para parâmetros fixos, e difícil de se obter uma aproximação.

O oposto de um clique é um conjunto independente, no sentido de que cada clique corresponde a um conjunto independente no seu grafo complementar. Portanto só nos cabe realizar a redução polinomial para resolvermos o problema assim como o de conjunto independente.

Para que isso seja feito, implementamos uma função Complemento, descrita na Figura 2.

```
Grafo Complemento(Grafo& g) {
  Grafo h(g);

  for(unsigned i = 0; i < h.size() - 1; i++) {
    for(unsigned j = i + 1; j < h.size(); j++) {
      h[i][j] = h[j][i] = !h[i][j];
    }
  }

  return h;
}
```

Figura 2: Função Complemento implementada para o problema de clique.

Tal função simplesmente passa por cada elemento da matriz de adjacências do Grafo G fornecido como entrada e atribui a negação do elemento presente à um novo Grafo H, que posteriormente é retornado como resultado da função.

Após obtermos o complemento do Grafo, resolvemos o problema assim como no Conjunto Independente, pelo método do *branch-and-bound*.

A tabela abaixo mostra a comparação dos resultados de acordo com o tamanho da instância gerada pelo gerador de instâncias.

INSTÂNCIAS (vértice, prob)	**TEMPO (em segundos)**	**MELHOR SOLUÇÃO**	**VÉRTICES**
grafo_10_2.txt	0.000009712	3	2 4 5

grafo_10_4.txt	0.000018599	3	0 6 4
grafo_10_6.txt	0.000024438	4	0 4 9 1
grafo_10_8.txt	0.000254146	5	0 3 4 5 2
grafo_20_2.txt	0.000045743	3	1 6 9
grafo_20_4.txt	0.000181581	5	7 10 11 12 19
grafo_20_6.txt	0.00538006	8	0 2 4 6 9 12 19 1
grafo_20_8.txt	0.428168	9	0 3 1 5 11 14 15 17 18
grafo_30_2.txt	0.000140754	3	0 25 1
grafo_30_4.txt	0.00098644	5	1 6 12 16 17
grafo_30_6.txt	0.242446	9	0 2 8 10 12 18 25 27 6
grafo_30_8.txt	1525.45	13	1 5 6 7 8 10 11 17 20 22 25 26 29
grafo_40_2.txt	0.000360072	4	0 23 32 25
grafo_40_4.txt	0.00278079	6	0 4 22 23 27 32
grafo_40_6.txt	0.308666	8	0 7 1 12 19 27 37 38
grafo_40_8.txt	0	0	0

Tabela 2: Resultado da execução do algoritmo para encontrar o Clique.

Satisfabilidade

O problema da satisfabilidade consiste em, dado uma fórmula normal conjuntiva, encontrar uma atribuição de valores-verdade às variáveis da fórmula que a torne verdadeira, ou informe que essa atribuição não existe.

A representação da fórmula é uma matriz, onde cada linha representa uma cláusula e cada coluna uma variável daquela cláusula. O valor na cédula pode ser 0, 1 ou 2, onde 0 representa que a variável existe e é negada, 1 representa que a variável existe e não está negada e o 2 representa que a variável não está presente na cláusula.

As figuras abaixo exemplificam a transformação de uma fórmula em uma matriz.

$$(x \vee y \vee z)\ (x \vee \overline{y})\ (y \vee \overline{z})\ (z \vee \overline{x})\ (\overline{x} \vee \overline{y} \vee \overline{z})$$

Figura 2: Exemplo de fórmula normal conjuntiva com as suas variáveis.

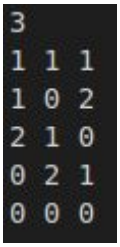

Figura 4: Fórmula da Figura 3 em uma matriz.

Para a resolução deste problema, foi executado os seguintes passos:

1. Fazemos a leitura da Fórmula, no formato de matriz.
2. Transformamos a fórmula em um Grafo G
    a. Conectamos todas as variáveis da mesma cláusula
    b. Conectamos todas as variáveis com com valores diferentes e que estejam presentes em cláusulas diferentes
3. Calculamos o Complemento do Grafo G
4. Executamos o Branch and Bound
5. Verificamos a condição de satisfabilidade
    a. O problema É SATISFAZÍVEL se o número de elementos na solução é o mesmo número de cláusulas da Fórmula inicial.

A solução deste problema é uma lista de variáveis que possui os valores 1 ou 0, sendo que 1 indica atribuição V à variável do índice e 0 indica atribuição F à variável do índice.

A tabela abaixo mostra a comparação do algoritmo de satisfabilidade de acordo com as instâncias e suas características.

INSTÂNCIAS (vértice, prob)	**TEMPO (em segundos)**	**SATISFAZ**	**VÉRTICES**
formula_2_3.txt	0.000008465	SIM	0 1
formula_2_5.txt	0.000008153	SIM	0 1
formula_2_7.txt	0.000049199	NÃO	3 4 5 6
formula_2_9.txt	0.000012806	NÃO	0 2 1
formula_4_3.txt	0.000005911	NÃO	0 1
formula_4_5.txt	0.0000015239	SIM	3 4 6 5
formula_4_7.txt	0.0000022063	NÃO	3 4 5 7 6
formula_4_9.txt	0.0000030058	NÃO	4 5 6 7 8

formula_6_3.txt	0.000014249	NÃO	0 1
formula_6_5.txt	0.000027158	NÃO	0 2 1
formula_6_7.txt	0.000061052	NÃO	3 4 5 7 6
formula_6_9.txt	0.000097938	NÃO	6 7 8 9 11 12 10
formula_8_3.txt	0.00002683	NÃO	7 8 9
formula_8_5.txt	0.000061032	NÃO	0 2 3 1
formula_8_7.txt	0.000046848	NÃO	0 2 3 1
formula_8_9.txt	0.00019888	NÃO	27 28 29 30 31 32 33

Tabela 3: Resultados da execução do algoritmo para verificar a Satisfabilidade.

Conclusão

Por fim, podemos concluir que neste trabalho prático, foi desenvolvimento um algoritmo para a extração de um Conjunto Independente de um determinado Grafo, representado computacionalmente por uma matriz de adjacência na linguagem C++. O restante dos problemas foram reduzidos polinomialmente ao problema do Conjunto Independente, sendo solucionado de tal forma.

Como todos eles estão na classe de problemas NP-Completo, conseguimos observar que o tempo de execução para determinados números de vértices e determinadas porcentagens de densidade do Grafo, os testes ultrapassaram um dia de execução, poor possuir uma complexidade exponencial.