Métodos Algébrico-Enumerativos para o Problema de Máxima Satisfatibilidade Ponderada

Anderson Delcio Parreira

Métodos Algébrico-Enumerativos para o Problema de Máxima Satisfatibilidade Ponderada

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Anderson Delcio Parreira e aprovada pela comissão julgadora.

Campinas, 10 de agosto de 1995

Prof. Dr. Marcus Vinicius S. Poggi de Aragão

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de MESTRE em Ciência da Computação.

Tese defendida e aprovada em, 10 **de** 08 **de 199** 5

Pela Banca Examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). ABILIO PEREIRA DE LUCENA FILHO

Prof(a). Dr(a). RICARDO DAHAB

Prof(a). Dr(a). MARCUS VINICIUS SOLEDADE POGGI DE ARAGÃO

Métodos Algébrico-Enumerativos para o Problema de Máxima Satisfatibilidade Ponderada[1]

Anderson Delcio Parreira[2]

Departamento de Ciência da Computação

IMECC - UNICAMP

Banca Examinadora:

- Marcus Vinicius S. Poggi de Aragão (Orientador)[3]
- Abilio Pereira de Lucena Neto[4]
- Ricardo Dahab[3]
- Claúdio Leonardo Lucchesi (suplente)[3]

---

1. Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.
2. O autor é Licenciado em Matemática pela Universidade Federal de Uberlândia.
3. Professor do Departamento de Ciência da Computação - IMECC - UNICAMP.
4. Pesquisador do LNCC-CNPq.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC

Parreira, Anderson Delcio

P248m Métodos algébrico-enumerativos para o problema de máxima satisfatibilidade ponderada / Anderson Delcio Parreira. -- Campinas, [SP: s.n.], 1995.

Orientador: Marcus Vinicius Soledade Poggi de Aragão
Dissertação (mestrado): Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Ciência da Computação.

1. Otimização Combinatória. 2. Programação não-linear. I. Poggi de Aragão, Marcus Vinicius Soledade. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Ciência da Computação. III. Título.

Agradecimentos

À Deus, pela minha existência e pelas alegrias concedidas nestes últimos dois anos.

Aos meus pais, Rosa e Antonio, que me deram muito amor, carinho, sou muito grato a eles, eu os amo muito.

Aos meus irmãos, Wemerson e Wesley, pela força e compreensão.

À minha adorada namorada, Eliany, por sua compreensão e apoio na conquista deste trabalho.

Aos companheiros, Clevan, Jaime e Júnior, foi muito bom dividir a casa e conhecer um pouquinho de vocês.

Ao meu orientador Prof. Marcus Poggi, que me auxiliou muito nestes anos que passamos juntos e principalmente pela sua amizade.

Aos amigos de batalha, Elaine, Hélvio, Humberto e Victor, que talvez não poderia ter concluído este trabalho sem nossas ilimitadas discussões.

À todos os colegas que tive o prazer de conhecer e conviver estes últimos anos, aos funcionários do DCC pela eficiência e competência nos serviços prestados.

Aos orgão de fomento CNPq e FAEP pelo apoio financeiro.

Resumo

Este trabalho tem como tema central a proposta de um método de resolução eficiente do Problema de Máxima Satisfatibilidade Ponderada. Esta proposta tem sua motivação em uma classe de instâncias deste problema que pode ser resolvida em tempo polinomial.

A classe de interesse neste trabalho é a das instâncias cujo grafo de co-ocorrência associado é uma *k*-árvore. O algoritmo capaz de resolvê-lo em tempo linear é um método algébrico conhecido como Algoritmo Fundamental de Hammer, Rosenberg e Rudeanu em sua versão modificada proposta por Crama, Hansen e Jaumard.

A abordagem deste trabalho está na utilização de métodos enumerativos, *branch & bound* como são mais frequentemente citados, de modo a transformar instâncias gerais em instâncias pertencentes à classe de interesse. Deste modo, tanto a enumeração como a resolução algébrica podem ter suas tarefas simplificadas.

Três estratégias para esta combinação algébrico-enumerativa foram propostas e implementadas. Os algoritmos resultantes foram extensivamente testados em diferentes conjuntos de instâncias, sendo seus resultados comparados com implementações dos algoritmos puramente algébrico e puramente enumerativo.

Esse trabalho apresenta também uma generalização dos limites superiores propostos por Hansen para programação não-linear 0-1 para o caso em que variáveis complementadas são incorporadas aos termos. Esses limites além de integrados ao método proposto, foram implementados e testados individualmente.

Abstract

The proposal of an efficient resolution on method for the Maximum Satisfiability Problem is the object of this work. The motivation for this proposal relies on a special class of instances that can be solved by a linear time algorithm.

This class is the one for which the co-occurrence graph associated to the instance is a partial k-tree. It was shown by Crama, Hansen and Jaumard how the Basic Algorithm of Hammer, Rosenberg and Rudeanu, an algebraic method, an solve these instances in linear time.

The approach here proposed consists in applying an enumerative method to derive from any general instance several linear time instances that constitute a partition of the solution space. This allows the enumerative and the algebraic approaches to have their tasks simplified.

Three strategies of this algebraic-enumerative combination where proposed and implemented. The resulting algorithm where tested on several different data sets. The results where compared with both algebraic and enumerative pure approaches.

This work also generalizes Hansen's upper bounds and penalties for non-linear 0-1 programming to the case where complemented and uncomplement variables appear in the product terms. These where also implemented, tested and incorporated to the combined codes.

Conteúdo

Agradecimentos vii

Resumo viii

Abstract ix

Capítulo 1 *Introdução* 1

1.1 Objetivo 1

1.2 Definição do problema 2

1.2.1 Formulação Matemática 5

1.3 Aplicações 7

1.3.1 Problema de Detecção de Falhas 7

1.3.2 Inconsistência de Base de dados em Sistemas Especialistas 8

1.3.3 Consistência em Consultas de Base de dados 8

1.4 Organização da tese 9

Capítulo 2 *Revisão Bibliográfica* 10

2.1 Introdução 11

2.2 Métodos Algébricos 11
2.2.1 Introdução 11
2.2.2 O Algoritmo Fundamental 12
2.3 Métodos Enumerativos 21
2.3.1 Introdução 21
2.3.2 O Algoritmo de Enumeração Implícita 21
2.4 Métodos Aproximados 23
2.4.1 Introdução 23
2.4.2 Heurísticas de Construção 24
2.4.3 Meta-heurísticas 27
2.5 Sumário 31

Capítulo 3 *Combinação Algébrico-Enumerativa* 32

3.1 Introdução 33
3.2 O Algoritmo ALGEN 35
3.3 Complexidade do ALGEN 36
3.4 Relação entre k e $|N_f|$ 38
3.5 Determinação dos Limites e Penalidades 39
3.5.1 Cálculo das penalidades 39
3.6 Outras Estratégias de Combinação 43
3.6.1 ALGEN-1 43
3.6.2 ALGEN-2 45
3.6.3 Comparação das estratégias 46
3.7 Técnicas Computacionais 47
3.7.1 Representação interna da função objetivo 47
3.7.2 Ramificação na árvore de busca 48
3.7.3 Eficiência no cálculo das penalidades 50
3.7.4 Agrupamento de termos idênticos 50
3.8 Sumário 51

Capítulo 4 *Análise das Estratégias para o ALGEN* 53

4.1 Introdução 54
4.2 Definições 54
4.3 O Problema LPKT 56
4.4 Heurísticas para o LPKT 57
4.4.1 Heurística Greedy 58
4.4.2 Heurística A 59
4.4.3 Heurística B 60
4.4.4 Heurística *Try k-width* 61

4.5 Comparações entre as Heurísticas 62
4.5.1 Variando k 63
4.5.2 Variando o tamanho do conjunto de vértices 65
4.5.3 Variando a densidade do grafo 66
4.5.4 Análise do tempo de cpu 68
4.6 Sensibilidade ao valor k 70
4.7 Conclusão 72

Capítulo 5 *Resultados Computacionais* 73
5.1 Introdução 74
5.2 Geração das instâncias para os testes 75
5.3 Comparação dos Limites Superiores 76
5.4 Sensibilidade da Solução Inicial 76
5.5 Resultados do Algoritmo Fundamental 77
5.6 Resultados do Algoritmo Enumerativo 79
5.7 Resultados do ALGEN 80
5.7.1 Variando o tamanho da k-árvore 80
5.7.2 Variando o tamanho do conjunto de variáveis 82
5.8 Resultados do ALGEN-1 84
5.8.1 Verificando 3-SAT 84
5.8.2 Maximizando 3-SAT 85
5.8.3 Maximizando r-SAT 90
5.9 Resultados do ALGEN-2 91
5.9.1 Variando o tamanho da k-árvore 91
5.9.2 Variando o tamanho do conjunto de variáveis e termos 93
5.10 Conclusões 95

Capítulo 6 *Conclusões* 96
6.1 Contribuições 96
6.2 Extensões 97

Apêndice A *Limites Superiores* 98

Referências 106

Lista de Figuras

Figura 1.1 Exemplo do problema de detecção de falhas. 8
Figura 2.1 Grafo de co-ocorrência referente a equação 2.1. 15
Figura 2.2 Conjunto formado pela eliminação do vértice x_1 15
Figura 2.3 Os vértices sombreados são vértices simpliciais. 16
Figura 2.4 Arestas do conjunto $F^{\alpha}=\{(x_1, x_5), (x_3, x_4)\}$ 16
Figura 2.5 Grafo de co-ocorrência de f. 18
Figura 2.6 Grafo de co-ocorrência corresponde a f_2, após a eliminação de x_6 18
Figura 2.7 Árvore de busca do algoritmo de enumeração. 22
Figura 3.1 Representação da complexidade do ALGEN. 38
Figura 3.2 Representações internas da função objetivo $f(x)$ correspondente a equação 3.3. 49
Figura 3.3 Apresentação do Algoritmo ALGEN. 52
Figura 4.1 2-árvore com no máximo 6 vértices. 55
Figura 4.2 3-árvore com no máximo 6 vértices. 56
Figura 4.3 Grafo de co-ocorrência G. 57
Figura 4.4 Desempenho das heurísticas durante a variação do tamanho da k-árvore, considerando o número de vértices e a densidade do grafo fixados em 80 e 10%, respectivamente. 64
Figura 4.5 Qualidade das soluções das heurísticas sobre a variação dos vértices utilizando-se valores fixos para a densidade e k-árvore iguais à 10% e 11-árvore, respectivamente. 65
Figura 4.6 Execução da heurística A para instância de 80 vértices no grafo o tamanho da maior k-árvore deve ser 11, com variação da densidade do grafo. 67
Figura 4.7 Qualidade dos resultados obtidos com a execução das heurísticas na verificação do desempenho das mesmas considerando densidade do grafo e tamanho da k-árvore fixados e três

variações do tamanho do conjunto de vértices. Devido o fato do tempo de processamento da heurística ℬ ser muito próximo de zero e não ser possível plotar seus pontos neste gráfico. . 69

Figura 4.8 Análise de pior caso para o algoritmo ALGEN. O algoritmo possui complexidade linear para um *k* fixo em 18, uma complexidade quadrática para *k* fixado em 12 e complexidade cúbica para *k* fixado em 6. 70

Figura 4.9 Desempenho dos algoritmos segundo a sensibilidade do valor de *k* a medida que o número de vértices cresce. 71

Figura 5.1 Qualidade das soluções dada pela execução do Algoritmo Fundamental, considerando o número de termos igual ao número de variáveis. 78

Figura 5.2 Qualidade das soluções dada pela execução do algoritmo ALGEN, resolvendo w-MAX-3SAT de 80x80 com o tamanho da *k*-árvore variável. 81

Figura 5.3 Desempenho do ALGEN para a variação do número de variáveis, tal que o número de termos é igual ao de variáveis e *k*-árvore fixada em 9. 83

Figura 5.4 Qualidade das soluções para a execução do ALGEN1 com a variação do tamanho da *k*-árvore. Soluções correspondentes às instâncias 80x80 e 50x100, apresentadas na tabela 5.8. 86

Figura 5.5 Desempenho do algoritmo durante a variação do número de variáveis. Este gráfico representa os dados apresentados na tabela 5.9. Devido o fato dos valores dos tempo de processamento da curva *nxn* serem muito próximos de zero, não é possível sua visualização no gráfico acima. 88

Figura 5.6 Desempenho do algoritmo para a variação do número de variáveis. Vale ressaltar que a curva *nxn* sobrepõe o eixo *n* devido o valores do eixo tempo serem muito próximos de zero. 89

Figura 5.7 Desempenho do algoritmo perante a variação do tamanho da *k*-árvore. Este gráfico representa do dados apresentados na tabela 5.11. 92

Figura 5.8 Desempenho do algoritmo ALGEN2 segundo a variação do número de variáveis, dados representados na tabela 5.12. 94

Lista de Tabelas

Tabela 4.1 Comparações das heurísticas com respeito ao parâmetro k. A linha sombreada indica a média, sobre 10 instâncias, para o tamanho do conjunto N_f 63

Tabela 4.2 Resultados obtidos pelas heurísticas com respeito ao paramêtro do número de vértices do grafo. A linha sombreada indica a média do tamanho do conjunto N_f sobre a execução de 10 instâncias..... 65

Tabela 4.3 Representação do desempenho da heurística A durante a variação da densidade do grafo no número de arestas. A linha sombreada indica a média do tamanho do conjunto N_f para a execução de 10 instâncias..... 66

Tabela 4.4 Desempenho das heurísticas referente ao tempo de processamento gasto..... 68

Tabela 4.5 Qualidade das soluções produzidas para a análise de desempenho dos algoritmo segundo a variação do valor de k..... 71

Tabela 5.1 Tabela produzida pelos resultados fornecidos pelos limites superiores $\bar{z}_2$, $\bar{z}_3$ e $\bar{z}_4$, para 10 instâncias MAX-3SAT tamanho $n = 40$ e $m = 80$. O algoritmo possui a mesma solução inicial no processamento das instâncias com cada limite..... 76

Tabela 5.2 Resultados obtidos para se efetuar comparações de forma a perceber a sensibilidade de se possuir uma solução inicial "boa"..... 77

Tabela 5.3 Resultados obtidos com a execução do Algoritmo Fundamental na sua forma revisitada, ou seja, BASIC ALGORITHM REVISITED..... 77

Tabela 5.4 Tabela definida com os resultados obtidos com a execução do algoritmo enumerativo puro, utilizado como parte do algoritmo combinado. As linhas sombreadas significam o cálculo das médias..... 79

Tabela 5.5 Resultados obtidos com instâncias 3-SAT, três variáveis por termo. Os itens em destaque apresentam o valor médio do algoritmo quando executado sobre 10 instâncias..... 80

Tabela 5.6 Tabela definida para se observar o desempenho do algoritmo a medida que aumenta-se o valor

de m (número de termos) em função de n. A média dos resultados é obtida sobre 10 instâncias diferentes. 82

Tabela 5.7 Resultados obtidos com a execução do ALGEN1 para verificar se o problema é satisfatível ou não, a linha em destaque é a média sobre 10 execuções. 84

Tabela 5.8 Resultados obtidos com instâncias 3-SAT, três variáveis por termo. Os itens em destaque apresentam o valor médio do algoritmo quando executado sobre 10 instâncias. 85

Tabela 5.9 Tabela definida para se observar o desempenho do algoritmo a medida que aumenta-se o valor de m (número de termos) em função do n. 87

Tabela 5.10 Qualidade das soluções na execução do ALGEN1 quando os termos possuem até 5 literais . 90

Tabela 5.11 Resultados obtidos com instâncias 3-SAT, três variáveis por termo. Os itens em destaque apresentam o valor médio do algoritmo quando executado sobre 10 instâncias. 91

Tabela 5.12 Tabela definida para demonstrar a qualidade das soluções encontradas pelo algoritmo ALGEN2 a medida que aumenta-se o valor de m (número de termos) em função do n. Pode-se notar ainda a variação sofrida com o crescimento do número de variáveis. 93

Capítulo 1

Introdução

1.1 Objetivo

O principal objetivo deste trabalho está em estudar e desenvolver técnicas para a resolução de problemas de programação não-linear em variáveis 0-1 sem restrições. Neste trabalho esse problema será referenciado como o problema de máxima satisfatibilidade ponderada e será denotado por w-MAXSAT.

O problema w-MAXSAT é uma generalização do problema de satisfatibilidade (SAT), um dos problemas exaustivamente estudados em otimização combinatória. Este interesse pode ser atribuído inicialmente à demonstração de sua pertinência à classe *NP-completo*, por Cook 1971 e mais tarde ao seu papel fundamental no campo da Inteligência Artificial.

O problema SAT pode ser enunciado da seguinte forma: dado um conjunto de variáveis lógicas e subconjuntos deste conjunto de variáveis (complementadas ou não), determine se existe uma atribuição de valores "verdadeiro" ou "falso" às variáveis lógicas tal que todo subconjunto possua pelo menos uma variável ou com valor "verdadeiro", caso esta não esteja complementada, ou "falso", caso esteja. Ou seja, baseia-se em verificar se todos estes subconjuntos podem ser satisfeitos simultaneamente. Uma expressão booleana está na forma normal conjuntiva (CNF) se sua composição é um produto booleano de subconjuntos do conjunto de variáveis. Essa expressão é satisfeita se, e somente se, existe alguma atribuição de valores "verdadeiro" e "falso" às variáveis, de tal forma que seu valor seja avaliado verdadeiro. Assim, uma expressão booleana corresponde a uma instância do problema SAT. No problema w-MAXSAT, atribui-se pesos às cláusulas e o que se deseja é encontrar uma atribuição às variáveis que forneça o máximo para o somatório dos

pesos. Observe que os pesos associados às cláusulas podem ser negativos. Assim a satisfatibilidade de todo o conjunto de cláusulas pode não corresponder à melhor solução.

Além da importância teórica, como em Lógica e Complexidade Computacional, os problemas SAT e w-MAXSAT, possuem também importância prática como em testes de VLSI, consistência de banco de dados e base de conhecimentos em sistemas especialistas. Tanto o problema SAT quanto o w-MAXSAT podem modelar uma grande variedade de problemas práticos de otimização, como apresentado na seção 1.3.

Devido a equivalência do problema w-MAXSAT ao de programação não-linear 0-1 (PNL-01) sem restrições e este (último) possuir uma vasta literatura, pode-se dispor do que foi desenvolvido na sua resolução para resolver w-MAXSAT. Dentro desta literatura encontram-se métodos que se utilizam de manipulações algébricas para encontrar sua solução ótima conhecidos por métodos algébricos. Como exemplo destes métodos pode-se citar o Algoritmo Fundamental de Hammer, Rosenberg e Rudeanu [HRR63]. Uma evolução deste algoritmo foi proposta por Crama, Hansen e Jaumard [CHJ90]. A abordagem teve sua motivação nos casos polinomiais de w-MAXSAT. Nestes casos, este algoritmo é capaz de explorar a existência de estruturas de *k*-árvores parciais no grafo de co-ocorrência definido pela instância a ser resolvida. Entretanto a eficiência deste algoritmo é reduzida para instâncias gerais. Na prática, a resolução de w-MAXSAT pelo método de enumeração implícita tem demonstrado uma eficiência superior a das outras abordagens.

O cerne desta dissertação é a combinação dos métodos algébricos com os enumerativos e confrontar os seus resultados com os de outros algoritmos anteriormente propostos. As próximas seções descrevem a definição formal do problema w-MAXSAT, algumas de suas aplicações e, finalmente, a organização da dissertação.

1.2 *Definição do problema*

Sejam $X = \{x_1, x_2, \ldots, x_n\}$, um conjunto de variáveis booleanas, a, uma atribuição de valores a X, e t, uma função $t: X \rightarrow \{True, False\}$. Se $t(x) = True$, diz-se que x é verdadeiro sobre t; se $t(x) = False$, diz-se que x é falso sobre t. Se x_i é uma variável em X, então x_i e $\bar{x}_i$ são **literais** sobre X. O literal x_i é verdadeiro se, e somente se, a variável x_i é verdadeira; o literal $\bar{x}_i$ é verdadeiro se, e somente se, a variável x_i é falsa.

Uma **cláusula** sobre X, é um conjunto de literais, tal como $\{x_1, \bar{x}_3, x_5\}$. Ela representa a disjunção dos literais e é satisfeita por uma atribuição verdadeira se, e somente se, pelo menos um de seus literais é verdadeiro sobre esta atribuição. A cláusula acima é satisfeita por t a menos que $a(x_1) = F$, $a(x_3) = T$, e $a(x_5) = F$. Uma **coleção** C de

cláusulas sobre X é satisfeita se, e somente se, existe alguma atribuição verdadeira para X que satisfaça simultaneamente todas as cláusulas pertencentes a C. Tal atribuição é chamada uma **atribuição verdadeira satisfatível** para C.

O problema SAT pode ser definido como segue:

Instância: *um conjunto X de variáveis booleanas, uma coleção C de cláusulas sobre X.*

Pergunta: *existe uma atribuição verdadeira para C?*

Observe o seguinte exemplo:

> Seja $C = (x_1 \vee x_2) \cdot (\bar{x}_3 \vee \bar{x}_4) \cdot (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \cdot (x_2 \vee x_3 \vee \bar{x}_4)$; o que se deseja é uma atribuição para x_1, x_2, x_3 e x_4, tal que todas as cláusulas de C sejam satisfeitas.
>
> Para isto, pode-se ter $x_1 = True$; $x_2 = True$; $x_3 = False$ e $x_4 = True$.

Considere agora o seguinte conjunto de cláusulas:

> Seja $C = (x_1 \vee x_2) \cdot (\bar{x}_3 \vee x_4) \cdot (\bar{x}_1 \vee x_2) \cdot (x_1 \vee \bar{x}_2) \cdot (\bar{x}_1 \vee \bar{x}_2)$; como no caso anterior, deseja-se uma atribuição para x_1, x_2, x_3 e x_4 tal que todas cláusulas de C sejam satisfeitas.
>
> Não existe uma atribuição para os elementos de X que satisfaça todas as cláusulas do conjunto C.

O contexto de Inteligência Artificial possui um interesse particular em extensões do Problema SAT. Uma classe abrangente de Sistemas Especialistas tem sua base de conhecimento definida como um conjunto de regras do tipo "se [condição] então [conseqüência]" e um conjunto de eventos realizados (*"Rule Based Expert Systems"* [BS85]). Os eventos realizados e a realizar, aqueles sobre os quais se quer utilizar o conhecimento obtido, são representados por variáveis lógicas (ou seja, cada evento ocorre ou não).

Da mesma forma o conhecimento representado por "se [evento x_1 ocorre] então [ocorre x_2]", ou seja, $x_1 \rightarrow x_2$ é incorporado à base de conhecimento ao se incluir a cláusula $x_1 \vee x_2$. Eventualmente a base de conhecimento é inconsistente e a resposta negativa ao problema SAT associado detecta inconsistência no conhecimento utilizado. O problema de eliminação desta inconsistência se reduz a determinar que regras devem ser retiradas da base. Diversas são as estratégias para o tratamento deste problema; procura-se com maior freqüência reter o máximo do conhecimento. Deste modo, a questão "qual é o máximo de cláusulas (regras) que podem ser satisfeitas?" vem a mente de forma natural. Define-se então um problema em que se busca satisfazer o máximo de cláusulas e não apenas verificar se todas podem ser satisfeitas.

Este problema é conhecido na literatura como o Problema de Máxima Satisfatibilidade, definido como segue:

Instância: *conjunto X de variáveis booleanas, uma coleção C de cláusulas sobre X e um inteiro* $\alpha \leq |C|$.

Pergunta: *existe uma atribuição verdadeira para X que satisfaça pelo menos* α *das cláusulas de C?*

Assim, no exemplo anterior, existe uma atribuição verdadeira para um $\alpha = 4$, isto é, pelo menos quatro cláusulas seriam satisfeitas, com $x_1 = T, x_2 = T, x_3 = F$ e $x_4 = F$, ou $x_1 = T, x_2 = F, x_3 = T$ e $x_4 = T$.

Pensando-se em termos de Sistemas Especialistas é natural que certas regras sejam mais importantes que outras, seja por terem sido mais testadas, seja por terem sido concebidas com maior embasamento teórico. Deste modo, se pesos são associados às regras de acordo com a sua confiança, então pode-se definir um problema para tentar satisfazer o máximo do somatório dos pesos das cláusulas.

Este problema é conhecido por Problema de Máxima Satisfatibilidade Ponderada, definido como segue:

Instância: *conjunto X de variáveis booleanas, uma coleção C de cláusulas sobre X e pesos* $\omega(i) \in Z, i \in C$ *associados às cláusulas de C.*

Pergunta: *existe uma atribuição verdadeira para X que maximize* $\sum_{i \in C_S(x)} \omega(i)$, *onde* C_S *é o subconjunto de C satisfeito pela atribuição sobre X?*

Considere o seguinte exemplo:

Sejam $C = \{c_1 = (x_1 \vee x_2), c_2 = (\bar{x}_3 \vee x_4), c_3 = (\bar{x}_1 \vee x_2), c_4 = (x_1 \vee \bar{x}_2), c_5 = (\bar{x}_1 \vee \bar{x}_2)\}$ uma coleção de cláusulas e $\omega_1 = 120$, $\omega_2 = 40$, $\omega_3 = -30$, $\omega_4 = 60$ e $\omega_5 = -50$, os pesos das cláusulas. Utilizando-se das mesmas atribuições do exemplo anterior tem-se que o máximo dos pesos das cláusulas encontrado com $x_1 = T, x_2 = T, x_3 = F$ e $x_4 = F$, é $\sum \omega_i = 190$.

Enquanto, para $x_1 = T, x_2 = F, x_3 = T$ e $x_4 = T$, obtém-se $\sum \omega_i = 170$.

O problema SAT possui duas classes de instâncias, muito conhecidas, que são resolvidas em tempo polinomial: a primeira, 2-Satisfatibilidade (2SAT), onde cada cláusula contém no máximo dois literais. Aspvall, Plass e Tarjan [APT79], propuseram um algoritmo para resolver o 2SAT em tempo $O(|C|)$. A segunda classe, é

Horn Satisfatibilidade (HORN-SAT), onde as instâncias desta classe contém cláusulas do tipo *Horn*, isto é, a cláusula possui somente um literal positivo (não complementado). Dowling e Gallier [DG84], propuseram um algoritmo, para HORN-SAT, em tempo $O(l)$, onde l é o tamanho do problema, isto é, o número total de ocorrência de literais. No caso geral, o problema SAT pertence à classe *NP-completo* [Cook71].

O problema MAXSAT pode facilmente ser demonstrado pertencer à classe *NP-completo*; basta considerar $\alpha = |C|$ e obtém-se o problema SAT.

Analogamente, pode-se mostrar a pertinência do problema w-MAXSAT à classe *NP-completo*, basta, para tal, colocar os pesos das cláusulas iguais a 1, obtendo assim o problema MAXSAT.

O problema w-MAXSAT também é referenciado na literatura como programação pseudo-booleana (Hammer e Rudeanu [HR68]), maximização de uma função real definida em variáveis 0-1, ou simplesmente MAXSAT (Georgakopoulos, Kavvadias e Papadimitriou [GKP88]).

1.2.1 *Formulação Matemática*

Considere um conjunto $C = \{C_1, C_2, ..., C_m\}$ de cláusulas definidas sobre um conjunto $X = \{x_1, x_2, ..., x_n\}$ de variáveis booleanas. Seja $C_S(X) \subseteq C$ o conjunto das cláusulas satisfeitas por uma atribuição de valores a X. A formulação do problema w-MAXSAT como um problema de decisão é enunciada da seguinte forma:

Versão decisão: *dados um conjunto X de variáveis booleanas, um conjunto C de cláusulas, pesos $\omega(i) \in Z, i \in C$ associados às cláusulas e um número $\alpha \in [\Sigma min(0, \omega_i), \Sigma max(0, \omega_i)]$. Deseja-se saber se existe uma atribuição para as variáveis de X, tal que*

$$\sum_{i \in C_S(X) \subset C} \omega(i) \geq \alpha$$

O maior interesse está na formulação do w-MAXSAT, como sendo um problema de otimização, enunciado como segue:

Versão otimização: *dado um conjunto C de cláusulas sobre um conjunto X de variáveis booleanas e pesos $\omega(i) \in Z, i \in C$ associados às cláusulas de C. Determine uma atribuição verdadeira para X que maximize $\sum_{i \in C_S(X)} \omega(i)$.*

1.2.1.1 Programação não-linear 0-1 sem restrições

Esta formulação também pode ser referenciada como problema de maximização pseudo-booleana ponderada sem restrições, observe a equação 1.1.

$$Max \qquad f(x) = \sum_{k=1}^{m} \omega_{N_k \bar{N}_k} \prod_{j \in N_k} x_j \prod_{j \in \bar{N}_k} \bar{x}_j$$

$$\Gamma \subseteq \left\{ (N_k, \bar{N}_k) \,\middle|\, (N_k \cup \bar{N}_k) \subseteq \{1, 2, \ldots, n\}, N_k \cap \bar{N}_k = \varnothing \right\} \qquad \text{(Eq 1.1)}$$

$$x_j = \{0, 1\} \qquad j = 1, 2, \ldots, n$$

$$\bar{x}_j = 1 - x_j$$

$$|\Gamma| = m, \omega_{N_k \bar{N}_k} \neq 0$$

onde, N_k (resp. $\bar{N}_k$) é o conjunto dos índices das variáveis não-complementadas (resp. complementadas) no termo k. Note que no caso de um termo constante tem-se $N_k = \bar{N}_k = \varnothing$. Como exemplo desta formulação considere a função:

$$Max \qquad f(x) = 5x_1 \cdot \bar{x}_2 - 7x_2 \cdot \bar{x}_3 + 4\bar{x}_1 \cdot x_3 \cdot x_4 \text{ ; onde,}$$

$$N_1 = \{x_1\}, \bar{N}_1 = \{\bar{x}_2\}, N_2 = \{x_2\}, \bar{N}_2 = \{\bar{x}_3\}, N_3 = \{x_3, x_4\}, \bar{N}_3 = \{\bar{x}_1\}$$

$$\omega_1 = 5, \omega_2 = -7, \omega_3 = 4$$

A formulação apresentada na equação 1.1 será utilizada no transcorrer desta dissertação. Uma justificativa desta escolha é que a representação na forma normal conjuntiva, utilizada no início desta dissertação, pode ser transformada na formulação acima. Para tal, utiliza-se da lei de De Morgan, da álgebra booleana, e do fato de que em variáveis 0-1 tem-se $(x_1 \wedge x_2) = (x_1 \cdot x_2)$ e $\bar{x}_i = (1 - x_i)$.

Considere o seguinte exemplo, onde as cláusulas são $x_1 \vee \bar{x}_2$ e $x_2 \vee \bar{x}_3$ com pesos 5 e -7 respectivamente. A função pseudo-booleana correspondente é:

$$Max \qquad f(x) = 5(x_1 \vee \bar{x}_2) + (-7)(x_2 \vee \bar{x}_3) \qquad \text{(Eq 1.2)}$$

Pela lei de De Morgan, $\overline{A \vee B} = \bar{A} \wedge \bar{B}$ ver [HR68], obtém-se:

$$Max \qquad f_1(x) = (-2) + (-5)(\bar{x}_1 \wedge x_2) + (7)(\bar{x}_2 \wedge x_3) \qquad \text{(Eq 1.3)}$$

Portanto, a equação 1.2 é equivalente à equação 1.3.

Agora utilizando-se do fato das variáveis serem 0-1, obtém-se a formulação dada pela equação 1.1 para a equação 1.3:

$$MAX \qquad f_2(x) = -2 - 5\bar{x}_1 x_2 + 7\bar{x}_2 x_3$$

$$N_1 = \bar{N}_1 = \varnothing, N_2 = \{x_2\}, \bar{N}_2 = \{\bar{x}_1\}, N_3 = \{x_3\}, \bar{N}_3 = \{\bar{x}_2\}$$

$$\omega_1 = -2, \omega_2 = -5, \omega_3 = 7$$

1.3 Aplicações

1.3.1 Problema de Detecção de Falhas

O crescimento da densidade dos circuitos em chips VLSI torna, a cada dia, o problema de detecção de falhas em um *chip* produzido mais complexo [Roth66, Goel81, BF76]. Deste modo, o interesse em métodos para resolução desse problema é cada vez maior. Esse problema pode ser enunciado da seguinte forma: existe algum padrão de entrada e/ou saída que pode detectar num circuito combinacional C uma falha de emperramento a 0 ou a 1? Fuji [Fuji90] demonstrou que esse problema é *NP-completo*. Uma maneira de resolver esse problema é considerar uma formulação do problema SAT, onde as linhas de entrada do circuito representam as variáveis e as portas lógicas representam os termos. Considere o exemplo seguinte:

> Seja F uma expressão formada por cláusulas CNF, onde cada variável aparece no máximo 3 vezes. Sem perda de generalidade, sejam $C_1, C_2, ..., C_p$ e $C_{p+1}, C_{p+2}, ..., C_q$ as cláusulas com variáveis não-complementadas e complementadas, respectivamente. A partir da expressão F constrói-se um circuito combinacional, apresentado na figura 1.1, de forma que:
>
> • $O_1, O_2, ..., O_p$ são portas **OU** e correspondem as cláusulas $C_1, C_2, ..., C_p$ tal que as linhas de entrada de O_i correspondem as variáveis de C_i. Por exemplo, considere uma cláusula $C_i = x_1 + x_2 + x_3$, então a saída de O_i será $x_1 + x_2 + x_3$.
>
> • $A_1, A_2, ..., A_{q-p}$ são portas **E** e correspondem às cláusulas $C_{p+1}, C_{p+2}, ..., C_q$, tal que as linhas de entrada de A_j correspondem as variáveis de C_j. Por exemplo se $C_j = \bar{x}_1 + \bar{x}_2$, então A_j é $x_1 \cdot x_2$.
>
> A falha de emperramento a 0 (*FALSE*) na entrada X_o é detectada se, e somente se, existe um teste tal que todas as saídas das portas O_i são 1 (*TRUE*) e de A_j são 0 (*FALSE*). Portanto a falha de X_o, fixado em 0 (*FALSE*), é detectável se, somente se, a expressão F é satisfatível.

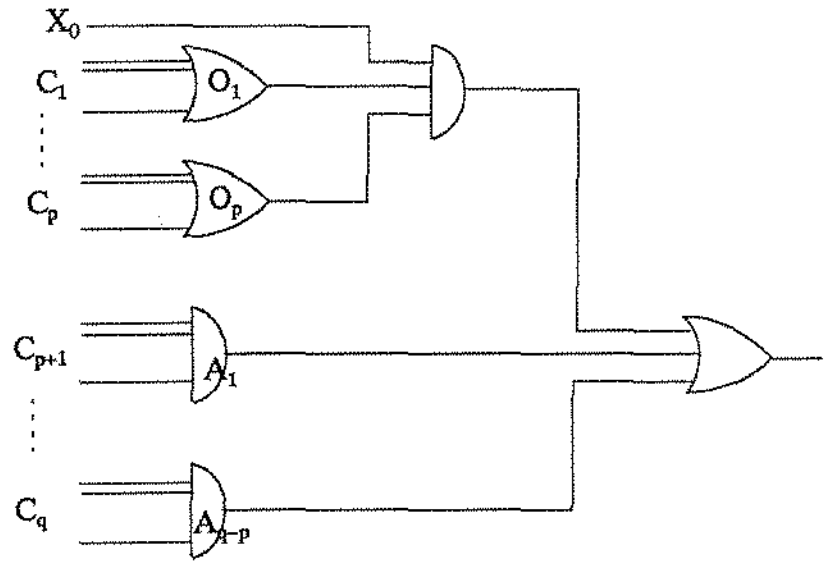


Figura 1.1 Exemplo do problema de detecção de falhas.

1.3.2 *Inconsistência de Base de dados em Sistemas Especialistas*

Em sistemas especialistas [HWL83, NP+85], base de dados são utilizadas como informação para o estudo sobre os objetos. Essas bases são usualmente expressadas por implicações lógicas. Neste caso, utiliza-se dos métodos de álgebra booleana para reescrever estas implicações como um conjunto de cláusulas. Como ilustração, se as informações estão representadas pelas seguintes implicações $x_1 \rightarrow x_2$, $x_1 \rightarrow x_3$, $\bar{x}_4 \rightarrow x_2$, utilizando-se a regra de eliminação condicional de álgebra booleana, obtém-se $\bar{x}_1 \vee x_2$, $\bar{x}_1 \vee x_3$, $\bar{x}_4 \vee x_2$, as quais constituem o conjunto de cláusulas. Um importante problema prático é verificar a consistência da base de dados depois de uma atualização feita pela adição e deleção de algumas das implicações. Se a base de dados não é consistente, é desejável encontrar o menor conjunto de implicações que será removido, ou todos os conjuntos mínimos de implicações, em seqüência, a fim de restaurar a consistência. Assim, este conjunto pode ser obtido com a resolução do problema MAXSAT formado pelas cláusulas correspondentes às implicações da base de dados.

1.3.3 *Consistência em Consultas de Base de dados*

Esse problema [GMN84, ASM85], que originou-se em banco de dados evolutivos, usam ferramentas de programação lógica para armazenar seqüencialmente as informações. Tipicamente, num problema mais geral de restrições de integridade, deseja-se saber a razão na qual o sistema não pode responder algumas consultas. A lógica pode ser utilizada como uma representação da linguagem e como um sistema de inferência. Assim, com este mapeamento pode-se caracterizar a resposta a consulta

encontrando uma resolução para o problema SAT. Devido a grandes números de fatos na base de dados corrente e também pelas numerosas atualizações dinâmicas, este problema é difícil para ser resolvido na prática; assim heurísticas eficientes são necessárias.

1.4 Organização da tese

O segundo capítulo, apresenta uma breve revisão de alguns dos métodos exatos e aproximados existentes para a resolução do problema w-MAXSAT, formulado na seção 1.2.1.1. Dentro dos métodos exatos, tem-se a motivação no algoritmo algébrico pela eficiência obtida em algumas classes do problema w-MAXSAT. A seguir estuda-se a complexidade desse algoritmo. Ainda na classe dos métodos exatos o estudo concentra-se em algoritmos de enumeração implícita. Na classe dos métodos aproximados são abordadas heurísticas de construção, de Busca Tabu e *Simmulated Annealing*.

O terceiro capítulo apresenta um algoritmo híbrido para resolução do problema w-MAXSAT, denominado ALGEN. Este algoritmo baseia-se na combinação de um algoritmo algébrico com um algoritmo de enumeração implícita. O capítulo descreve ainda uma formulação do algoritmo de enumeração implícita utilizado na combinação. Esta se utiliza de generalizações de cálculo dos limites superiores, definidos por Hansen [Hans79], para a função do problema representado pela equação 1.1.

A definição de estratégias para o algoritmo ALGEN pode ser vista como um problema aparentemente novo na literatura. Este problema, que corresponde a uma generalização do Problema de Verificação da Existência de uma *k*-árvore apresentado por Arnborg, Corneil e Proskurowski em [ACP87], é abordado no quarto capítulo. Ainda neste capítulo, são descritas algumas heurísticas para a sua resolução aproximada.

O quinto capítulo apresenta os resultados das experiências computacionais descritos por tabelas e gráficos de comparações e de comentários destes resultados.

No sexto capítulo são apresentadas as conclusões e as contribuições deste trabalho.

E finalmente no apêndice A, apresenta-se a demonstração formal dos limites superiores utilizados pelo algoritmo de enumeração implícita, que faz parte do algoritmo proposto ALGEN.

Capítulo 2

Revisão Bibliográfica

Este capítulo, apresenta alguns algoritmos da literatura para a resolução do problema w-MAXSAT. A apresentação de uma lista exaustiva de artigos sobre este problema seria muito extensa e está além dos objetivos desta dissertação. Deste modo, discute-se aqui apenas as principais técnicas de resolução.

Inicialmente apresenta-se a classe dos algoritmos algébricos, exemplificada pelo Algoritmo Fundamental, de Hammer, Rosenberg e Rudeanu. Em seguida, são descritas as características básicas de um algoritmo de enumeração implícita.

Esse capítulo aborda também alguns métodos aproximados, algoritmos heurísticos, para a resolução do problema w-MAXSAT, onde são apresentadas heurísticas de construção e meta-heurísticas.

2.1 Introdução

Existe uma variedade de técnicas que podem ser utilizadas para a resolução de w-MAXSAT. Este capítulo aborda, brevemente, dois métodos encontrados na literatura. Estes métodos podem ser classificados como: métodos exatos, aproximados, de linearização composto de algoritmos enumerativos e de planos de cortes. Vários desses métodos são mais adequados para resolver os casos com restrições para o problema w-MAXSAT, como linearização. Assim, o estudo concentra-se sobre os métodos exatos e aproximados que são eficientes em problemas com e sem restrições.

2.2 Métodos Algébricos

2.2.1 Introdução

Métodos algébricos são métodos exatos que se utilizam de manipulações algébricas para determinar a solução ótima para um problema. Um exemplo de método algébrico é o Algoritmo Fundamental, desenvolvido por Hammer, Rosenberg e Rudeanu [HRR63] e proposto há mais de 30 anos. Uma forma simplificada dada por Hammer e Rudeanu está apresentada no livro *"Boolean Methods in Operations Research and Related Areas"* [HR68]. Este algoritmo determina o máximo de uma função não-linear em variáveis 0-1 ou pseudo-booleana utilizando-se uma eliminação recursiva de variáveis. As condições de otimalidade local são exploradas para a produção, em cada iteração, de uma nova função dependente de menos variáveis e que tenha o mesmo valor ótimo do passo anterior.

Não se sabe de nenhum esforço de programação deste método. Atribui-se tal fato à grande dificuldade de implementação das manipulações de fórmulas booleanas. Além disso, o espaço de memória necessário (para armazenar resultados intermediários e mesmo a nova função) é frequentemente muito grande. Isto inviabiliza a resolução de problemas de porte médio nos computadores disponíveis na década de 70. Assim, com o aparecimento de novos métodos, o Algoritmo Fundamental foi deixado de lado até recentemente quando Crama, Hansen e Jaumard [CHJ90] prosseguiram no estudo deste algoritmo considerando os seguintes aspectos:

(I) o fato do Algoritmo Fundamental poder ser adaptado para solucionar eficientemente problemas pertencentes às classes particulares, mesmo não apresentando um bom desempenho para instâncias arbitrárias;

(II) a possível simplificação de alguns passos do Algoritmo Fundamental;

(III) a disponibilidade de computadores poderosos com o tamanho da memória elevada, implicando num aumento substancial do tamanho dos problemas que se poderia resolver.

2.2.2 *O Algoritmo Fundamental*

O problema w-MAXSAT é equivalente a maximização de um polinômio em variáveis booleanas complementadas ou não (conforme apresentado na seção 1.2.1). Desde modo, para resolução de w-MAXSAT, apresenta-se o esquema do Algoritmo Fundamental de Hammer, Rosenberg e Rudeanu. Seu procedimento combina a programação dinâmica de Bellman [Bell57] com técnicas booleanas, onde necessita-se a resolução de equações e inequações pseudo-booleanas.

O Algoritmo Fundamental se processa eliminando uma variável por iteração. A condição de otimalidade local é utilizada para determinar o valor ótimo da variável eliminada. Esse valor ótimo é dado por uma expressão booleana dependente somente das variáveis restantes. Em seguida, a função a ser maximizada é reescrita nas variáveis restantes substituindo-se a variável eliminada pela expressão calculada.

Apresenta-se a seguir, o esquema do Algoritmo Fundamental proposto por Hammer, Rosenberg e Rudeanu [HRR63] para a resolução de w-MAXSAT.

Algoritmo Fundamental

Seja $f_i(x_i, x_{i+1}, \ldots, x_n)$

Logo, como f_1 contém todas as n variáveis, $f_1 = f$.

Colocando-se em evidência x_1 na função f_1, obtém-se f_1 como segue:

$$f_1(x_1, x_2, \ldots, x_n) = x_1 \cdot g_1(x_2, x_3, \ldots, x_n) + h_1(x_2, x_3, \ldots, x_n)$$

onde as funções g_1 e h_1 não dependem de x_1.

A condição de otimalidade local requer que o valor ótimo de x_1 seja 1 se g_1 é positivo, 0 se g_1 é negativo e arbitrário quando g_1 se anula. Desde modo, x_1^* é dado pela função ϕ_i definida abaixo:

$$\phi_1(x_2, x_3, \ldots, x_n) = \begin{cases} 1, & se\ g_1(x_2, x_3, \ldots, x_n) > 0 \\ 0, & caso\ contrário \end{cases}$$

Assumindo-se que ϕ_i tenha sido obtida, pode-se escrever f_2 substituindo x_1 por ϕ_1,

$$f_2(x_2, x_3, \ldots, x_n) = f_1(\phi_1(x_2, x_3, \ldots, x_n), x_2, x_3, \ldots, x_n)$$

Reduzindo-se assim a maximização da função original em n variáveis para a de f_2, a qual depende somente de $n-1$ variáveis, $x_2, x_3, \ldots, x_n$.

Prossegue-se no processo de eliminação para $x_2, x_3, \ldots, x_{n-1}$, sucessivamente até a obtenção de uma função dependente somente de uma variável, x_n, representada por

$$f_n(x_n) = C_n x_n + D_n$$

$$x_n = \begin{cases} 1, & se\ C_n > 0 \\ 0, & caso\ contrário \end{cases}$$

Assim, conhecendo-se o valor de x_n^*, pode-se determinar $x_{n-1}^*, x_{n-2}^*, \ldots, x_1^*$, recursivamente, sendo

$$x_n^* = \begin{cases} 1 \\ 0 \end{cases}$$

$$x_{n-1}^* = \phi_{n-1}(x_n^*)$$

$$x_{n-2}^* = \phi_{n-2}(x_{n-1}^*, x_n^*)$$

$$\vdots$$

$$x_1^* = \phi_1(x_2^*, \ldots, x_{n-1}^*, x_n^*)$$

2.2.2.1 *Análise de Complexidade do Algoritmo Fundamental*

Observe que a eficiência desse algoritmo depende de quão rápido as funções booleanas $\phi_1, \phi_2, \ldots, \phi_{n-1}$ são determinadas, o que pode ser feito de várias formas. Uma forma simples, porém longa, de se obter g_i é considerar todas as possíveis atribuições, zero ou um, às variáveis $x_{i+1}, x_{i+2}, \ldots, x_n$. Então ϕ_i é definida como a soma de produtos de $l_{i+1}, l_{i+2}, \ldots, l_n$, onde l_i é um literal de x_i (x_i ou $\bar{x}_i$). Esses produtos correspondem a cada uma das atribuições de valores a $x_{i+1}, x_{i+2}, \ldots, x_n$ que possui g_i positivo. l_i será x_i quando x_i tiver valor 1 na atribuição correspondente e $\bar{x}_i$ caso contrário.

> Por exemplo:
> Se $g_1(x_2, x_3, x_4) = -4x_2 + 6x_3\bar{x}_4 - 1$ então as atribuições que tem g_1 positivo são $x_2 = T$, $x_3 = T$ e $x_4 = F$ ou $x_2 = F$, $x_3 = T$ e $x_4 = F$. Os produtos l_i's correspondentes são $x_2x_3\bar{x}_4$ e $\bar{x}_2x_3\bar{x}_4$.

A complexidade de determinar a função ϕ_i, desse modo, é exponencial no número de variáveis de g_i, pois é realizada uma enumeração completa. Considerando tais aspectos, Crama, Hansen e Jaumard [CHJ90] propuseram o uso da técnica *branch-and-bound* para reduzir as manipulações booleanas envolvidas no processo de eliminação, que entretanto também executa um número exponencial de operações no pior caso.

Considere S_i o conjunto das variáveis pertencentes a g_i, tal que $S_i \subseteq \{x_{i+1}, x_{i+2}, \ldots, x_n\}$, para $i = 1, 2, \ldots, n-1$.

Teorema 2.1 (Crama, Hansen e Jaumard [CHJ90]) O Algoritmo Fundamental resolve o problema w-MAXSAT em tempo linear, para toda instância de w-MAX-SAT que tenha k_{MAX} como limitante superior de $|S_i|$ para todo i.

> *Dem:* Pela hipótese, $|S_i|$ é limitado em todo o processo ($i = 1, 2, \ldots, n-1$), por uma constante k_{MAX}. O esforço computacional em cada passo da eliminação

tem, portanto, um limite superior constante da ordem de $2^{k_{MAX}}$. Como o processo de resolução termina após n eliminações, o Algoritmo Fundamental resolve o problema w-MAXSAT em tempo linear, isto é, $O(n2^{k_{MAX}})$.

■

Pode-se verificar que o número de variáveis que aparecem em g_i, $|S_i|$, varia de acordo com a ordem na qual as variáveis são eliminadas. Como conseqüência, a máxima eficiência do Algoritmo Fundamental será obtida pela seqüência de eliminação que minimiza o máximo da cardinalidade de S_i (visto que o número de possíveis condições na iteração i é exponencial no número de variáveis em g_i).

O problema agora está em encontrar a seqüência que determina o mínimo do máximo $|S_i|$ (para $i = 1, \ldots, n$), dentre todas as seqüências de eliminação possíveis. Para compreender melhor este problema, considere um grafo auxiliar definido pela instância a ser resolvida. Este grafo é denominado grafo de co-ocorrência.

Definição 2.1 Um grafo de co-ocorrência, de uma função não-linear 0-1, é definido como sendo $G_c = (V, A)$, *onde o conjunto de vértices* $V = \{x_1, x_2, \ldots, x_n\}$ *corresponde às variáveis da função 0-1 e o conjunto de arestas*

$$A = \left\{ (x_i, x_j) \mid \exists (N_k, \bar{N}_k) \in \Gamma \text{ e } i \in (N_k \cup \bar{N}_k) \text{ e } j \in (N_k \cup \bar{N}_k) \right\}$$ *está associado às cláusulas da função.*

Veja na figura 2.1 o grafo de co-ocorrência correspondente à equação 2.1.

$$f(x) = -3x_1x_2\bar{x}_4 + 7x_3\bar{x}_6 + 2x_1x_3 - 9x_4x_5x_6 \qquad \text{(Eq 2.1)}$$

O processo de eliminação de uma variável da função $f(x)$ que leva $f_1(x_1, \ldots, x_n)$ à $f_2(x_2, \ldots, x_n)$ reflete-se no grafo de co-ocorrência de forma mais complexa que a simples eliminação do vértice correspondente a x_1. Isto é, a função $f_2(x_2, \ldots, x_n)$ poderá conter termos reunindo variáveis que não apareciam em f_1 em um mesmo termo. Com isto, arestas que não estavam presentes em G_c^1 (denota-se por G_c^i o grafo de co-ocorrência de $f_i(x_i, \ldots, x_n)$) estarão em G_c^2. O conjunto destas arestas é denotado K_a^1 (K_a^i para a função f_i). As arestas que são adicionadas correspondem ao que é denominado de *fill-in* de uma eliminação (ver [RTL76]). Observe a figura 2.2.

Apresentam-se a seguir algumas definições de teoria dos grafos, dadas por Arnborg *et al* [ACP87]. Para conceitos gerais de teoria dos grafos, recomenda-se consultar Bondy

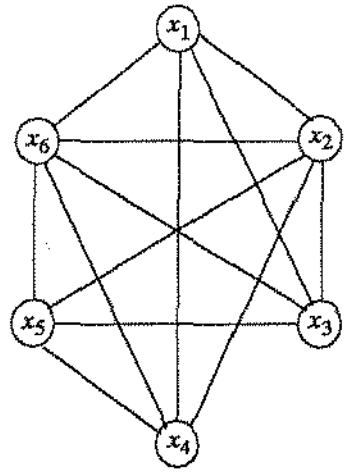


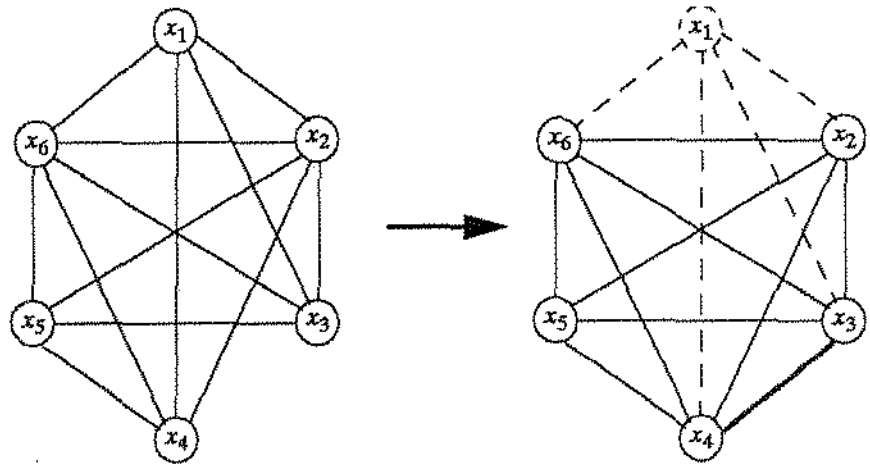
Figura 2.1 Grafo de co-ocorrência referente a equação 2.1.

e Murty [BM80]. Será considerado, nesse texto, que todos os grafos são simples e não-direcionados.

Definição 2.2 Um vértice v é simplicial num grafo G se os vizinhos de v induzem um subgrafo completo em G. Veja figura 2.3.

Um esquema de eliminação é uma ordenação π dos vértices de um grafo. Observando o processo de eliminação, verifica-se que no pior caso apareceram termos que combinam todos os pares de variáveis da variável eliminada. O grafo formado pela

Figura 2.2 Conjunto $K_a^1 = \{ (x_3, x_4) \}$ formado pela eliminação do vértice x_1.



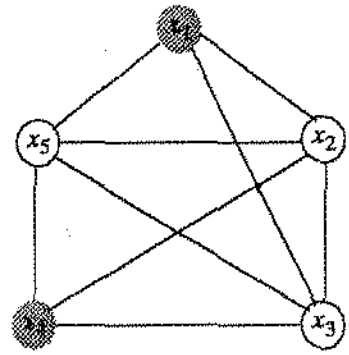


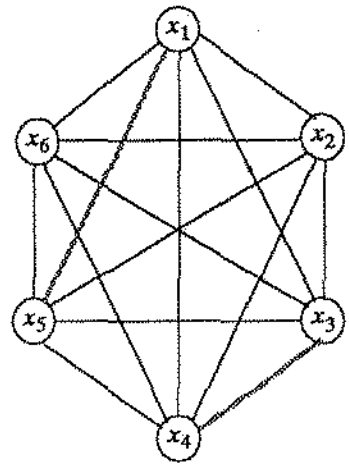
Figura 2.3 Os vértices sombreados são vértices simpliciais.

inserção das arestas de K_a^1, $G = (V, A \cup F^{\pi})$ onde F^{π} contém as arestas de K_a^1. Se a aresta $(u, v) \in F^{\pi}$ então existe um vértice w precedendo u e v em π tal que w é adjacente a u e v, mas $(u, v) \notin A$.

No grafo da figura 2.2, considere a eliminação dos vértices $\pi = \{x_6, x_2, x_3, x_4, x_1, x_5\}$, assim as arestas de F^{π} seriam $F^{\pi} = \{(x_3, x_4), (x_1, x_5)\}$. Veja o grafo da figura 2.4.

Definição 2.3 Um grafo tem um esquema de eliminação perfeito se existe π tal que F^{π} é vazio, isto é, existe uma ordem de eliminação dos vértices de G tal que cada vértice ao ser eliminado seja simplicial.

Figura 2.4 Arestas do conjunto $F^{\pi}=\{(x_1, x_5), (x_3, x_4)\}$.



Uma k-árvore é um grafo com $k+1$ vértices completo. Portanto, uma k-árvore com n vértices ($n > k+1$) pode ser construída a partir de uma k-árvore com $n-1$ vértices, adicionando-se um vértice adjacente somente aos vértices de qualquer clique de k-vértices do grafo. Uma k-árvore possui um k-esquema de eliminação perfeito, se os primeiros $n-k$ vértices são eliminados com grau k. Uma k-árvore parcial é qualquer grafo obtido pela remoção de arestas de uma k-árvore.

Assim, encontrar a melhor seqüência de eliminação para o Algoritmo Fundamental é equivalente encontrar a k-árvore parcial, com o menor k possível, no grafo de co-ocorrência. Esse problema foi formulado por Arnborg, Corneil e Proskurowski [ACP87] como o do reconhecimento de uma k-árvore parcial e definido como segue: dado um grafo G e um inteiro l, existe uma k-árvore parcial onde $k \leq l$? Arnborg *et al.* provaram que esse problema é *NP-completo* e propuseram também um algoritmo que reconhece uma k-árvore parcial para um dado k em tempo polinomial, mais precisamente em $O(n^{k+2})$. O Algoritmo Fundamental que utiliza o grafo de co-ocorrência para seleção sucessiva da próxima variável a eliminar é denominado Algoritmo Fundamental Revisitado (*BASIC ALGORITHM REVISITED*) [CHJ90].

2.2.2.2 *Exemplo de Execução do Algoritmo Fundamental*

Considere a maximização da seguinte função pseudo-booleana[1] f:

$$f(x) = -3x_1x_2\bar{x}_3 + 9x_4x_5x_6 + 4x_5x_6 - 5\bar{x}_1x_6 - 2x_1\bar{x}_3x_5 - 6\bar{x}_4x_6 - 2\bar{x}_2 - 5x_4 \qquad \text{(Eq 2.2)}$$

Inicialmente, constrói-se o grafo de co-ocorrência G_1, representado na figura 2.5.

Neste exemplo adota-se uma heurística de eliminação da variável que escolhe sempre o vértice de menor grau no grafo de co-ocorrência. Como critério de desempate utiliza-se a ordem inversa da ordem lexicográfica. Deste modo obtém-se x_6 como o primeiro vértice a ser eliminado. Lembrando-se que $\bar{x}_i = 1 - x_i$ (para todo i do conjunto de literais). Assim, f é reescrita colocando-se x_1 em evidência:

$$f_1(x) = x_6 \cdot g_1(x) + h_1(x); \text{ onde,}$$

$$g_1(x) = 9x_4x_5 + 4x_5 - 5\bar{x}_1 - 6\bar{x}_4$$

$$h_1(x) = -3x_1x_2\bar{x}_3 - 2x_1\bar{x}_3x_5 - 2\bar{x}_2 - 5x_4$$

Assim, para

$\phi_1(x_1, x_4, x_5) = (x_1x_4x_5) + (\bar{x}_1x_4x_5) + (x_1x_4\bar{x}_5)$, tem-se g_1 positivo.

1. Função pseudo-booleana é uma função cujo o conjunto domínio é formado pelo conjunto booleano {0, 1} e o conjunto imagem é o conjunto dos números reais.

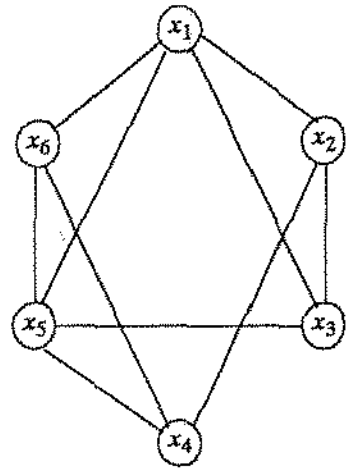


Figura 2.5 Grafo de co-ocorrência de f.

Substituindo x_6 por $\phi_1(x)$ em $f_1(x)$ tem-se,

$$f_2(x) = -3x_1x_2\bar{x}_3 + 13x_1x_4x_5 + 8\bar{x}_1x_4x_5 - 2x_1\bar{x}_3x_5 - 2\bar{x}_2 - 5x_4$$

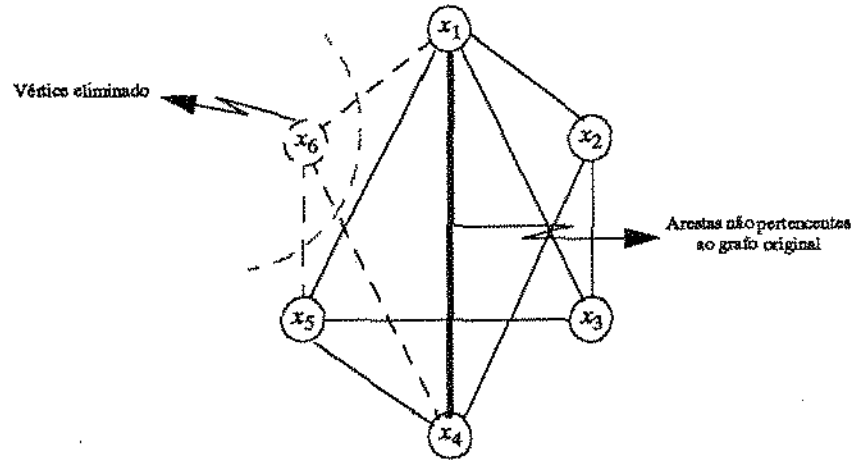
Observe que no novo grafo de co-ocorrência, figura 2.6, a aresta (x_2, x_6) deve ser incluída, pois existe termos de f_2 que contém as variáveis x_2 e x_6.

De acordo com o processo de eliminação, x_5 será eliminado, escrevendo-se f_2 como,

$$f_2(x_1, x_2, x_3, x_4) = x_5 \cdot g_2(x_1, x_2, x_3, x_4) + h_2(x_1, x_2, x_3, x_4); \text{ onde,}$$

$$g_2(x_1, x_2, x_3, x_4) = 13x_1x_4 + 8\bar{x}_1x_4 - 2x_1\bar{x}_3 \text{ e}$$

Figura 2.6 Grafo de co-ocorrência corresponde a f_2, após a eliminação de x_6.



$$h_2(x_1, x_2, x_3, x_4) = -3x_1x_2\bar{x}_3 - 2\bar{x}_2 - 5x_4$$

Observe que $g_2 > 0$, se

$$\phi_2(x_1, x_2, x_3, x_4) = (x_1x_3x_4) + (x_1\bar{x}_3x_4) + (\bar{x}_1x_3x_6) + (\bar{x}_1\bar{x}_3x_4)$$

Logo x_5 será substituído por ϕ_2 na função $f_2(x)$, obtendo-se

$$f_3(x) = -3x_1x_2\bar{x}_3 + 13x_1x_3x_4 + 11x_1\bar{x}_3x_4 + 21\bar{x}_1x_3x_4 + \\ +21\bar{x}_1\bar{x}_3x_4 - 2\bar{x}_2 - 5x_4$$

No próximo passo x_4 será eliminado e f_3 é reescrita como

$$f_3(x_1, x_2, x_3, x_4) = x_4 \cdot g_3(x_1, x_2, x_3) + h_3(x_1, x_2, x_3); \text{ onde}$$

$$g_3(x_1, x_2, x_3) = 13x_1x_3 + 11x_1\bar{x}_3 + 21\bar{x}_1x_3 + 21\bar{x}_1\bar{x}_3 - 5 \text{ e}$$

$$h_3(x_1, x_2, x_3) = -3x_1x_2\bar{x}_3 - 2\bar{x}_2$$

Como g_3 é sempre positivo independentemente de x_1, x_2 e x_3, tem-se $\phi_3 = 1$. Portanto, x_4 será substituindo por ϕ_3 em f_3, resultando em,

$$f_4(x_1, x_2, x_3) = -3x_1x_2\bar{x}_3 + 13x_1x_3 + 11x_1\bar{x}_3 + 21\bar{x}_1x_3 + 21\bar{x}_1\bar{x}_3 - 2\bar{x}_2 - 5$$

Agora x_3 será eliminado:

$$f_4(x_1, x_2, x_3) = x_3 \cdot g_4(x_1, x_2) + h_4(x_1, x_2); \text{ onde}$$

$$g_4(x_1, x_2) = 9x_5x_6 + 6x_6 - 5 \text{ e}$$

$$h_4(x_1, x_2) = 5x_5x_6 + 5\bar{x}_5x_6 + 4x_5x_6 - 11x_6$$

Para g_4 assumir somente valores positivos, deve-se ter

$$\phi_4(x_5, x_6) = (x_1x_2) + (x_1\bar{x}_2)$$

Assim, substituindo x_3 por ϕ_4 em f_4, obtém-se

$$f_5(x_1, x_2) = 13x_1x_2 + 13x_1\bar{x}_2 - 2\bar{x}_2$$

Agora x_5 é eliminado em f_5, obtendo-se

$$f_5(x_1, x_2) = x_2 \cdot g_5(x_1) + h_5(x_1); \text{ onde}$$

$$g_5(x_1) = 2 \text{ e}$$

$$h_5(x_6) = 13x_1 - 7$$

Note que g_5 é sempre positivo, logo $\phi_5 = x_2 = 1$.

Assim, substituindo x_2 por ϕ_5 em f_5 tem-se,

$$f_6(x_1) = 13x_1 - 5$$

O máximo valor para f_6 é obtido quando $x_1 = 1$.

Portanto,

$$x_1^* = 1$$
$$\Downarrow$$
$$x_2^* = \phi_5(x_1^*) = 1$$
$$\Downarrow$$
$$x_3^* = \phi_4(x_1^*, x_2^*) = 1$$
$$\Downarrow$$
$$x_4^* = \phi_3(x_1^*, x_2^*, x_3^*) = 1$$
$$\Downarrow$$
$$x_5^* = \phi_2(x_1^*, x_2^*, x_3^*, x_4^*) = 1$$
$$\Downarrow$$
$$x_6^* = \phi_2(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*) = 1$$
$$\Downarrow$$
$$f^* = 8.$$

2.3 Métodos Enumerativos

2.3.1 Introdução

Os algoritmos enumerativos são técnicas de otimização que se utilizam de pesquisa arborescente (*branch-and-bound*) para que possam encontrar soluções exatas, isto é, soluções ótimas [Taha75, NW88, GN72].

Um procedimento enumerativo foi proposto por Egon Balas em 1963 [Bala63]. Em 1966, Lawler e Bell [LB66] propuseram um algoritmo de enumeração lexicográfica para programas não-lineares 0-1 com restrições. Pouco tempo depois, tal método foi estendido por Mao e Wallingford [MW68, MW69] para o caso geral. Nos anos 60 e 70 vários pesquisadores propuseram vários algoritmos *branch-and-bound* para programação não-linear 0-1 com ou sem restrições.

É necessário definir nesses algoritmos:

- limites superiores e inferiores para o valor da solução ótima;
- um critério de particionamento do espaço de soluções;
- uma estratégia de pesquisa na árvore de sub-soluções;

Um limite inferior é dado por qualquer solução para o problema. Um limite superior é uma expectativa do maior valor que pode ser alcançado por uma solução. Eles são importantes porque permitem determinar que uma dada sub-solução não é capaz de gerar uma solução melhor do que a dada pelo limite inferior. Deste modo, o limite superior é o maior responsável pela redução da árvore de busca. Como o problema w-MAXSAT é restrito a variáveis 0-1, o critério de particionamento do conjunto de soluções válidas é, naturalmente, definido pela fixação do valor de uma variável em 0 em uma sub-solução e em 1 no espaço complementar. Uma estratégia de pesquisa na árvore de sub-soluções é a busca em profundidade que tem como principal característica a economia de memória e a facilidade de implementação. O algoritmo termina quando todas as folhas da árvore de sub-soluções forem pesquisadas. Como o número de folhas é finito, o algoritmo sempre termina num tempo finito. A seguir é descrito um algoritmo de enumeração implícita para a resolução do problema w-MAXSAT.

2.3.2 O Algoritmo de Enumeração Implícita

O procedimento enumerativo está relacionado com uma árvore composta de nodos (nós) e ramos. Um nodo corresponde a uma atribuição de valores 0-1 a um subconjunto das variáveis do problema. Uma variável pode pertencer a três estados: fixado em 1, fixado em 0 ou livre (não fixada). Observe a figura 2.7.

O algoritmo é iniciado com uma ou mais variáveis sendo fixadas em 0 ou 1; e então gradualmente o número das variáveis fixadas aumenta, ou seja, a "solução" é construída. A estratégia de exploração considerada é uma busca em profundidade.

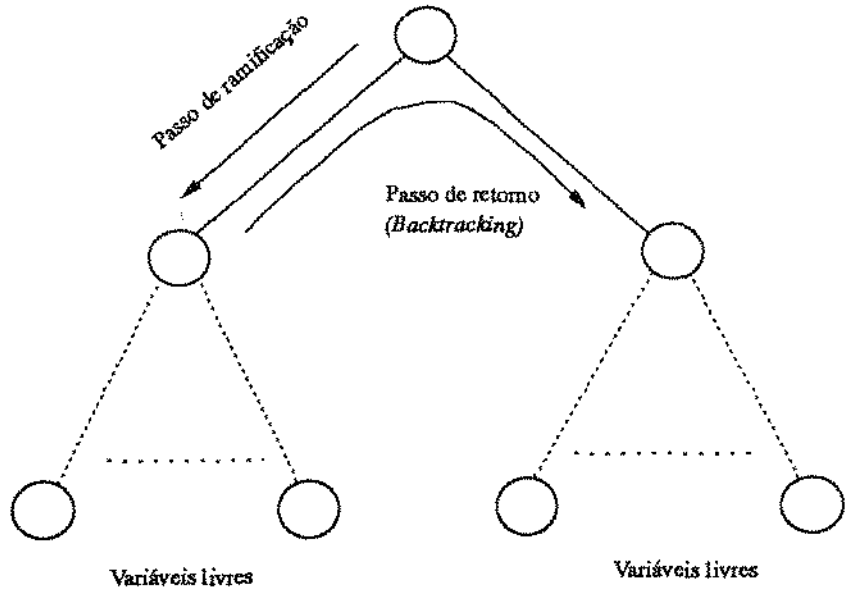


Figura 2.7 Árvore de busca do algoritmo de enumeração.

O limite superior utilizado é obtido pela identificação de uma função que é sempre superior a $f(x)$, no caso $\bar{f}(x)$. Um exemplo de uma tal função é a função obtida quando os termos que possuem coeficientes negativos são desprezados. Entende-se por **penalidades** associadas a uma variável livre as variações de $f(x)$ (limite superior) decorrente da imposição do valor 0 ou 1 a esta mesma variável. Denota-se por p_j^0 (resp. p_j^1) a variação de $f(x)$ decorrente da fixação de x_j à 0 (resp. à 1).

A seguir apresenta-se a descrição de um algoritmo de enumeração implícita de acordo com a proposta de Hansen, Jaumard e Mathon [HJM90].

Algoritmo Enumeração Implícita

1. Inicialização.

Determine uma solução inicial, "solução incumbente", x_{inc} e seu valor correspondente, "valor incumbente", z_{inc} utilizando um método heurístico.

2. Teste de otimalidade direta.

Calcule um limite superior $\bar{f}$ para f (função a ser maximizada) do subproblema corrente.

Se $\bar{f} \leq z_{inc}$ **Vá para** o passo 6 (*Backtracking*).

3. Teste de solução direta.

Se para uma solução parcial $\tilde{x}$, $f(\tilde{x}) > z_{inc}$

Então

$x_{inc} \leftarrow \tilde{x}$ e $z_{inc} \leftarrow \tilde{z}$

Senão processe o próximo passo.

4. Teste de otimalidade condicional.

Para cada variável x_j do subproblema corrente, calcule as penalidades p_j^0

e p_j^1 impostas por $\bar{f}$.

Se $\bar{f} - p_j^0 \leq z_{inc}$ **Então** fixa x_j à 1;

Se $\bar{f} - p_j^1 \leq z_{inc}$ **Então** fixa x_j à 0.

Se pelo menos uma variável foi fixada **Então Retorne** ao passo 2.

5. *Branching*.

Escolha uma variável x_j e uma fixação $x_j = 0$ ou $x_j = 1$ para ser explorada primeiro, de acordo com alguma regra de ramificação.

Retorne ao passo 2.

6. *Backtracking*.

Escolha um subproblema que não tenha sido explorado.

Se não existir tal subproblema

Então

PARE: z_{inc} é o valor ótimo e x_{inc} corresponde a solução.

Senão

Retorne ao passo 2, para explorar o ramo restante.

Usualmente os algoritmos de enumeração implícita utilizam-se de inúmeros testes de diversos tipos como apresentados acima. Uma vez que o caso geral desse algoritmo seja conhecido, pode-se considerar específico somente a formulação dos limites superiores (ou inferiores) impostos sobre a função a ser maximizada (ou minimizada).

2.4 *Métodos Aproximados*

2.4.1 *Introdução*

Uma abordagem alternativa para problemas *NP-completos* é a utilização de métodos aproximados. Sua necessidade provém do custo computacional requerido para encontrar uma solução ótima ou provar a otimalidade de tal solução, quando encontrada, o que é dispendioso para grandes instâncias de problemas. Assim, tais algoritmos são propostos de modo que seu tempo de execução seja pequeno, mesmo que nem sempre apresente uma solução de boa qualidade. Embora as soluções encontradas não sejam garantidamente ótimas, a qualidade destas soluções pode, em certos casos, ser estimada. Pode-se classificar estas heurísticas em duas classes: heurísticas de construção e heurísticas de melhoria. As heurísticas de construção iniciam com todas as variáveis livres determinando gradualmente seus valores. As

heurísticas de melhoria podem ser vistas também como uma busca local, onde parte-se de uma solução determinada por algum processo (aleatório ou por uma heurística de construção) e a partir desse ponto tenta-se melhorar a solução modificando-se as atribuições de valores às variáveis.

O problema característico desses algoritmos está relacionado com uma grande tendência de retornarem aos máximos/mínimos locais já obtidos, ou ainda de terminarem sua execução nos primeiros locais visitados, ficando distantes do ponto ótimo global do problema abordado.

A fim de solucionar tais problemas surgiram as meta-heurísticas, que tentam evitar o término do algoritmo nos primeiros máximos locais e prevenir a ocorrência de ciclos.

2.4.2 *Heurísticas de Construção*

2.4.2.1 *Heurísticas de Johnson*

Johnson [John74] apresenta duas heurísticas gulosas, bastante intuitivas, com complexidade $O(l \log l)$, onde l é o somatório das cardinalidades do conjunto de cláusulas. Estas heurísticas foram propostas para o problema de maximização não ponderado.

A primeira heurística procede escolhendo a cada iteração, o literal[1] que ocorre no maior número de cláusulas. Em seguida, atribui o valor 1 (*TRUE*) a este literal. A iteração se completa acrescentando o número de cláusulas satisfeitas ao número total de cláusulas satisfeitas e retirando-as do conjunto de cláusulas com variáveis livres. O algoritmo termina quando não houver mais variáveis livres nas cláusulas restantes. A seguir apresenta-se uma descrição da mesma.

As variáveis utilizadas na representação do algoritmo são:

TRUE	-	Conjunto das variáveis fixadas em 1 (*TRUE*)
LEFT	-	Conjunto das cláusulas que contém variáveis livres
LIT	-	Conjunto dos literais complementados ou não
z(*TRUE*)	-	valor da função objetivo para as fixações do conjunto *TRUE*

Para qualquer literal x^{α}, seja x^{α} com $\alpha \in \{0, 1\}$ igual a x se $\alpha = 1$ e a $\bar{x}$ se $\alpha = 0$.

Heurística John1

TRUE = ∅;
LEFT = *C*;
LIT = $X \cup \bar{X}$;
z(*TRUE*) = 0;
Enquanto *LEFT* = ∅ **faça**
  Seja x^{α} o literal em *LIT* que está contido no maior número de cláusulas de *LEFT* e

1. Os literais da variável x são: x e $\bar{x} = 1 - x$.

Seja U_T o conjunto de cláusulas em *LEFT* que contém x;
Faça: $z(TRUE) = z(TRUE) + |U_T|$;
$LEFT = LEFT - U_T$;
$TRUE = TRUE \cup \{x^{\alpha}\}$;
$LIT = LIT - \{x, \bar{x}\}$;
Fim-faça
Fim-enquanto
Retorne o valor de $z(TRUE)$;

A segunda heurística, proposta por Johnson, difere da anterior apenas pela atribuição de um peso igual a 2^{-c} (onde c = cardinalidade das cláusulas), a cada cláusula. Escolhe-se qualquer literal livre. Dois conjuntos são considerados inicialmente: o primeiro formado pelas cláusulas que contém o literal e o segundo com aquelas que contém o complemento do mesmo. Assim, o conjunto que contiver o maior peso atribuirá *TRUE* ao literal correspondente. O número de cláusulas que forem satisfeitas com esta atribuição será acrescentado ao total de cláusulas satisfeitas e as cláusulas serão retiradas do conjunto inicial. Em seguida, para o conjunto menor, dobra-se o peso de todas as cláusulas. O algoritmo termina quando não existirem mais cláusulas.

Apresenta-se a seguir a descrição da heurística, dada por Johnson.

As variáveis utilizadas na representação do algoritmo são:

TRUE	-	Conjunto das variáveis fixadas em 1 (*TRUE*)
LEFT	-	Conjunto das cláusulas que contém variáveis livres
LIT	-	Conjunto dos literais complementados ou não
$z(TRUE)$	-	Valor da função objetivo para as fixações do conjunto *TRUE*
$w(C_i)$	-	peso atribuído à cláusula C_i

Heurística John2

1. **Atribui-se** para cada cláusula $C_i \in C$ um peso $w(C_i) = 2^{|C_i|}$;
   $TRUE = \emptyset$;
   $LEFT = C$;
   $LIT = X \cup \bar{X}$;
   $z(TRUE) = 0$;
2. **Se** $LEFT = \emptyset$ **Então PARE** e **Retorne** $z(TRUE)$ como o ótimo aproximado;
3. Seja x qualquer literal que ocorra em *LIT* e em uma cláusula de *LEFT*;
   Seja U_T o conjunto de cláusulas pertencente a *LEFT* contendo x;
   Seja U_F o conjunto de cláusulas pertencente a *LEFT* contendo $\bar{x}$;
4. **Se** $\sum_{C_i \in U_T} w(C_i) = \sum_{C_i \in U_F} w(C_i)$
   Então $TRUE = TRUE \cup \{x\}$;
   $z(TRUE) = z(TRUE) + |U_T|$;
   $LEFT = LEFT - U_T$;
   Para cada $C_i \in U_F$ **faça** $w(C_i) = 2 \cdot w(C_i)$;

Senão $TRUE = TRUE \cup \{\bar{x}\}$;
$z(TRUE) = z(TRUE) + |U_F|$;
$LEFT = LEFT - U_F$;
Para cada $C_i \in U_T$ **faça** $w(C_i) = 2 \cdot w(C_i)$;
Fim-se;
5. $LIT = LIT - \{x, \bar{x}\}$;
Vá para 2.

A diferença básica entre as duas heurísticas é o fato da primeira não ter grande preocupação com cláusulas de menor cardinalidade, ficando assim muito distante do valor ótimo. Este fato não ocorre com John2 porque inicialmente tenta-se satisfazer as cláusulas com menos literais.

Os piores casos possíveis para a análise de cada heurística mencionada anteriormente são, respectivamente, para o caso não ponderado:

$$\text{John1: } z_1 \geq \frac{q}{q+1} z^* \text{ e John2: } z_2 \geq \frac{2^{q-1}}{2^q} z^*$$

onde, q denota o menor número de literais contidos em qualquer cláusula e z^* denota o maior número de cláusulas que podem ser satisfeitas simultaneamente, ou seja, o valor da solução ótima.

2.4.2.2 Heurística de Lieberherr

Lieberherr [Lieb82], propôs um algoritmo de aproximação baseado no número médio de cláusulas satisfeitas entre todas atribuições com exatamente α variáveis fixadas em 1 (*TRUE*). Esta heurística também foi proposta para o problema de maximização não ponderado.

Sejam $mean_\alpha(C)$ o número médio de cláusulas satisfeitas entre todas as atribuições com exatamente α variáveis fixadas em 1 e $maxmean(C) = \max_{0 \leq \alpha \leq n} mean_\alpha(C)$.
Essa heurística está dividida em dois procedimentos: o primeiro determina o valor do α de maior média (procedimento MAXMEAN), o que pode ser feito testando α de 0 a n e calculando a média através de uma fórmula obtida por análise combinatória; o segundo encontra uma atribuição para as variáveis de modo a garantir que esta tenha o número de cláusulas satisfeitas igual à média de α atribuições (procedimento MEAN). Segue a descrição dos procedimentos, conforme a proposta de Lieberherr.

Procedimento ***MAXMEAN***
1. **Encontre** α $(0 \leq \alpha \leq n)$ por uma busca linear, tal que $maxmean(C) = mean_\alpha(C)$.
2. **Aplique** o procedimento **MEAN** para C e α para encontrar uma atribuição satisfazendo pelo menos $maxmean(C)$ relações.

Procedimento ***MEAN***
1. $mean_{-1}(C) = 0$;
2. **Para todas** variáveis $x_i \in C$ **faça**

Se $mean_{\alpha}(C_{/x_i=1}) > mean_{\alpha}(C_{/x_i=0})$
Então $x_i \leftarrow 1$;
$\alpha \leftarrow \alpha$-1;
$C \leftarrow C_{/x_i=1}$;
Senão $x_i \leftarrow 0$;
$C \leftarrow C_{/x_i=0}$;
Fim-se
Fim-para

2.4.3 *Meta-heurísticas*

2.4.3.1 *Introdução*

Meta-heurísticas são heurísticas que atuam em um nível mais alto procurando guiar heurísticas subordinadas para escapar de ótimos locais ou encontrar ótimos locais alternativos. Como heurísticas subordinadas pode-se citar heurísticas de construção e de busca local. Dentre os principais paradigmas nos quais são baseados as meta-heurísticas pode-se citar *simulated annealing*, busca tabu, algoritmos genéticos, GRASP (*greed randomized adaptive search procedures*) e times assíncronos. Esta seção concentra-se no estudo dos dois primeiros paradigmas.

2.4.3.2 *Simulated Annealing*

Esse algoritmo baseia-se no princípio físico que rege o comportamento de partículas (inicialmente em equilíbrio numa temperatura T) ao serem resfriadas. Se o resfriamento é feito rapidamente, as partículas cristalizam-se com uma energia elevada; caso contrário (lento e gradativo), o sistema pode ser levado a níveis bem mais baixos de energia.

O algoritmo proposto por Metropolis [MR+53], procura simular tal comportamento. Considera-se o grupo de átomos e seus possíveis deslocamentos de uma só vez. Seu procedimento consiste na escolha aleatória de uma nova configuração de partículas, a qual será aceita com uma probabilidade $P = \text{MIN}\left(e^{\frac{-\Delta E}{KT}}, 1\right)$ (onde ΔE é a variação da energia do grupo de átomos e K a constante de Boltzman), reduzindo a temperatura T e repetindo a operação. Após um determinado número de iterações, obtém-se P muito próximo de zero para $\Delta E > 0$; com isto rejeita-se todas as novas configurações de átomos que aumentam a energia do sistema, que cristaliza-se pela não existência do deslocamento dos átomos de uma só vez de tal forma que reduza a energia.

Kirkpatrick [KGV83], adaptou esse algoritmo para a resolução de um problema de otimização combinatória. Considera-se as configurações possíveis do grupo de átomos, os seus possíveis deslocamentos de uma única vez e a energia do sistema como, respectivamente, o conjunto de soluções viáveis, a vizinhança de uma solução viável e a função objetivo do problema combinatório.

Esse algoritmo pode ser visto como uma generalização de uma Busca Local [PS82], onde ao invés de escolher a "melhor", é escolhida aleatoriamente uma solução vizinha. Além disso, para "escapar" das soluções locais, outras "piores" são aceitas com uma probabilidade, inicialmente significativa, mas se aproxima de zero a cada iteração.

Para o problema w-MAXSAT, com respeito aos valores de $X = \{x_1, x_2, \ldots, x_n\} \in \{0, 1\}$, o conjunto de soluções viáveis E é o próprio $\{0, 1\}^n$.

A definição de uma vizinhança V de uma solução válida em w-MAXSAT, deve satisfazer a condição: a partir de qualquer solução viável seja possível atingir qualquer solução de E. Isto pode ser realizado através da troca sucessiva da solução por outra viável pertencente à vizinhança V. Assim, as vizinhanças possíveis em w-MAXSAT são as trocas dos valores dos componentes da solução de 0 (*FALSE*) para 1 (*TRUE*) e vice-versa. Uma boa escolha da vizinhança pode ter caráter decisivo no desempenho da heurística SA.

Apresenta-se a seguir o algoritmo SA, adaptado ao problema w-MAXSAT, segundo a formulação proposta por Hansen e Jaumard [HJ90]. O procedimento consiste na definição do número de tentativas de aceitação de uma solução vizinha *(rep)*, seguida da redução da temperatura T. Se em nenhuma das tentativas uma solução vizinha for aceita, o algoritmo termina.

As variáveis utilizadas na representação do algoritmo são:

T	-	representa a "temperatura" numa dada iteração
rep	-	número de tentativas para aceitação de uma solução vizinha sem reduzir a temperatura
change	-	indica a aceitação de uma solução durante uma tentativa *rep*
a	-	fração restante da "temperatura" após *rep* tentativas, $0 < a < 1$
S	-	solução corrente durante a execução do algoritmo
S'	-	solução vizinha
$V(S)$	-	conjunto de soluções vizinhas a S
x	-	variável aleatória de distribuição uniforme
P	-	probabilidade de aceitação de uma solução ruim

Algoritmo SA

1. $T \leftarrow$ valor inicial grande;
   Determine uma solução factível S e calcule $z(S)$;
   change $\leftarrow$ *true*;
2. **Enquanto** *change* **faça**
   change $\leftarrow$ *false*;
   Repita para *rep*
   $S' \leftarrow V(S)$;

```
        calcule δ, valor da troca correspondente na função objetivo;
        Se δ ≥ 0 Então Vá para Aceita;
        P(δ) ← exp(δ/T);
        Gere aleatoriamente uma variável x uniformemente distribuída em
            [0,1];
        Se x < P(δ) Então Vá para Aceita
        Senão Vá para Saída;
        Aceita:  S ← S';
                 z(S) ← z(S) + δ;
                 Se δ = 0 Então change ← true;
        Saída;
    Fim-repita;
    T ← T.a;
Fim-enquanto;
```

Apesar de possuírem um papel decisivo no tempo de execução e na qualidade da solução, os parâmetros T e *rep* são determinados através de várias tentativas de execução do algoritmo e da análise dos resultados obtidos.

2.4.3.3 *Busca Tabu (BS)*

Busca Tabu (do inglês *Tabu Search*), é um procedimento que explora os limites entre as técnicas de programação inteira e de inteligência artificial guiando heurísticas de subida (algoritmo do gradiente) para continuarem a exploração do espaço de soluções sem confundir-se pela ausência de movimento que melhore a solução corrente e sem retornar a um ótimo local previamente visitado.

O algoritmo parte de uma solução inicial, escolhendo a direção de maior variação da função objetivo por unidade (direção de maior subida) e obtendo a solução ótima em tal direção. Quando esta última for a de menor descida, o algoritmo, ao invés de terminar como acontece no algoritmo do gradiente, segue por esta direção. No entanto, seguindo pela direção de menor descida, o algoritmo impede a reutilização desta direção por um determinado número de iterações. Tal proibição é feita por uma lista, conhecida como Lista Tabu. Isto é necessário devido à possibilidade destas direções serem as de maior subida a partir da solução seguinte, criando um ciclo. Essa lista não elimina totalmente a possibilidade de ciclagem do algoritmo, ela garante apenas um número determinado de iterações sem a criação de um ciclo. Existem algumas propostas para detectar e parar a ciclagem, como por exemplo: definir um número fixo de iterações nas quais a melhor solução encontrada não foi alterada; ou ainda variar o tamanho da Lista Tabu. Pode-se observar que a heurística SA não explora necessariamente todas as direções das trocas locais. A Busca Tabu pode ser dita mais objetiva (ou gulosa) que SA.

Tal heurística, assim como a SA, exige a definição de vizinhança para cada solução, do tamanho da Lista Tabu que contém as direções proibidas. Adaptado ao problema w-MAXSAT, o algoritmo é descrito seguindo a formulação também proposta por Hansen e Jaumard [HJ90].

Descrição das variáveis utilizadas na apresentação da heurística TS:

rep	-	número de iterações para verificar a melhora ou não de uma solução conhecida
change	-	indica se uma solução foi melhorada
d_j	-	número de iterações em que a direção d_j está proibida
S	-	solução corrente
$V(S)$	-	conjunto de soluções vizinhas a S
S_j	-	solução vizinha à solução corrente S atingida ao se tomar a direção j
S^*	-	melhor solução conhecida
z^*	-	valor da função objetivo em S^*
p	-	indica o número de iterações em que uma direção ficará proibida

Algoritmo TS

1. Selecione uma solução factível S e determine $z(S)$;
   Para cada $j \in J$ **faça** $d_j = 0$;
   change ← *true*;
   $S^* \leftarrow S$;
   $z^* \leftarrow z(S)$;
2. **Enquanto** *change* **faça**
   change ← *false*
   Repita *rep*
   $S_j \leftarrow S$, modificando a j-ésima direção;
   $\delta_j \leftarrow z(S_j) - z(S)$ para $j \in J$ com $d_j = 0$;
   Seja $\delta_k = max\ \{\delta_j | j \in J, d_j = 0\}$;
   $S \leftarrow S_k$;
   Se $\delta_k \leq 0$ **Então** $d_k \leftarrow p$;
   Se $z(S_k) > z^*$
   Então $z^* \leftarrow z(S_k)$;
   $S^* \leftarrow S_k$;
   change ← *true*;
   $d_j \leftarrow d_j - 1$ para todo $j \in J$ tal que $d_j > 0$;
   Fim-repita;
   Fim-enquanto;

Os parâmetros d_j, *rep* e p desta heurística também influenciam significativamente em seu desempenho. Como no algoritmo SA, são determinados empiricamente baseando-se em resultados obtidos previamente experimentais.

2.5 Sumário

Este capítulo abordou algumas técnicas utilizadas na resolução do problema w-MAXSAT encontradas na literatura. Essas técnicas podem ser classificadas em: Métodos Exatos e Métodos Aproximados. Primeiramente apresentou-se a classe dos métodos exatos representada pelos algoritmos algébricos e enumerativos. Os métodos exatos são conhecidos por determinarem a solução ótima para o problema a ser resolvido. Finalmente, foram descritos alguns algoritmos representantes da classe dos métodos aproximados. Esses algoritmos são conhecidos por serem de fácil implementação e pela rápida determinação de uma solução, tendo como dificuldade a qualidade das soluções determinadas.

Capítulo 3

Combinação Algébrico-Enumerativa

O Algoritmo Fundamental resolve uma certa classe de instâncias do problema w-MAXSAT com complexidade linear. Neste capítulo investiga-se a aplicação desse algoritmo, unido ao de enumeração implícita, para a resolução de instâncias gerais. Um dos resultados obtidos é que este algoritmo híbrido permite resolver uma classe de instâncias mais ampla do w-MAXSAT em tempo polinomial.

A descrição deste algoritmo algébrico-enumerativo (ALGEN), baseado na combinação do Algoritmo Fundamental com um Algoritmo de Enumeração Implícita, é o tema central deste capítulo. Na primeira parte descreve-se o ALGEN e analisa-se sua complexidade. Em seguida são apresentadas algumas generalizações dos cálculos de limites superiores propostos por Hansen, os quais são utilizados no desenvolvimento do algoritmo enumerativo. Finalmente, outras estratégias de combinação para o ALGEN são propostas e analisadas.

3.1 Introdução

O presente capítulo consiste na proposta de combinar algoritmos algébricos com enumerativos na resolução do w-MAXSAT. Esta proposta tem sua motivação nos casos polinomiais de w-MAXSAT que são resolvidos pelos algoritmos algébricos. Nestes casos, esses algoritmos exploram a existência de estruturas de k-árvores parciais no grafo de co-ocorrência definido pela instância a ser resolvida. Como apresentado na seção 2.2.2.1 a complexidade do Algoritmo Fundamental é dada por $O(n2^k)$. Portanto, sua eficiência se torna reduzida com o aumento de k, onde k é uma característica da instância a ser resolvida.

Combinações similares foram propostas anteriormente. Em 1973, Bradley e Wahi [BW73] fizeram uso da combinação desses métodos na resolução de problemas de programação inteira com n variáveis e n ou $n+1$ restrições de desigualdades. Tal combinação tinha como proposta transformar esse problema em um algebricamente equivalente, denominado Canônico Hermite e, a partir deste, executar a eliminação de Fourier-Motzkin[1]. Essa operação algébrica transforma o problema de tal forma que utiliza um esquema de enumeração implícita para determinar a solução ótima. O algoritmo foi implementado em FORTRAN IV no IBM-7094. O código não possui qualquer estrutura especial ou algum conhecimento a priori de soluções factíveis. O algoritmo obteve uma boa performance, na época, para uma variedade de problemas. Os autores consideraram 28 problemas testes de várias fontes tais como teoria dos grafos, problema da mochila (alocação), do inglês *knapsack*, problema de recobrimento, entre outros. Dentre esses testes, 23 determinaram uma solução ótima para o programa inteiro original, os demais determinaram apenas um limite superior. Não foi realizada nenhuma comparação desse algoritmo com os vários apresentados na literatura.

Aplicar qualquer versão do Algoritmo Fundamental para maximizar um problema geral de w-MAXSAT pode ser frustrante. A eliminação de uma simples variável x_i tendo em vista que g_i depende de muitas outras variáveis, pode dispender muito tempo e além disso requerer um grande consumo de memória. Algumas vezes isto é esperado pois, a determinação de ϕ_i requer que sejam encontradas todas as condições nas quais as variáveis booleanas de g_i que o tornem positivo (como visto no capítulo anterior). Obviamente isto requer mais esforço do que uma simples busca, que se utiliza de um esquema *branch-and-bound* simplificado, para a maximização da função.

No entanto, a habilidade potencial desse algoritmo não pode ser descartada na determinação eficiente do valor máximo do w-MAXSAT. De fato, investigações mais profunda sugerem a remoção de um subconjunto N_f de N (conjunto das variáveis pertencentes à função original) de forma a permitir a obtenção de uma k-árvore parcial

1. É uma técnica utilizada na resolução de um sistema de desigualdades lineares em variáveis contínuas.

associada ao grafo de co-ocorrência, onde k possui um valor adequado. Assim, ao fixar as variáveis do subconjunto N_f, o problema resultante pode ser resolvido em tempo linear pelo Algoritmo Fundamental. Em outras palavras, se uma enumeração for executada nas variáveis de N_f, o Algoritmo Fundamental pode resolver eficientemente o problema resultante a partir de qualquer folha da árvore de busca.

A necessidade de se obter um valor de k adequado pode ser observada considerando que no passo de eliminação do Algoritmo Fundamental, uma variável (x_1) ao ser eliminada está associada a um vértice cujo grau é no máximo k no grafo de co-ocorrência da função $f(x)$. Uma conseqüência imediata deste fato é que podem ser obtidos até $2 \cdot 3^k$ termos na função $\phi_1(x_2, \ldots, x_n)$. Desta forma, termos contendo qualquer subconjunto das k variáveis das quais a função $g_1(x)$ depende podem estar presentes em $\phi_1(x_2, \ldots, x_n)$, resultando no número de termos dado pela equação 3.1,

$$2^0 \cdot \binom{k}{0} + 2^1 \cdot \binom{k}{1} + 2^2 \cdot \binom{k}{2} + \ldots + 2^k \cdot \binom{k}{k} = \sum_{i=0}^{k} 2^i \cdot \binom{k}{i} = 3^k \qquad \text{(Eq 3.1)}$$

Uma vez que o vértice representa a variável e seu complemento, deve-se multiplicar o resultado da equação 3.1 por 2, resultando em $2 \cdot 3^k$ termos. Analogamente, cada variável x_i ocorre no máximo em $2 \cdot 3^k$ termos não contendo $x_1, \ldots, x_{i-1}$. Portanto, o número máximo de termos de f_i ($i = 1, \ldots, n$) é $2 \cdot 3^k \cdot n$, isto é, $O(n)$ termos. E mais, como $G_c(f)$ não tem clique com $k+2$ vértices, pois os vértices possuem grau no máximo k, todos os termos de f_i tem no máximo $k+1$ variáveis. Devido a esses motivos, se k for muito grande ou variável, o número de termos será exponencial nesse valor.

Os algoritmos algébricos procuram a cada iteração reescrever o problema em um número inferior de variáveis, eliminando uma por vez. A complexidade dessa operação é definida pelo número de variáveis que se relacionam diretamente com aquela a ser eliminada. Tal número corresponde ao grau do vértice associado à variável no grafo de co-ocorrência no instante de sua eliminação, o que justifica a performance inferior para grandes valores de k. Os algoritmos enumerativos fixam, a cada nível de sua árvore de busca, o valor de uma variável 0-1 de forma que esta fique igual a zero em um ramo e a um no outro. Isto provoca um impacto que em geral é proporcional ao grau do vértice associado à variável fixada no grafo de co-ocorrência. Observe que, ao contrário do que ocorre para os algoritmos algébricos, quanto maior o impacto, melhor a performance dos algoritmos enumerativos. Pois, é razoável considerar que o grau do vértice represente uma estimativa do impacto da fixação desta variável no tamanho da árvore de enumeração. Entretanto, outras estratégias que consideram a imprecisão deste ponto serão discutidas no decorrer deste capítulo.

As observações acima sugerem um comportamento complementar das abordagens aqui discutidas e propõem a seguinte questão: **"Será que existe a possibilidade de se**

aproveitar parcialmente a eficiência relativa dos algoritmos algébricos dentro de uma mesma instância?". Experiências realizadas indicam que a resposta é afirmativa. Um novo problema é definir a melhor estratégia para a combinação destes métodos.

O capítulo está organizado da seguinte forma: a próxima seção apresenta o algoritmo híbrido ALGEN e suas características; a seção 3.3 descreve sua complexidade. Na seção 3.4 é descrita uma relação entre o valor de k e o subconjunto N_f. A seção 3.5, por sua vez, apresenta a generalização do cálculo dos limites superiores proposto por Hansen que são utilizados como parte do algoritmo enumerativo. A seção 3.6 relaciona outras estratégias de combinação. Na seção 3.7 são apresentadas algumas estratégias de implementação para tornar o ALGEN mais eficiente.

3.2 *O Algoritmo ALGEN*

Como mencionado anteriormente, o algoritmo ALGEN é o produto da combinação de um algoritmo algébrico com um algoritmo de enumeração implícita. Sua proposta para resolver o problema w-MAXSAT é de encontrar inicialmente um subconjunto $N_f \subseteq N$, onde N é o conjunto de variáveis pertencentes à instância do problema, de tal forma que o subgrafo induzido por $N - N_f$ no grafo de co-ocorrência seja uma k-árvore parcial associada ao grafo de co-ocorrência para um dado valor de k. Em seguida, para o conjunto N_f, aplica-se o algoritmo de enumeração implícita descrito da maneira apresentada na seção 2.3. No instante em que todas as variáveis de N_f estiverem fixadas, isto é ao atingir uma folha, executa-se o Algoritmo Fundamental no conjunto $N - N_f$ para encontrar uma solução.

O algoritmo ALGEN pode ser descrito da seguinte forma: determina-se um conjunto $N_f \subseteq N$, de tal forma que o subgrafo induzido por $N - N_f$ seja uma k-árvore parcial. Em seguida, aplica-se o algoritmo de enumeração nas variáveis do conjunto N_f. Quando todas variáveis de N_f são fixadas, o Algoritmo Fundamental é executado para atribuir valores as variáveis restantes. O ALGEN termina quando não há mais enumeração a fazer.

Apresenta-se a seguir o esquema do algoritmo ALGEN.

Algoritmo ALGEN

Passo 1 - Defina uma estratégia de combinação.

Escolha um valor para k.

Encontre N_f tal que o subgrafo induzido de G_c (grafo de co-ocorrência associado a $f(x)$) em $N - N_f$ seja uma k-árvore.

Armazene um esquema de eliminação δ provando que este subgrafo é uma k-árvore parcial.

Passo 2 - Inicie a execução do algoritmo de enumeração.

Seja x_{opt} uma solução determinada por uma heurística e z_{opt} o valor associado a x_{opt}.

Inicialize S o conjunto de subproblemas abertos com o elemento $S_1 = (N_1^1, N_1^0)$, onde N_1^1 e N_1^0 são o conjunto de variáveis fixadas em 0 e em 1, respectivamente, e estão ambos vazios.

Passo 3 - Teste de Otimalidade.

Escolha um subproblema S_i de S.

Se S está vazio **PARE**, e x_{opt} corrente é a solução ótima com valor z_{opt} ótimo.

Senão Calcule um limite superior $\bar{f}$ de $f(x)$.

Se $\bar{f} \leq z_{opt}$ **Remova** S_i de S

Repita esse passo.

Passo 4 - Teste de Fixação Condicional

Calcule para cada variável as penalidades e que são impostas a f quando x_j é fixada a 0 ou 1, respectivamente.

Se $\bar{f} - p_j^0 \leq z_{opt}$, **Adicione** x_j a N_i^1 e **Retorne** ao passo 3.

Se $\bar{f} - p_j^1 \leq z_{opt}$, **Adicione** x_j a N_i^0 e **Retorne** ao passo 3.

Passo 5 - *Branching*

Se não existir nenhuma variável $x_j \in N_f - (N_i^1, N_i^0)$, **Vá para** o passo 6.

Senão considere n_s o número de subproblemas de S.

Adicione em S o subproblema $S_{n_s+1} = (N_{n_s+1}^1, N_{n_s+1}^0)$ e

$S_{n_s+2} = (N_{n_s+2}^1, N_{n_s+2}^0)$, onde $N_{n_s+1}^0 = N_i^0$, $N_{n_s+1}^1 = N_i^1 \cup \{x_j\}$,

$N_{n_s+2}^0 = N_i^0 \cup \{x_j\}$ e $N_{n_s+2}^1 = N_i^1$,

Remova S_i de S e **Retorne** ao passo 3.

Passo 6 - Resolução Algébrica

Resolva S_i pelo Algoritmo Fundamental e sejam x_i e z_i, respectivamente, a solução ótima e o valor ótimo encontrado.

Se $z_i > z_{opt}$ **Atualize** $x_{opt} \leftarrow x_i$ e $z_{opt} \leftarrow z_i$

Retorne ao passo 3.

3.3 Complexidade do ALGEN

A complexidade do algoritmo ALGEN depende fundamentalmente da complexidade do Algoritmo Enumerativo e do Algoritmo Fundamental. Para o Algoritmo Fundamental o interesse está em conservar o limite superior $2^{k_{MAX}}$ (como apresentado no capítulo anterior), para que sua complexidade seja $O(n)$. Para tal, deve ser

observado o que ocorre com o subgrafo do grafo de co-ocorrência G_c formado após a fixação das variáveis do conjunto N_f. Dessa observação provém o seguinte teorema:

Teorema 3.1 Se G_c induzido por $N - \{v\}$ é uma k-árvore parcial, então o subgrafo obtido após fixar v também é.

Dem: Consideram-se dois possíveis casos.
Caso 1: O literal correspondente a v é fixado em *TRUE*

Neste caso, nenhuma aresta associada ao termo que contém esse literal é removida, pois esse termo permanece na função apenas sem o literal correspondente a v. A conclusão do teorema para este caso é trivial, isto porque a fixação procede como uma eliminação simples do vértice.
Caso 2: O literal é fixado em *FALSE*

Uma conseqüência direta desta é a eliminação dos termos que contém o literal de v. Assim, tem-se um novo subgrafo formado pela remoção das arestas, associadas aos termos, e dos vértices (quando esses não aparecem em nenhum outro termo), associados às variáveis contidas nesses termos, do grafo de co-ocorrência G_c. O grafo resultante será um subgrafo de G_c, e portanto será uma k-árvore parcial.

■

O teorema 3.1 garante a preservação do limite superior para os cálculos algébricos realizados pelo Algoritmo Fundamental, enquanto que o limite superior do algoritmo enumerativo será no pior caso $2^{|N_f|}$. Observe a figura 3.1.

As descrições anteriores sugerem o seguinte teorema:

Teorema 3.2 Para qualquer k fixo, o algoritmo ALGEN resolve qualquer instância de f(x) em $O(n^{c+1})$, onde $|N_f|$ é limitado superiormente por $c \cdot \log_2 n$.

Dem: Trivial.

Observe que o resultado é obtido se uma enumeração completa, com $c \cdot \log_2 n$ variáveis 0-1, for resolvida em $O(n^c)$ de tal modo que uma k-árvore parcial possa ser determinada em $O(n)$. Daí segue o resultado.

■

Esse teorema garante a complexidade polinomial do ALGEN na generalização das classes de instâncias de $f(x)$ que possuem uma k-árvore parcial associada ao grafo de co-ocorrência de $f(x)$. Estas novas classes correspondem às instâncias cujo grafo de co-ocorrência contém uma k-árvore parcial com pelo menos $|N - N_f|$ vértices.

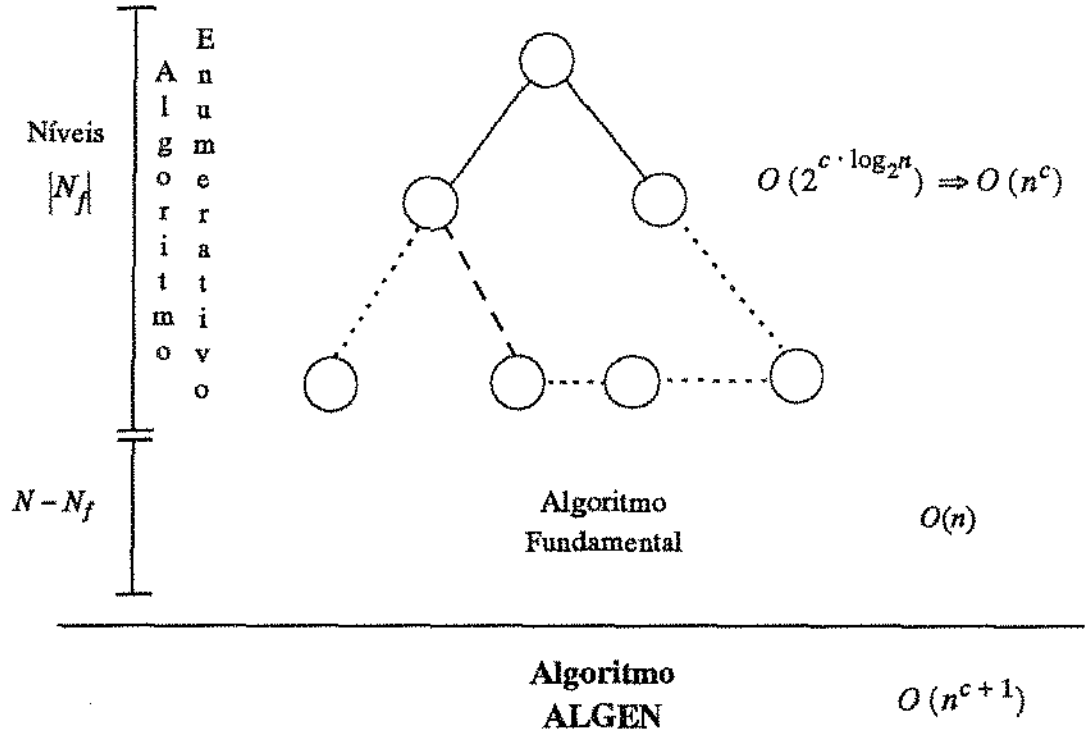


Figura 3.1 Representação da complexidade do ALGEN.

3.4 *Relação entre k e $|N_f|$*

Deseja-se escolher um subconjunto N_f de N, de tal forma que o algoritmo enumerativo explore tão poucas variáveis quanto possível na árvore de busca. E mais, que o subgrafo induzido de G_c, por $N - N_f$, seja uma k-árvore parcial. Uma vez que o número de nodos explorados na árvore de busca, depende fortemente da cardinalidade de N_f, pois quanto maior N_f maior o número de variáveis a serem enumeradas.

Entretanto, a cardinalidade de N_f depende do valor de k. Se k for muito pequeno tem-se um valor grande para N_f e vice-versa. É claro que se $k = n \Rightarrow N_f = 0$ e se $k = 0 \Rightarrow N_f = n$. Portanto, tem-se a intuição de que para qualquer instância I de P, existe k tal que o número de operações do ALGEN para a resolução de I é estritamente inferior ao número de operações tanto do Algoritmo Fundamental puro quanto do Algoritmo de Enumeração puro. Esta intuição pode ser reforçada pelos resultados computacionais apresentados na seção 4.7.

3.5 Determinação dos Limites e Penalidades

O cálculo de limites inferiores e superiores (do inglês *lower bounds* e *upper bounds*) são importantes tanto para os métodos aproximados quanto para os exatos. As penalidades são cálculos associados ao limite superior da função objetivo.

O cálculo de penalidades foi utilizado por Beale e Small [BS65] e Driebeeck [Drie66], na resolução de problemas de programação linear inteira mista (PLIM), isto é, o conjunto de variáveis é composto de variáveis contínuas e inteiras.

Sejam z_{max} e δ um limite superior conhecido para o problema w-MAXSAT e o decremento de z_{max} resultante da fixação de uma variável a um dado valor, respectivamente. Então o valor da melhor solução associada é $z_{max} - \delta$. Se z_{LI} for um limite inferior, conhecido para o problema w-MAXSAT, a atribuição binária resultando δ pode ser descartada caso $z_{max} - \delta \leq z_{LI}$. Observe que δ pode ser determinado com exatidão pela simples substituição do valor na variável. Entretanto tal processo pode ser extremamente caro do ponto de vista computacional. A proposta seria estimar um limite inferior $\underline{\delta}$ para δ, usando as informações pertencentes à função objetivo. Embora $z_{max} - \underline{\delta}$ não seja tão forte quanto $z_{max} - \delta$, é evidente que uma fixação obtendo $z_{max} - \underline{\delta} \leq z_{LI}$ também poderá ser descartada. Apesar dessa condição ser menos forte, isto pode ser tolerado uma vez que, como será visto a seguir, $\underline{\delta}$ pode ser calculado fácil e rapidamente.

O limite $\underline{\delta}$ define a chamada **penalidade**. As penalidades ajudam a eliminar alguns ramos da árvore de busca de um algoritmo de enumeração, isto é, elimina-se o conjunto de soluções sub-ótimas, uma vez que alguma solução seja conhecida. Uma outra utilidade para as penalidades está relacionada com o critério para selecionar as variáveis a serem enumeradas. Fortes penalidades resultam em fortes limites superiores para o conjunto de soluções, e isto ocasiona poucas ramificações na árvore de busca, poucos cálculos e portanto reduzindo o tamanho da árvore de busca. Várias formas para se calcular esses limites podem ser encontradas na literatura. Existem várias propostas para o cálculo de penalidades; a próxima sub-seção apresenta algumas delas.

3.5.1 Cálculo das penalidades

Uma penalidade é uma estimativa da variação de $f(x)$ quando uma de suas variáveis assume um dado valor. Para o problema em estudo (maximização) será um decremento que pode ser dado ao limite superior quando uma variável livre x_j for fixada em zero ou em um. Assim, considere a função de maximização representada pela equação 3.2.

$$Max \qquad f(x) = \sum_{k=1}^{m} \omega_{N_k \bar{N}_k} \prod_{j \in N_k} x_j \prod_{j \in \bar{N}_k} \bar{x}_j$$

$$\Gamma \subseteq \left\{ (N_k, \bar{N}_k) \,\middle|\, (N_k \cup \bar{N}_k) \subseteq \{1, 2, \ldots, n\}, N_k \cap \bar{N}_k = \varnothing \right\} \qquad \text{(Eq 3.2)}$$

$$x_j = \{0, 1\} \qquad j = 1, 2, \ldots, n$$

$$\bar{x}_j = 1 - x_j$$

$$|\Gamma| = m, \omega_{N_k \bar{N}_k} \neq 0$$

Um limite superior bastante intuitivo para $f(x)$, é considerar a soma de todos coeficientes positivos. Isto sugere a seguinte proposição.

Proposição 3.1 $\bar{z}_1 = \sum_{k=1}^{m} max(0, \omega_k)$ *é um limite superior para f(x).*

Dem: Trivial.

Considere um problema que possua poucos termos com coeficientes positivos na solução ótima. Pode-se observar que o limite superior dado por $\bar{z}_1$ estará muito distante do valor ótimo, conseqüentemente muitos nodos serão enumerados. Logo, $\bar{z}_1$ pode ser melhorado se forem considerados os seguintes aspectos:

1. Se o literal for fixada em zero, os coeficientes dos termos positivos que o contém, não precisam ser incluídos no limite e os termos lineares negativos que possuem seu complemento deverão ser acrescentados ao limite superior, pois poderão fazer parte da melhor solução.
2. Se o literal for fixada em um, os termos lineares negativos que o contém, deverão ser adicionados no limite superior e os termos com coeficientes positivos que possuem o seu complemento deverão ser retirados do limite superior.

Hansen [Hans79] propôs o uso de penalidades aditivas para determinar melhores limites. As penalidades são ditas **aditivas** quando a soma das penalidades correspondentes podem ser subtraídas do limite superior corrente (ao que estão associadas), conservando a propriedade de limite superior válido.

Proposição 3.2 As penalidades aditivas, mostradas abaixo, podem ser associadas a $\bar{z}_1$

$$p'^{0}_{j} = \sum_{\substack{k=1 \\ j \in N_k}}^{m} \alpha_{jk} max(0, \omega_k) - \min_{\bar{N}_r = \{j\}} (0, \omega_r) \; e$$

$$p'^{1}_{j} = \sum_{\substack{k=1 \\ j \in \bar{N}_k}}^{m} \alpha_{jk} max(0, \omega_k) - \min_{N_r = \{j\}} (0, \omega_r)$$

onde,

$$0 \le \alpha_{jk} \le 1 \qquad 1 \le k \le m, \forall j \in N_k \cup \bar{N}_k \quad e$$

$$\sum_{j \in N_k \cup \bar{N}_k} \alpha_{jk} = 1 \qquad 1 \le k \le m$$

Dem: Veja o apêndice A.

Corolário 3.2.1 $\bar{z}_2 = \bar{z}_1 - \sum_{j=1}^{n} min(p'^{0}_{j}, p'^{1}_{j})$ *é um limite superior para f(x).*

Em outras palavras, as penalidades da proposição 3.2 distribuem os coeficientes dos termos entre suas variáveis. Pode-se dizer que a variação de $\bar{z}_1$ para $\bar{z}_2$ é uma estimativa da degradação de $\bar{z}_1$ frente ao fato de que todas as variáveis deverão, em algum momento, assumir valores binários.

Os valores de α_{jk} ($j = 1, ..., k$), que fornecem o menor limite superior $\bar{z}_2$, podem ser encontrados resolvendo um problema de fluxo máximo (ver [Hans84, HJM89]). Uma vez que a resolução desse problema repetidas vezes consome muito tempo, α_{jk} pode ser melhor determinado escolhendo-se valores heurísticos, o que torna mais rápido o algoritmo. Uma discussão mais detalhada segue na seção 3.7.3.

Se existirem vários termos lineares onde todo termo k tem-se coeficiente $\omega_k < 0$, o limite $\bar{z}_2$ é apertado (*tied*). Caso contrário, poderá ser melhorado como segue:

Seja $q_j = p'^{0}_{j} - p'^{1}_{j} \qquad \forall j \in \{1, ..., n\}$.

Proposição 3.3 As penalidades abaixo podem ser associadas ao limite superior $\bar{z}_2$.

$$p^{\pi}{}_j^0 = r_j^0 + \sum_{\substack{(k,j') \\ j \in \bar{N}_k \\ \omega_k < 0 \\ j' \in \bar{N}_k}} min\,(-\omega_k, r_{j'}^1) + \sum_{\substack{(k,j') \\ j \in \bar{N}_k \\ \omega_k < 0 \\ j' \in N_k}} min\,(-\omega_k, r_{j'}^0)$$

$$p^{\pi}{}_j^1 = r_j^1 + \sum_{\substack{(k,j') \\ j \in N_k \\ \omega_k < 0 \\ j' \in \bar{N}_k}} min\,(-\omega_k, r_{j'}^1) + \sum_{\substack{(k,j') \\ j \in N_k \\ \omega_k < 0 \\ j' \in N_k}} min\,(-\omega_k, r_{j'}^0)$$

onde,

$$r_j^0 = max\,(0, q_j)$$
$$e$$
$$r_j^1 = -min\,(0, q_j)$$

para todo (k, j') são tais que $\omega_k < 0$ *e* $|N_k| = 2, |\bar{N}_k| = 2$.

Dem: Veja o apêndice A.

Uma variável livre, em algum momento, assumirá um valor zero ou um. Conseqüentemente, o limite superior $\bar{z}_2$ será obrigatoriamente decrescido de pelo menos o menor valor entre as penalidades de fixar esta variável. Como este argumento é válido para qualquer variável livre, obtém-se:

Corolário 3.3.1 $\bar{z}_3 = \bar{z}_2 - \max\limits_{j=1,\ldots,n} \min\,(p^{\pi}{}_j^0, p^{\pi}{}_j^1)$ *é um limite superior para f(x).*

Considerando ainda o limite superior $\bar{z}_2$ e utilizando-se uma distribuição da variação das penalidades das variáveis nos termos (com coeficientes negativos), pode-se obter um novo limite superior também válido.

Proposição 3.4 $\bar{z}_4$, *definido como segue, é um limite superior para f(x).*

$$\bar{z}_4 = \bar{z}_2 - \sum_{\substack{k=1 \\ \omega_k < 0 \\ |N_k \cup \bar{N}_k| \geq 2}}^{m} min\Big(-\omega_k, \min_{j \in N_k} (\beta_{jk} max\,(0, q_j)), \min_{j \in \bar{N}_k} (-\gamma_{jk} min\,(0, q_j))\Big)$$

Dem: Encontra-se no apêndice A.

3.6 Outras Estratégias de Combinação

No desenvolvimento do algoritmo ALGEN surgiram outras formas de se combinar o algoritmo enumerativo com o algébrico. Apresenta-se a seguir, duas novas formas de combinação. A primeira está relacionada com um pré-processamento das variáveis; a segunda está na verificação da existência de uma k-árvore parcial após algumas variáveis serem fixadas pelo algoritmo de enumeração.

3.6.1 ALGEN-1

A determinação de N_f tende a selecionar os vértices de maior grau do grafo de co-ocorrência G_c, isto é, variáveis que aparecem em um maior número de termos e tem influência significantiva no valor de $f(x)$, sugerindo "boas" variáveis para uma ramificação na árvore de busca. Entretanto, variáveis associadas aos coeficientes dos termos, os quais possuem valores extremos (muito grande ou muito pequeno), podem ter grande influência no valor da função objetivo e, ainda assim, não serem escolhidas para integrarem o conjunto N_f, pois podem possuir um grau pequeno em G_c.

Esta observação sugere que antes da determinação do conjunto N_f, pela heurística do passo 1, ocorra o seguinte procedimento:

(I) Realiza-se um pré-processamento de forma a se obter variáveis que possuem uma influência significativa para $f(x)$, atribuindo-as ao conjunto N_p.

(II) Utiliza-se uma heurística para determinar uma k-árvore parcial do conjunto de variáveis induzidas por $N - N_p$.

(III) Constrói-se um conjunto N_f com as variáveis que não pertençam ao subconjunto das variáveis que definam uma k-árvore e com as variáveis do conjunto N_p.

Essas modificações sugerem o seguinte algoritmo.

Algoritmo ALGEN-1

Passo 1 - Pré-processamento.
Defina o número de variáveis a serem retiradas inicialmente de N.
Segundo uma estratégia de pré-processamento retire-as de N e armazene-as em N_p.

Passo 2 - Defina uma estratégia de combinação.
Escolha um valor para k.
Encontre N_f, tal que o subgrafo induzido de G_c (grafo de co-ocorrência associado a $f(x)$) em $N - N_f - N_p$, seja uma k-árvore.
Armazene um esquema de eliminação δ provando que este subgrafo é uma k-árvore parcial.
Acrescente à N_f o conjunto N_p.

Passo 3 - Inicie a execução do algoritmo de enumeração.

Seja x_{opt} uma solução determinada por uma heurística e z_{opt} o valor associado a x_{opt}.

Inicialize S o conjunto de subproblemas abertos com o elemento $S_1 = (N_1^1, N_1^0)$, onde N_1^1 e N_1^0 são o conjunto de variáveis fixadas em 1 e em 0 respectivamente, e estão ambos vazios.

Passo 3 - Teste de Otimalidade.

Escolha um subproblema S_i de S.

Se S está vazio **PARE**, e x_{opt} corrente é a solução ótima com valor z_{opt} ótimo.

Senão Calcule um limite superior $\bar{f}$ de $f(x)$.

Se $\bar{f} \leq z_{opt}$ **Remova** S_i de S e

Repita esse passo.

Passo 4 - Teste de Fixação Condicional

Calcule para cada variável as penalidades e que são impostas a f quando x_j é fixada a 1 ou 0, respectivamente.

Se $\bar{f} - p_j^0 \leq z_{opt}$, **Adicione** x_j a N_1^0 e **Retorne** ao passo 3.

Se $\bar{f} - p_j^1 \leq z_{opt}$, **Adicione** x_j a N_1^1 e **Retorne** ao passo 3.

Passo 5 - *Branching*

Se não existir nenhuma variável $x_j \in N_f - (N_i^1, N_i^0)$, **Vá para** o passo 6.

Senão considere n_s o número de subproblemas de S.

Adicione em S o subproblema $S_{n_s+1} = (N_{n_s+1}^1, N_{n_s+1}^0)$ e

$S_{n_s+2} = (N_{n_s+2}^1, N_{n_s+2}^0)$, onde $N_{n_s+1}^0 = N_i^0$, $N_{n_s+1}^1 = N_i^1 \cup \{x_j\}$,

$N_{n_s+2}^0 = N_i^0 \cup \{x_j\}$ e $N_{n_s+2}^1 = N_i^1$,

Remova S_i de S

Retorne ao passo 3.

Passo 6 - Resolução Algébrica

Resolva S_i pelo Algoritmo Fundamental e sejam x_i e z_i, respectivamente a solução ótima e o valor ótimo encontrado.

Se $z_i > z_{opt}$, **Atualize** $x_{opt} \leftarrow x_i$ e $z_{opt} \leftarrow z_i$.

Retorne ao passo 3.

A realização do pré-processamento pode ocorrer de várias formas:

1. Cálculo das penalidades para cada variável.
   Este processo pode ser realizado utilizando algumas das formas de se calcular as penalidades apresentadas anteriormente, realizando a seguir uma ordenação crescente dos respectivos valores. Uma vez ordenadas as penalidades, decide-se por uma percentagem de variáveis que serão incluídas no conjunto N_p.
   Assim, no conjunto formado por $N - N_p$, realiza-se uma busca para determinar uma

k-árvore, sendo que as variáveis desse conjunto não pertencentes à mesma serão incluídas, juntamente com N_p, no conjunto N_f.

2. Análise das cardinalidades das variáveis.
   Sabe-se, para cada variável, o número de termos que possuem peso positivo e negativo. Neste caso, se uma variável só aparece em termos negativos pode-se fixá-la, antecipadamente, em zero e com isto retirar todos os termos que a contém de *f(x)*. Para o caso da variável só aparecer em termos positivos, ela também poderá ser fixada a priori, em um; neste caso os termos possuirão menos variáveis, podendo vir a se tornar termos constantes.
   A vantagem dessa proposta ocorre quando o problema não contém variáveis complementadas, isto é, $\bar{x}$. Nos problemas que contém estes tipos de variáveis deve-se tomar um pouco mais de cuidado. Neste caso, melhores resultados poderão ser conseguidos com o cálculo das penalidades.

As experiências computacionais preliminares demonstraram que a primeira forma para se realizar o pré-processamento é superior à segunda, pois o conjunto de variáveis dada pela primeira considera valores mais fortes que auxiliam na escolha da variável a ser fixada.

3.6.2 ALGEN-2

Uma outra estratégia para o ALGEN é considerar a verificação de *k*-árvore parcial não no início de sua execução, mas sim durante a execução do algoritmo de enumeração. Deixa-se esse processar com o conjunto de variáveis do problema original, e após uma percentagem de variáveis fixadas, inicia-se a verificação da existência de *k*-árvore. Essa verificação ocorre sempre a cada passo da fixação.

Neste algoritmo, inicialmente determina-se o número de variáveis a serem enumeradas, γ, sem verificar a existência de *k*-árvore. Este número pode ser um percentual (50% p.ex.) dos valores encontrados no pré-processamento dos algoritmos ALGEN e ALGEN-1. Em seguida, utiliza-se uma heurística para verificar se o grafo de co-ocorrência corrente é uma *k*-árvore parcial. Caso a resposta não seja afirmativa, continua-se descendo na árvore de busca. Caso contrário, aplica-se o Algoritmo Fundamental para determinar as atribuições restantes. A finalização desse algoritmo também está associada com a do enumerativo. Segue uma apresentação do mesmo.

Algoritmo ALGEN-2

Passo 1 - Inicie a execução do algoritmo de enumeração.
Seja x_{opt} uma solução determinada por uma heurística e z_{opt} o valor associado a x_{opt}.
Inicialize S o conjunto de subproblemas abertos com o elemento
$S_1 = (N_i^1, N_i^0)$, onde N_i^1 e N_i^0 são o conjunto de variáveis fixadas em 1 e em 0 respectivamente, e estão ambos vazios.
Passo 2 - Teste de Otimalidade.

Escolha um subproblema S_i de S.

Se S está vazio **PARE**, e x_{opt} corrente é a solução ótima com valor z_{opt} ótimo.
Senão calcule um limite superior $\bar{f}$ de $f(x)$.

Se $\bar{f} \leq z_{opt}$ **Remova** S_i de S

Repita esse passo.

Passo 3 - Teste de Fixação Condicional

Calcule para cada variável as penalidades e que são impostas a f quando x_j é fixada a 1 ou 0, respectivamente.

Se $\bar{f} - p_j^0 \leq z_{opt}$, **Adicione** x_j a N_1^0 e **Retorne** ao passo 3.

Se $\bar{f} - p_j^1 \leq z_{opt}$, **Adicione** x_j a N_1^1 e **Retorne** ao passo 3.

Passo 4 - Verificação de *k*-árvore

Se não enumerou-se γ variáveis, **siga para** o passo seguinte.
Senão Se a heurística encontra uma seqüência de eliminação que identifica uma no grafo de co-ocorrência, **Vá para** o passo 6.
Senão Vá para o passo 5.

Passo 5 - *Branching*

Considere n_s o número de subproblemas de S.

Adicione em S o subproblema $S_{n_s+1} = (N^1_{n_s+1}, N^0_{n_s+1})$ e

$S_{n_s+2} = (N^1_{n_s+2}, N^0_{n_s+2})$, onde $N^0_{n_s+1} = N^0_i$, $N^1_{n_s+1} = N^1_i \cup \{x_j\}$,

$N^0_{n_s+2} = N^0_i \cup \{x_j\}$ e $N^1_{n_s+2} = N^1_i$,

Remova S_i de S

Incremente N_s de duas unidades

Retorne ao passo 3.

Passo 6 - Resolução Algébrica

Resolva S_i pelo Algoritmo Fundamental e sejam x_i e z_i, respectivamente a solução ótima e o valor ótimo encontrado.

Se $z_i > z_{opt}$ **Atualize** $x_{opt} \leftarrow x_i$ e $z_{opt} \leftarrow z_i$.

Retorne ao passo 3.

O ALGEN-2 pode proporcionar uma estratégia que forneça o mínimo de variáveis para o algoritmo de enumeração, isto é, N_f será pequeno. Observe ainda que, o conjunto N_f não possuirá tamanho fixo, isto é, para um dado ramo ($x_i = 1$ por ex.) ele é menor (ou maior) que para o outro ($x_i = 0$).

3.6.3 Comparação das estratégias

Na estratégia inicial de combinação, ALGEN, verificou-se que pode ser realizada a determinação de um conjunto de variáveis, as quais teriam um grande impacto na função objetivo. A vantagem da estratégia ALGEN-1 sobre a inicial é justamente a busca pelas variáveis de maior impacto no tamanho da árvore de enumeração.

Na terceira estratégia, ALGEN-2, a vantagem sobre ALGEN e ALGEN-1 encontra-se na maior liberdade do algoritmo de enumeração na manipulação das variáveis, utilizando-se da possibilidade de se verificar a existência de uma *k*-árvore após um número pequeno de fixações. Essa liberdade tenta evitar bifurcações (*branching*) desnecessárias que certamente ocorrem nas duas outras estratégias. A sua desvantagem provêm do custo computacional dispendido na verificação da *k*-árvore. Este custo pode ser muito grande. Um dos motivos pode estar relacionado com a não exploração de heurísticas eficientes para detecção de uma *k*-árvore parcial.

3.7 Técnicas Computacionais

A redução do tempo de processamento é um dos interesses principais no projeto de um algoritmo. Nesta seção são apresentados quatro pontos relacionados com esse interesse. Primeiro, representação interna da função objetivo. Segundo, a ramificação na árvore de busca. Terceiro, eficiência no cálculo das penalidades. E finalmente, o agrupamento dos termos que envolvem as mesmas variáveis.

3.7.1 Representação interna da função objetivo

São utilizadas duas classes de acesso à função objetivo: (*i*) dado o índice de um termo, lista-se todos literais deste e o coeficiente correspondente; (*ii*) dada uma variável x_k, lista-se todos os termos contendo x_k ou $\bar{x}_k$.

A primeira representação de *f(x)* consiste de um vetor, com uma das posições contendo uma lista ligada L_i, cada uma delas associadas aos literais que aparecem em T_i de *f*. Os elementos da lista L_i são índices associados aos literais de T_i. Os índices podem ser positivos ou negativos dependendo de estarem associados à variável ou ao complemento. Esse vetor possui dimensão *m*, número de termos, definido como segue:

TERMS[*i*].coef : posição de valor inteiro (ou real), que contém o coeficiente do *i*-ésimo termo de *f*.

TERMS[*i*].card_inicial : posição de valor inteiro, contém a cardinalidade inicial do *i*-ésimo termo de *f*.

TERMS[*i*].card_parcial : posição de valor inteiro, este valor é decrementado à medida que uma variável é fixada em *TRUE*. Corresponde à cardinalidade atual do *i*-ésimo termo.

TERMS[*i*].var : contém a lista L_i do *i*-ésimo termo de *f*.

TERMS[*i*].atrib : posição que assume valores: *UNDEF* se o termo não está definido; *ACTIVE* se o termo está livre; *TRUE* se o termo está fixado em um; e *FALSE* se o termo está fixado em zero.

A segunda representação de f, consiste em um outro vetor associado a variável x_k e $\bar{x}_k$ de f. Neste também, umas das posições contém uma lista ligada L'_i, que são índices associados aos termos contendo x_k ou $\bar{x}_k$. As posições das variáveis estão relacionadas com os índices pares se o literal corresponde a $\bar{x}_k$ e ímpares caso contrário. O vetor possui dimensão $2n$, definido como segue:

VAR[i].card_inicial : posição de valor inteiro, contém a cardinalidade total do i-ésimo literal de f.

VAR[i].card_pos : posição de valor inteiro, contém a cardinalidade de termos positivos em que o i-ésimo literal aparece.

VAR[i].card_neg : posição de valor inteiro, contém a cardinalidade de termos negativos em que o i-ésimo literal aparece.

VAR[i].terms : contém a lista L'_i, que possui os termos em que a mesma aparece.

VAR[i].atrib : posição de valor booleano, assumindo valores: *UNDEF* se a variável não existe; *ACTIVE* se a variável está livre; *TRUE* se está fixada em um; e *FALSE* se está fixada em zero.

Como uma ilustração das representações internas da função,

$$f(x) = 3x_1x_2\bar{x}_4 - 7x_3\bar{x}_6 - 2x_1\bar{x}_2 + 9x_4x_5x_6 + 4\bar{x}_5x_6 \qquad \text{(Eq 3.3)}$$

veja figura 3.2.

Para se determinar a posição i do vetor VAR, é necessário realizar a seguinte operação: se x_k é variável direta, isto é positiva, $i = 2 \cdot (x_k - 1)$; caso seja complementada $i = abs(2 \cdot (1 - abs(x_k))) + 1$. Assim, para a função da equação 3.3, encontrar a posição de x_1 na estrutura VAR, tem-se $(i = 2 \cdot (1-1)) \Rightarrow i = 0$, logo VAR[0] possui as respectivas atribuições de x_1. No caso de $\bar{x}_4$ tem-se $i = abs(2 \cdot (1 - abs(-4))) + 1 \Rightarrow i = 7$.

3.7.2 *Ramificação na árvore de busca*

Várias regras de ramificação têm sido propostas. No processo de seleção das variáveis do algoritmo ALGEN apresenta-se a seguir uma lista, não exaustiva, dos diversos critérios de ramificação propostas na literatura e utilizadas em certos códigos comerciais ou experimentais.

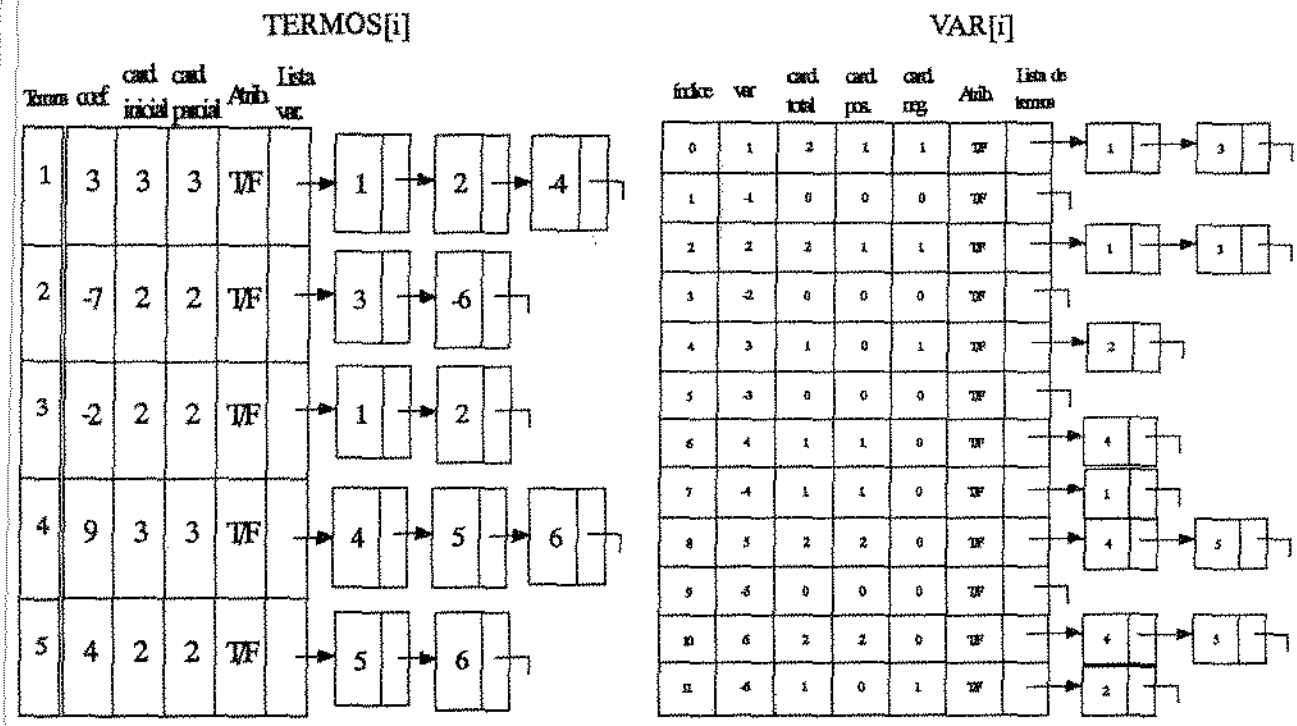


Figura 3.2 Representações internas da função objetivo $f(x)$ correspondente a equação 3.3.

Seleção 1: Ordem lexicográfica

As variáveis livres são consideradas em ordem crescente dos índices. Essa será fixada em *TRUE* se seu custo dentro da função objetivo é positivo e *FALSE* caso contrário.

Seleção 2: Custo máximo

Fixa-se a variável livre de custo máximo dentro da função objetivo.

Seleção 3:

- Para cada variável livre escolha aquela de maior valor absoluto da diferença de suas penalidades (penalidade da variável ir a zero e dela ir a um).
- A variável receberá *TRUE* se a penalidade à zero for maior ou igual a penalidade à um; caso contrário receberá *FALSE*.

Seleção 4:

- Escolhe-se a variável que possuir o menor valor do mínimo entre as penalidades.
- Caso esse valor seja zero, escolhe-se o maior valor entre a diferença absoluta das penalidades.
- A variável receberá *TRUE* se a penalidade à zero for maior ou igual a penalidade à um, caso contrário receberá *FALSE*.

As experiências computacionais realizadas inicialmente mostraram S4 como critério de seleção mais eficiente. S4 pode ser considerado um caso geral de S3. Ambos, S3 e S4, tiveram eficiência muito superior a apresentada por S1 e S2.

3.7.3 *Eficiência no cálculo das penalidades*

Os cálculos de α_{jk} e β_{jk} para os limites e penalidades utilizadas nas proposições 3.2 e 3.4 não são de grande complexidade computacional.

Os valores de α_{jk} que minimizam $\bar{z}_2$ podem ser obtidos resolvendo um problema de fluxo máximo. Este problema é definido como se segue: $\bar{z}_2$ corresponde a um função com coeficientes positivos para termos polinomiais e negativos para termos lineares. Associando-se aos coeficientes positivos um conjunto V_1 de vértices e os coeficientes negativos à um conjunto V_2, obtém-se a instância do problema de fluxo máximo introduzindo-se um vértice fonte s com arcos para todos os vértices em V_1 com capacidades iguais aos coeficientes dos termos correspondentes, arcos partindo de todos os vértices de V_1 para todos os vértices de V_2 com capacidade infinita, e arcos partindo dos vértices de V_2 para um vértice sorvedouro t com capacidades de valor simétrico aos respectivos coeficientes. Os valores de α_{jk} são dados pela fração da capacidade utilizada nos arcos. O valor de $\bar{z}_2$ corresponderá a soma de todos coeficientes positivos subtraída do fluxo máximo obtido. O maior problema dessa determinação de α_{jk}, provêm do alto custo computacional dispendido para resolver o problema de fluxo em rede.

Para dado α_{jk}, valores do β_{jk}, na qual minimize $\bar{z}_2$ e $\bar{z}_4$, pode ser determinado por programação linear.

3.7.4 *Agrupamento de termos idênticos*

A representação da função f_k não é atualizada cada vez que uma variável é fixada no procedimento enumerativo, isto é, o vetor TERM não é compactado para eliminar termos com um literal fixado em *False*, e reduzir termos com um literal fixado em *True*. Ao invés, o efeito de literais fixados é considerado durante os testes. Um problema da não atualização está diretamente relacionada com os cálculo das penalidades e com a escolha da próxima variável a ser enumerada.

Uma forma eficiente de agrupar esses termos seria utilizar uma árvore-AVL [Knuth75]. Cada nodo da árvore-AVL tem um valor igual ao código numérico e um rótulo igual ao coeficiente do termo correspondente.

3.8 Sumário

Este capítulo abordou a combinação do Algoritmo Fundamental, pertencente ao método algébrico, com o algoritmo de enumeração implícita ambos métodos exatos, para a resolução do problema w-MAXSAT e foi detalhada a estratégia de combinação. O primeiro passo para a combinação é a determinação de um subconjunto de N, denominado N_f, de tal forma que $N - N_f$ defina uma k-árvore parcial para um determinado k. Em seguida, aplica-se o algoritmo de enumeração com chamadas ao Algoritmo Fundamental em cada folha da árvore de busca. Apresentaram-se algumas generalizações dos cálculos de limites e penalidades, propostos por Hansen *et al*, utilizados na redução da árvore de busca do algoritmo enumerativo. Em seguida, foram apresentadas duas outras estratégias de combinação dos métodos algébricos enumerativos, com a definição dos novos algoritmos. E finalmente, descreveram-se algumas estratégias computacionais realizadas com a implementação do algoritmo ALGEN, bem como ALGEN-1 e ALGEN-2.

A combinação, formando o ALGEN, pode ser ilustrada segundo o algoritmo apresentado na figura 3.3.

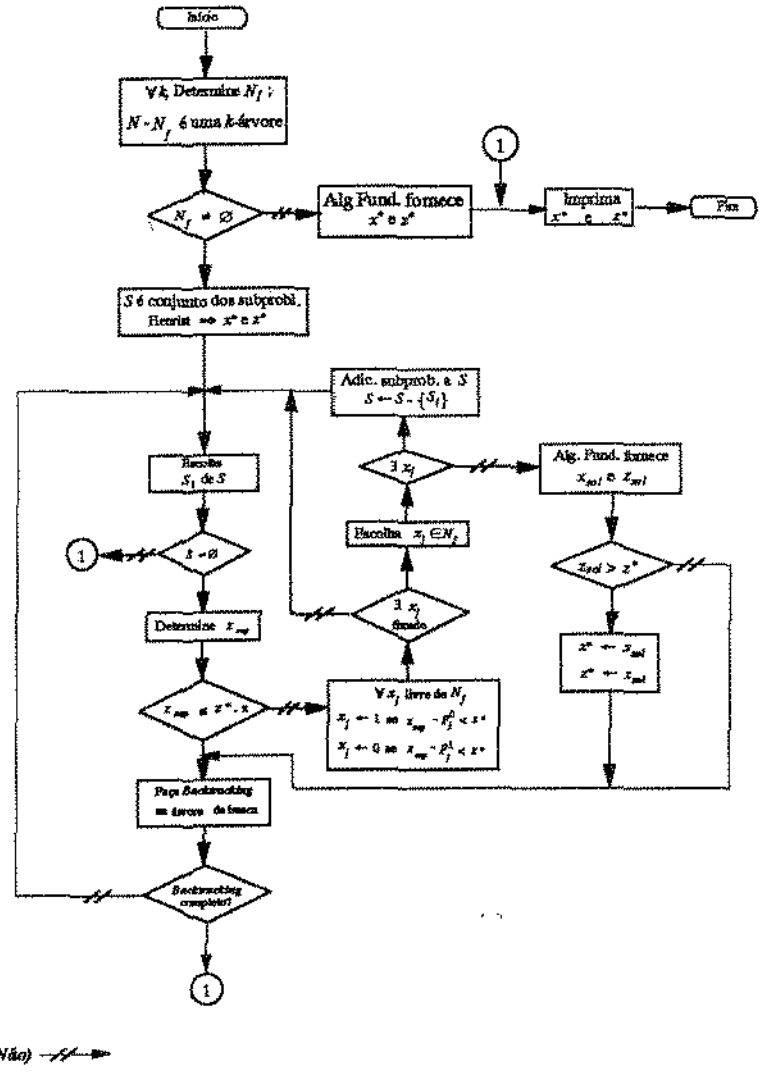


Figura 3.3 Apresentação do Algoritmo ALGEN.

Capítulo 4

Análise das Estratégias para o ALGEN

Como foi visto no capítulo anterior, existe a necessidade de se determinar um subconjunto das variáveis que serão utilizadas pelo algoritmo de enumeração dentro do ALGEN. A determinação desse subconjunto está relacionado com a busca de uma *k*-árvore parcial no grafo de co-ocorrência dado pela instância do problema.

Este capítulo apresenta, uma caracterização do problema relacionado com a estratégia a ser empregada no ALGEN. Dentro deste escopo, são propostas e avaliadas algumas heurísticas.

4.1 Introdução

Uma estratégia fundamental para o ALGEN, está relacionada com a determinação do conjunto formado pelas variáveis a serem enumeradas. Esta estratégia está relacionada com a necessidade de se determinar uma k-árvore parcial no grafo de co-ocorrência definido pela instância do problema. Entendido que o algoritmo algébrico não deverá eliminar variáveis que co-ocorrem com mais de k outras em uma dada iteração.

O interesse no estudo de k-árvores ou k-árvores parciais surgiu através de algumas questões práticas como reabilitação de redes de comunicação na presença de restrições do tipo falha no nodo e na linha; *broadcasting* concorrente, numa rede comum; e avaliação de filas em sistemas de banco de dados relacionais. Para esses problemas, restritos a k-árvores, existem algoritmos eficientes que exploram soluções seguindo a propriedade de separação de k-árvores [ROSE74].

Dentro do contexto de k-árvores para um grafo dado existem dois problemas a serem resolvidos: o primeiro é decidir se o grafo é uma k-árvore e o segundo é encontrar o menor valor para k tal que o grafo permaneça uma k-árvore.

A próxima seção descreve algumas definições de teoria dos grafos. A seção 4.3 define o problema LPKT e seu subproblema. Na seção 4.4, apresentam-se algumas heurísticas para a resolução do LPKT com alguns exemplos.

4.2 Definições

Árvores são conceitos básicos em teoria dos grafos. Existem aplicações de árvores em áreas como química, redes elétricas e teoria dos jogos.

Existem várias maneiras equivalentes de definir árvores, a mais simples é:

Definição 4.1 Uma árvore é um grafo conexo e sem circuito.

Uma definição menos freqüente utiliza o princípio de indução.

Definição 4.2 Um grafo que contém um vértice simples é uma árvore, e uma árvore com $n+1$ vértices é obtida de uma outra com n vértices adicionando-se um novo vértice adjacente a exatamente outro vértice.

Essa definição sugere uma extensão dos conceitos de uma árvore para uma dimensão maior.

Definição 4.3 Seja uma k-árvore um conjunto com k vértices mutuamente adjacentes, uma k-árvore com $n+1$ vértices é determinada adicionando-se, em uma k-árvore com n vértices, um novo vértice adjacente à k vértices antigos mutuamente adjacentes.

As figuras 4.1 e 4.2 indicam, respectivamente, uma 2- e 3-árvore com no máximo seis vértices. Para melhor caracterização de *k*-árvores veja Rose [ROSE74] e Beineke [BP71]

Uma outra definição para *k*-árvore, está relacionada com a clique de um grafo. Uma **clique**, em $G = (N, A)$, onde N é o conjunto de vértices e A o conjunto de arestas, é um subconjunto não vazio de vértices tal que cada par distinto desses possui uma aresta em G. Assim, a definição recursiva de uma *k*-árvore é:

Definição 4.4 Uma k-árvore com k vértices é um grafo cujo conjunto de vértices é uma clique com k vértices (k-clique). Dado qualquer k-árvore $T_k(n)$, com n vértices, uma k-árvore com n+1 vértices é obtida quando o (n+1)-ésimo vértice é adjacente a cada vértice de uma k-clique em $T_k(n)$.

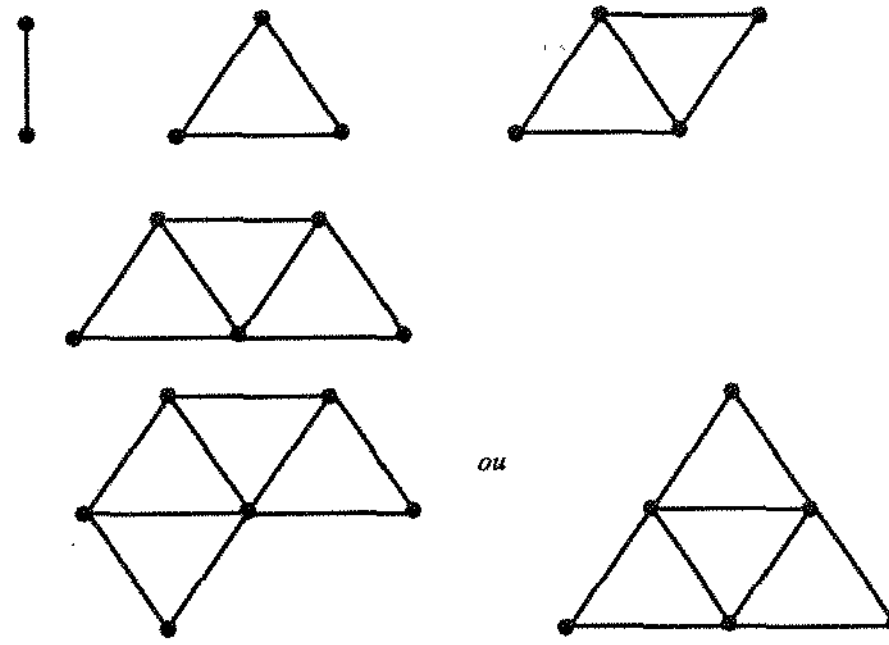
Um grafo parcial pode ser definido como:

Definição 4.5 Dado um grafo G. Um grafo parcial de G contém todos os seus vértices e um subconjunto das arestas.

Isto sugere uma definição de *k*-árvore parcial:

Definição 4.6 Uma k-árvore parcial é qualquer grafo parcial de uma k-árvore.

Figura 4.1 2-árvore com no máximo 6 vértices.



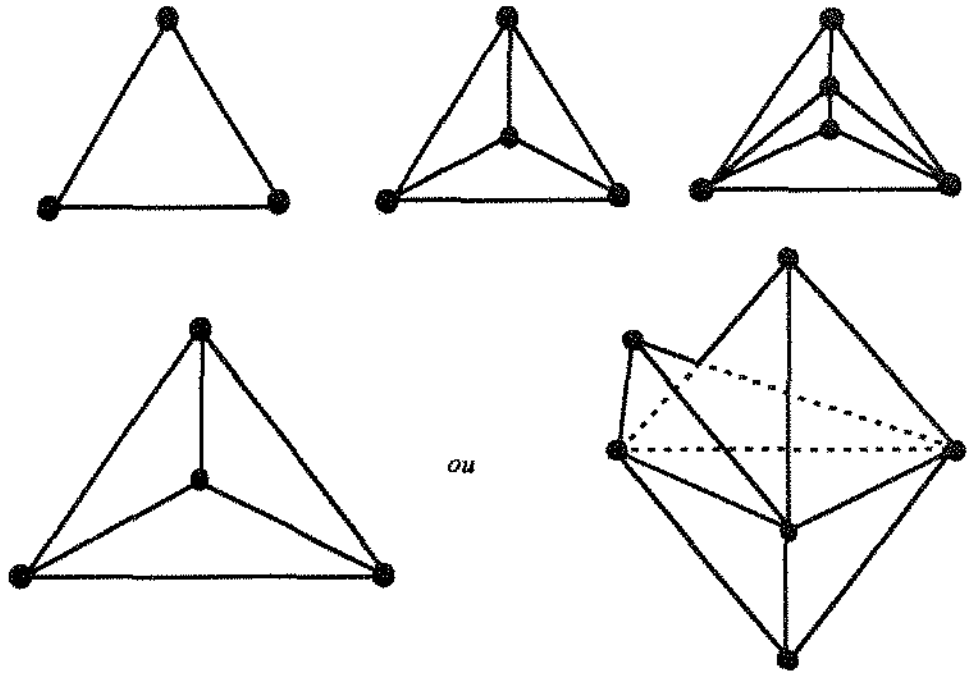


Figura 4.2 3-árvore com no máximo 6 vértices.

4.3 O Problema LPKT

O problema de reconhecimento de uma k-árvore parcial, do inglês *Partial k-tree recognition problem* (*PARTIAL K-TREE*), pode ser entendido como verificar a existência de uma k-árvore parcial num grafo, para um k conhecido. Sua formalização pode ser enunciada como segue:

PARTIAL K-TREE: dados um grafo G e um inteiro l, existe uma k-árvore em G tal que $k \leq l$*?*

PARTIAL K-TREE é *NP-completo* [GJ79]. A demonstração de sua *NP-completude* foi realizada por Arnborg *et al* [ACP87].

Um outro problema envolvendo k-árvore parcial aparece quando deseja-se determinar a k-árvore parcial que contém o maior número de nodos do grafo. O problema de encontrar a maior k-árvore parcial, do inglês *Largest Partial k-tree problem* (LPKT), é definido como:

Problema LPKT: dado um grafo G e um inteiro k, encontrar o maior subconjunto T de N tal que o subgrafo de G, induzido por T, seja uma k-árvore parcial.

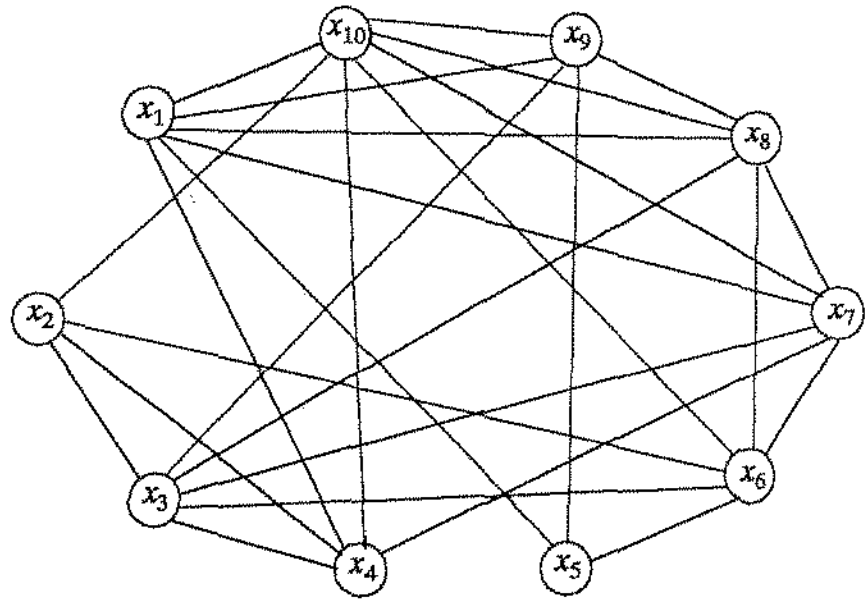


Figura 4.3 Grafo de co-ocorrência G.

Encontrar uma solução ótima para LPKT, é equivalente a determinar o subconjunto $N_f = N - T$ de cardinalidade mínima para um dado k. Entretanto, a cardinalidade de N_f não é o único parâmetro a influenciar no número de vértices explorados na árvore de busca do ALGEN. Pode ser que a fixação de variáveis do conjunto T reduzam significativamente a árvore de busca do algoritmo enumerativo. Observa-se que a solução ótima de LPKT tenta colocar em N_f os vértices com maiores graus do grafo de co-ocorrência G_c, isto é, as variáveis que aparecem em $f(x)$ distribuídas nos termos com o maior número de variáveis diferentes. Essas provavelmente tem um maior significado no valor de $f(x)$, sugerindo serem "boas" variáveis para a enumeração.

O problema LPKT pode ser visto como uma generalização do problema *PARTIAL K-TREE* sendo, portanto, claramente *NP-difícil*. A próxima seção apresenta uma abordagem aproximada para a resolução do LPKT, seguida de alguns resultados computacionais.

4.4 *Heurísticas para o LPKT*

A tendência é determinar soluções aproximadas para o LPKT, uma vez que ele pertence à classe *NP-completo* e que no algoritmo ALGEN não há interesse em um grande consumo de tempo no pré-processamento. De fato o tempo computacional requerido para se determinar uma solução ótima e provar sua otimalidade, pode ser grande quando o problema cresce. Nesta seção, são apresentadas três heurísticas para

resolver o problema LPKT e uma heurística para determinar se um grafo é uma *k*-árvore parcial ou não. Esta última é apresentada a seguir.

4.4.1 Heurística Greedy

Esta heurística verifica se um dado grafo é uma *k*-árvore parcial, para um dado *k*, eliminando repetidamente o vértice de G com menor grau no subgrafo resultante de G após a eliminação anterior. A resposta pode ser "sim" se o grafo é uma *k*-árvore parcial e "talvez" se a heurística não consegue encontrar uma seqüência que prove esta condição. Caso a resposta seja positiva, a eliminação define uma seqüência $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, dos vértices de N. A *k*-árvore parcial é formada pelos π_i's tal que $\partial(\pi_i) \leq k$ (onde ∂ indica o grau do vértice no grafo corrente) para todo $i = 1, ..., n$. Considerando que o vértice ao ser eliminado deve ser simplicial, no passo de eliminação são inseridas todas as arestas entre os seus vizinhos, caso não existam, para torná-lo simplicial. Os algoritmos deste capítulo utilizam a notação abaixo:

As variáveis utilizadas na representação da heurística são:

- G é um grafo, $G = (N, A)$, N conjunto de vértices e A conjunto de arestas;

- k é uma constante;

- v_i é o i-ésimo vértice pertencente a N;

- $\partial(v_i)$ é o grau do vértice v_i, no grafo corrente;

- S é o subconjunto de N, onde os seus vértices não pertençam a *k*-árvore;

- Π é uma seqüência de eliminação dos vértices;

Heurística *Greedy*

Dados G e k
$S \leftarrow \varnothing$
vértices = $|N|$
Enquanto (*vértices* > 0)
  Para todo v_i e N
    Encontre o vértice de menor grau
  Se $\partial(v_i) > k$ **Então**
    PARE, Responda *não foi possível identificar o grafo como uma k-árvore parcial*
  Senão
    $\Pi \leftarrow \Pi + \{v_i\}$, segundo a ordem de eliminação
  Elimine v_i de N inserindo as arestas que faltam para torná-lo simplicial
  vértices ← *vértices* – 1
Fim-enquanto
Π contém uma seqüência que prova que o grafo é uma *k*-árvore parcial.

4.4.1.1 Exemplo

Considere o grafo G da figura 4.3, deseja-se encontrar a maior 4-árvore do grafo G. Este problema será utilizado nos exemplos de execução das heurísticas destes capítulo.

Para este grafo G, a heurística *greedy* não determinou uma 4-árvore no grafo, pois eliminou 5, 2 e retornou não.

4.4.2 Heurística $\mathbb{A}$

A heurística $\mathbb{A}$ consiste de dois passos. No primeiro, é eliminado repetidamente o vértice de G com maior grau no subgrafo induzido de G. Essa eliminação define uma seqüência $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, dos vértices de N. O segundo passo, define uma família de subgrafos de G que correspondem ao induzido por $N_i = N - \{\pi_1, \pi_2, ..., \pi_{i-1}\}$ $(N_1 = N)$ para $i = 1, ..., n$ e chamado o i-ésimo subgrafo G_i. Então utiliza-se a heurística *greedy*, descrita anteriormente, para testar se o subgrafo G_i é uma k-árvore parcial para $i = 1, ..., n$. A primeira resposta positiva obtida pela heurística, por exemplo i^*, define o conjunto de variáveis a serem incorporadas em N_{i^*} que são $\pi_1, ..., \pi_{i^*-1}$. Definindo também, o esquema de eliminação π^* seguido pelo algoritmo algébrico.

O processo de eliminação possui duas formas:

1. Eliminação normal
   Uma eliminação pode ser considerada uma eliminação normal se ao eliminar um vértice v_i do grafo G, não inserindo nenhuma aresta no subgrafo.
2. Eliminação simplicial
   Uma eliminação pode ser considerada uma eliminação simplicial se ao eliminar um vértice v_i do grafo G, ela insere todas as arestas no subgrafo de forma a tornar v_i um vértice simplicial. Esta é utilizada dentro da heurística *greedy*.

A heurística pode ser representada como segue.

Heurística $\mathbb{A}$

Dados G e k
$S \leftarrow \emptyset$
vértices = $|N|$
Enquanto (*vértices* > 0)
  Para todo $v_i \in N$
    Encontre o vértice v_i de maior grau
  $S \leftarrow S + \{v_i\}$
  Elimine v_i de N
  vértices $\leftarrow$ *vértices* – 1
Fim-enquanto

Para $j = 0$ **até** $|N|-1$
  Para $l = 1$ **até** j
    Utilize a eliminação normal para eliminar v_l de S
  Fim-para
  vértices = $|N|$
  $\Pi = \varnothing$
  Enquanto (*vértices* > 0)
    Para todo $v_i \in N$
      Encontre o vértice v_i de menor grau
    $\Pi \leftarrow \Pi + \{v_i\}$, segundo a ordem de eliminação
    Se $(\partial(v_i) > k)$ **então** (Heurística *greedy*)
      Utilize a eliminação simplicial para eliminar v_i de S
    Senão
      Termine o enquanto
    vértices ← *vértices* – 1
  Fim-enquanto
  Se (*vértices* = 0) **Então PARE**
Fim-para
Π contém os vértices da k-árvore parcial na seqüência de eliminação a ser seguida.

4.4.2.1 Exemplo

Para o grafo G da figura 4.3 a seqüência de eliminação Π e o conjunto S fornecidos pela heurística A, juntamente com a ordem de eliminação do vértice, são:

{Ordem de eliminação: Vértice eliminado}
S = {1:7; 2:1}
Π = {3:5; 4:2; 5:4; 6:3; 7:6; 8:8; 9:9; 10:10}

4.4.3 Heurística 𝔅

Esta heurística tem como base a heurística *greedy*, de forma que ao se encontrar um vértice, no passo da eliminação, com grau maior do que k, este é retirado do grafo e prossegue-se com a determinação da k-árvore parcial.

A heurística 𝔅 consiste de eliminar repetidamente o vértice de G com menor grau no subgrafo induzido de G. Essa eliminação define uma seqüência $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, dos vértices de N. A cada passo da eliminação, é verificado se o grau do vértice eliminado é maior do que k, se isto ocorrer retire-o da seqüência e prossiga com a heurística. Quando todos os vértices forem eliminados obtém-se como conjunto de variáveis a serem incorporadas a N_f, os vértices retirados da seqüência. O esquema de eliminação π^* a ser seguido pelo algoritmo algébrico também é definido. A heurística 𝔅 pode ser descrita como segue.

Heurística ℍ

Dados G e k
$S \leftarrow \varnothing$
vértices = $|N|$
Enquanto (*vértices* > 0)
  Para todo $v_i \in N$
    Encontre o vértice v_i de menor grau
  Se $\partial(v_i) > k$ **então**
    $S \leftarrow S + \{v_i\}$
    Utilize a eliminação normal para eliminar v_i de N
  Senão
    $\Pi \leftarrow \Pi + \{v_i\}$, segundo a ordem de eliminação
    Utilize a eliminação simplicial para eliminar v_i de N
  vértices ← *vértices* – 1
Fim-enquanto
Π contém os vértices da k-árvore parcial na seqüência de eliminação a ser seguida.

4.4.3.1 *Exemplo*

Para o grafo G da figura 4.3 a seqüência de eliminação Π e o conjunto S fornecidos pela heurística ℍ, juntamente com a ordem de eliminação do vértice, são:

S = {3:10; 4:7;}
Π = {1:4; 2:2; 5:5; 6:1; 7:3; 8:6; 9:8; 10:9}

4.4.4 *Heurística Try k-width*

A heurística *try k-width* constrói uma k-árvore parcial tentando encontrar uma seqüência, $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, eliminando repetidamente o vértice de G que possui grau k ou o mais próximo de k (e menor) no subgrafo induzido de G. Se não é encontrado um tal vértice a heurística faz uma eliminação simples do vértice de menor grau no grafo corrente. A k-árvore parcial será formada pelos π_i's tal que $\partial(\pi_i) \leq k$ para todo $i = 1, \ldots, n$.

A seguir tem-se uma apresentação da heurística *try k-width*.

Heurística *try k-width*

Dados G e k
$S \leftarrow \varnothing$
vértices = $|N|$
cont = **verdadeiro**
Enquanto (*vértices* > 0)
  real_k = k

Enquanto (*cont*) **faça**
  Para todo $v_i \in N$
    Encontre o vértice v_i, tal que $\partial(v_i) \leq real\_k$ e a diferença entre
      $\partial(v_i)$ e $real\_k$ seja mínima.
  Se foi encontrado um vértice v_i **Então**
    cont = **falso**
  Senão
    $real\_k \leftarrow real\_k + 1$
Fim-enquanto
Se $real\_k > k + 1$ **então**
  $S \leftarrow S + \{v_i\}$
  Utilize a eliminação normal para eliminar v_i de N
Senão
  $\Pi \leftarrow \Pi + \{v_i\}$, segundo a ordem de eliminação
  Utilize a eliminação simplicial para eliminar v_i de N
vértices ← *vértices* – 1
Fim-enquanto
Π contém os vértices da k-árvore parcial na seqüência de eliminação a ser seguida.

4.4.4.1 *Exemplo*

Esta heurística executada em cima do grafo G, figura 4.3, encontra a seqüência de eliminação Π e o conjunto S, dados abaixo.

S = {3:4; 4:10; 5:9}
Π = {1:5; 2:2; 6:8; 7:7; 8:6; 9:3; 10:1}

4.5 *Comparações entre as Heurísticas*

Com o objetivo de se definir qual a melhor heurística para ser utilizada na definição da estratégia do ALGEN, as heurísticas *greedy*, 𝔸, 𝔹 e *try k-width* foram implementadas na linguagem C e testadas utilizando-se um computador SUN SPARC 10 ligado em rede. Os grafos foram gerados randomicamente. Para cada valor da k-árvore utilizado, é apresentada a média dos resultados obtidos em 10 execuções.

As experiências realizadas contemplam três tipos de avaliações das heurísticas: na primeira, considera-se o número de vértices e a densidade do grafo fixados variando o tamanho da k-árvore a ser encontrada; na segunda, fixa-se o valor de k e a densidade do grafo e varia-se o número de vértices no grafo; finalmente, na terceira, varia-se a densidade do grafo e fixa-se o valor de k e o número de vértices. Esses resultados estão apresentados nas tabelas 4.1 e 4.2.

4.5.1 Variando k

A avaliação dos resultados numéricos apresentados pelas heurísticas é baseada nas seguintes estatísticas, para os mesmos testes:

Max $|N_f|$
: maior valor para o conjunto que contém os vértices que não pertençam a *k*-árvore parcial, ou ainda a menor *k*-árvore parcial encontrada.

Min $|N_f|$
: maior *k*-árvore parcial encontrada.

$\mu\,|N_f|$
: média do tamanho do conjunto N_f.

σ
: desvio padrão, representa a dispersão do conjunto das soluções obtidas pelas heurísticas.

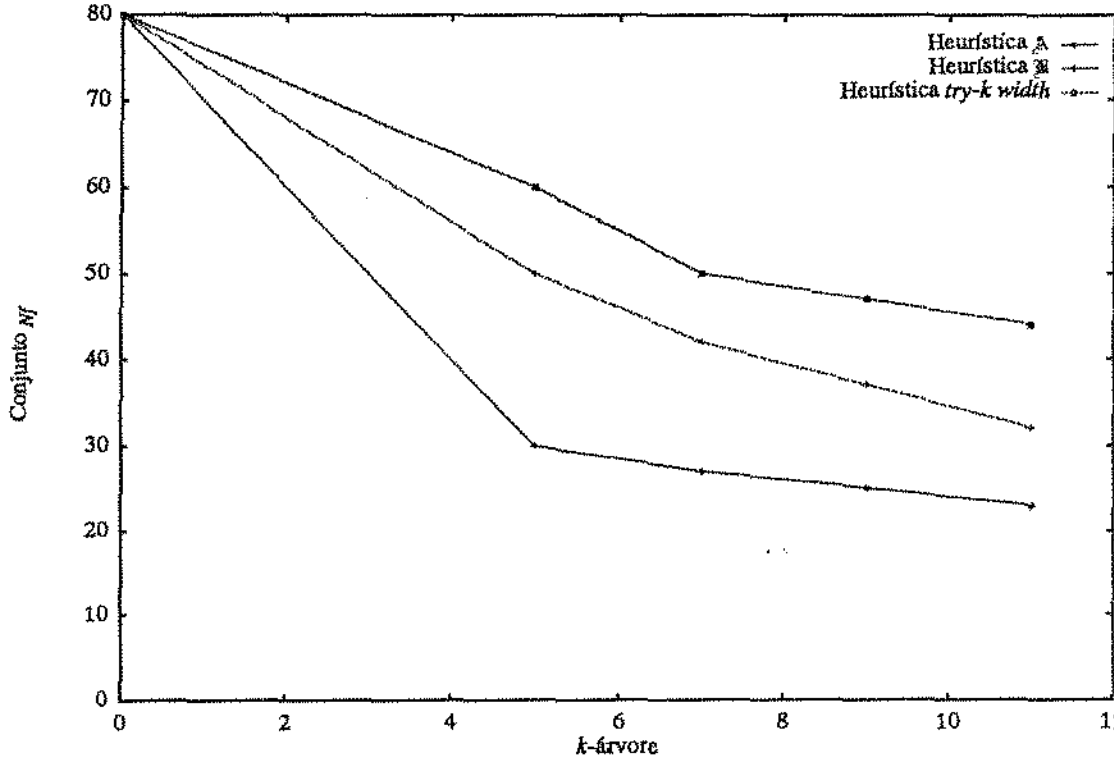
Pode-se observar que a heurística Afoi sempre melhor que as outras para todas as instâncias, obtendo um bom desempenho. Observe que mesmo a heurística *try k-width* que considera valores mais próximos de *k* não conseguiu obter *k*-árvore maiores que a heurística A. Isto pode ser melhor observado considerando o gráfico apresentado na figura 4.4.

Não foram considerados os aspectos de tempo de execução das heurísticas nestes testes devido ao fato de serem muito rápidas para o tamanho do problema apresentado.

Tabela 4.1 Comparações das heurísticas com respeito ao parâmetro *k*. A linha sombreada indica a média, sobre 10 instâncias, para o tamanho do conjunto N_f.

Heurísticas	A								*try k-width*			
Densidade	10%				10%				10%			
Vértices	80				80				80			
k	5	7	9	11	5	7	9	11	5	7	9	11
Max $\|N_f\|$	33	30	29	27	57	48	42	37	68	56	51	50
Min $\|N_f\|$	25	23	20	19	44	38	33	28	56	47	42	39
$\mu\,\|N_f\|$	30	27	25	23	50	42	37	32	60	50	47	44
σ	*0.78*	*0.82*	*0.95*	*0.95*	*1.34*	*1.11*	*0.97*	*0.99*	*1.34*	*1.22*	*0.84*	*1.27*

Figura 4.4 Desempenho das heurísticas durante a variação do tamanho da *k*-árvore, considerando o número de vértices e a densidade do grafo fixados em 80 e 10%, respectivamente.



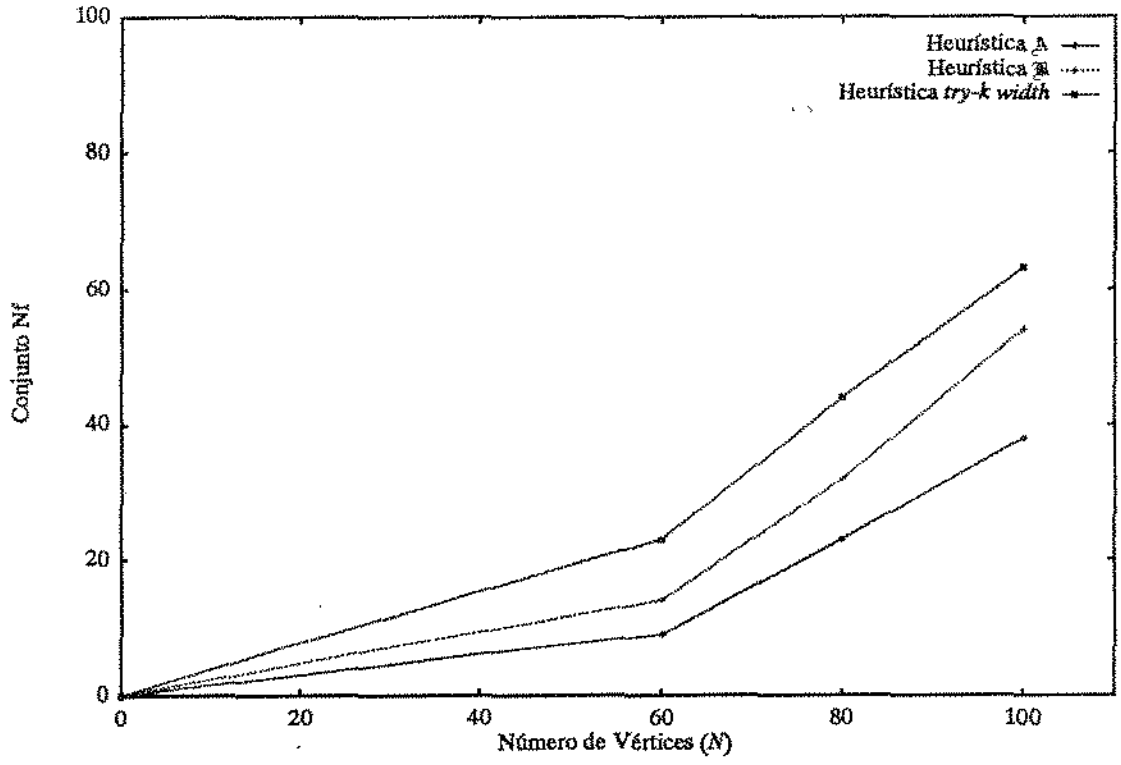
4.5.2 Variando o tamanho do conjunto de vértices

A avaliação dos resultados nesse caso segue as mesmas descritas na seção anterior. Onde agora são considerados fixos o valor de k e a densidade do grafo. O parâmetro aqui analisado é o tamanho do conjunto de vértices do grafo.

Tabela 4.2 Resultados obtidos pelas heurísticas com respeito ao paramêtro do número de vértices do grafo. A linha sombreada indica a média do tamanho do conjunto N_f sobre a execução de 10 instâncias.

Heurísticas	A			B			*try k-width*		
Densidade	10%			10%			10%		
k	11			11			11		
Vértices	60	80	100	60	80	100	60	80	100
Max $\lvert N_f \rvert$	13	27	42	18	37	58	28	50	68
Min $\lvert N_f \rvert$	6	19	35	11	28	50	17	39	60
$\mu\ \lvert N_f \rvert$	**9**	**23**	**38**	**14**	**32**	**54**	**23**	**44**	**63**
σ	*0.70*	*0.95*	*0.83*	*0.77*	*0.99*	*0.83*	*1.19*	*1.27*	*0.76*

Figura 4.5 Qualidade das soluções das heurísticas sobre a variação dos vértices utilizando-se valores fixos para a densidade e k-árvore iguais à 10% e 11-árvore, respectivamente.



4.5.3 *Variando a densidade do grafo*

Para este caso será considerada somente a execução da heurística A, devido ao seu melhor desempenho nos casos anteriores. A avaliação dos resultados ocorrerá da forma descrita anteriormente.

Observe que o desempenho da heurística não se torna muito ruim a medida que o a densidade do grafo cresce. O gráfico apresentado na figura 4.6 fornece uma idéia de crescimento do conjunto N_f (aproximadamente linear).

Tabela 4.3 Representação do desempenho da heurística A durante a variação da densidade do grafo no número de arestas. A linha sombreada indica a média do tamanho do conjunto N_f para a execução de 10 instâncias.

Heurística	A			
Vértices	80			
k	11			
Densidades	5%	10%	15%	20%
Max $\|N_f\|$	12	27	36	43
Min $\|N_f\|$	4	19	31	37
$\mu\ \|N_f\|$	7	23	33	40
σ	*0.82*	*0.95*	*0.52*	*0.55*

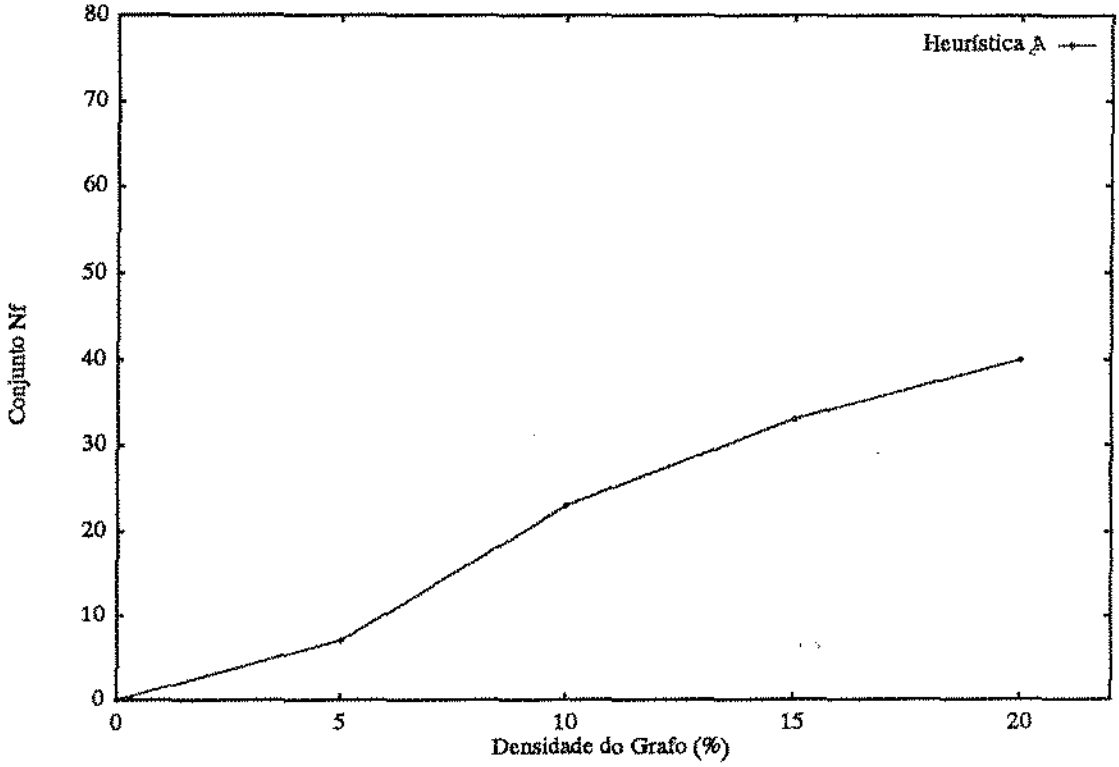


Figura 4.6 Execução da heurística A para instância de 80 vértices no grafo o tamanho da maior *k*-árvore deve ser 11, com variação da densidade do grafo.

4.5.4 Análise do tempo de cpu

Nos resultados obtidos anteriormente não foi demonstrado o desempenho das heurísticas no que diz respeito a tempo de processamento. Com este intuito, é apresentado a seguir o desempenho das heurísticas considerando três tipos de instâncias de 100 à 300 vértices no grafo, onde o valor de k e a densidade estão fixados em 11 e 5%, respectivamente.

A avaliação dos resultados segue as definições apresentadas anteriormente acrescidas de:

tcpu max
: indica o maior tempo de cpu gasto pela heurística.

tcpu min
: indica o menor tempo de cpu gasto pela heurística.

tcpu μ
: indica a média dos tempos de cpu.

σ
: desvio padrão sobre o tempo de cpu gasto pela heurística.

A tabela 4.4 apresenta uma melhor eficiência da heurística Â em se determinar a maior k-árvore, mas um melhor desempenho para a para heurística *try k-width*. Uma melhor comprovação deste fato pode ser observada no gráfico da figura 4.7 que representa a performance das heurísticas.

Tabela 4.4 Desempenho das heurísticas referente ao tempo de processamento gasto.

Heurísticas	Â			B̂			*try k-width*		
Densidade	5%			5%			5%		
k	11			11			11		
Vértices	100	200	300	100	200	300	100	200	300
Max $\lvert N_f \rvert$	22	95	178	32	127	245	49	146	258
Min $\lvert N_f \rvert$	12	85	170	25	125	239	34	137	251
$\mu \lvert N_f \rvert$	**18**	**90**	**173**	**27**	**126**	**243**	**41**	**142**	**254**
tcpu Max	4.71	13.21	46.35	0.01	0.04	0.08	0.06	0.93	4.53
tcpu min	2.78	12.46	45.02	0.01	0.04	0.07	0.03	0.72	3.98
tcpu μ	**4.04**	**12.84**	**45.75**	**0.01**	**0.04**	**0.07**	**0.05**	**0.85**	**4.25**
σ	**0.54**	**0.26**	**0.40**	**0.00**	**0.00**	**0.00**	**0.01**	**0.06**	**0.19**

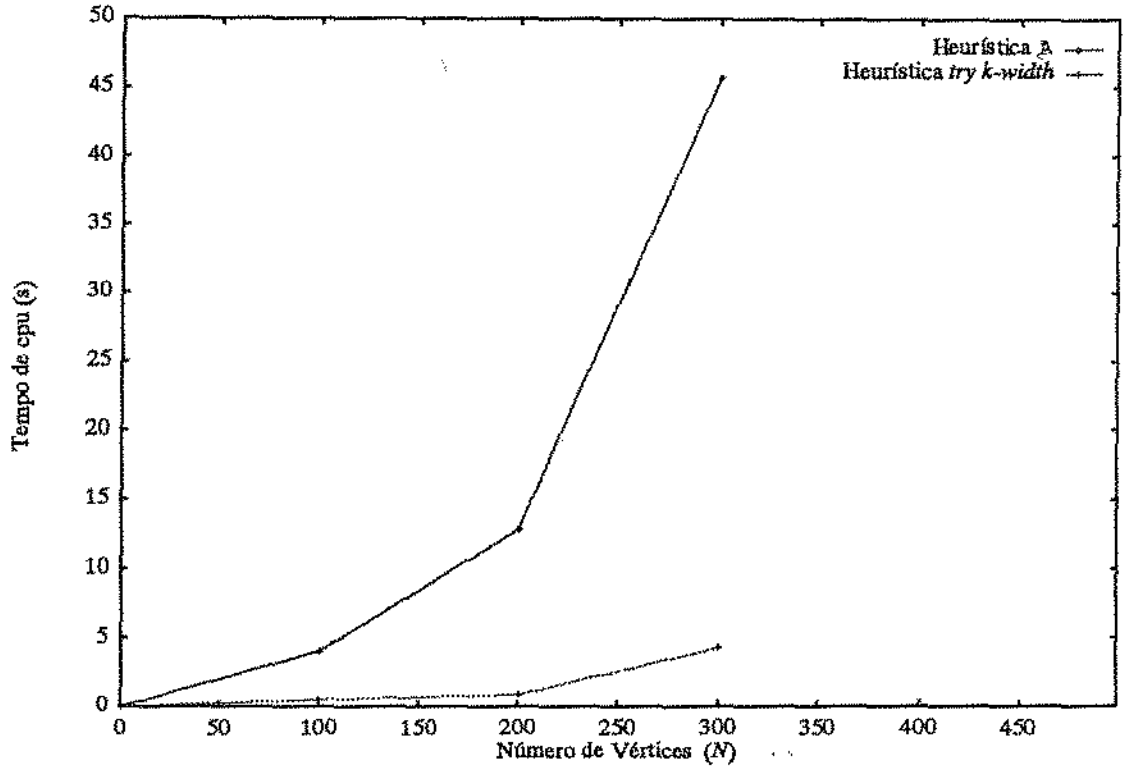


Figura 4.7 Qualidade dos resultados obtidos com a execução das heurísticas na verificação do desempenho das mesmas considerando densidade do grafo e tamanho da k-árvore fixados e três variações do tamanho do conjunto de vértices. Devido o fato do tempo de processamento da heurística B ser muito próximo de zero e não ser possível plotar seus pontos neste gráfico.

4.6 Sensibilidade ao valor k

Considere que seja possível encontrar, para qualquer grafo de co-ocorrência obtido de uma instância do w-MAXSAT, uma k-árvore parcial contendo n, $n-\log n$ e $n-2\cdot\log n$ respectivamente para k igual a 6, 12 e 18. Sendo esta hipótese verdadeira, o Algoritmo ALGEN tem seu desempenho de pior caso limitado por uma função linear (k=18), quadrática (k=12) e cúbica (k=6), pois o conjunto de variáveis a serem enumeradas tem cardinalidade zero, $\log n$ e $2\cdot\log n$, respectivamente. Entretanto, o melhor desempenho sob as condições acima somente será do algoritmo linear (k=18) para valores de n superiores a 750, observe o gráfico na figura 4.8. Isto decorre do crescimento da constante do algoritmo ALGEN com o valor de k, que como visto no capítulo 3 é proporcional a 3^k.

No intuito de obter o comportamento descrito acima para o Algoritmo ALGEN em instâncias reais, propõe-se a seguinte experiência: gera-se um conjunto de instâncias para $n = m$ igual a 24, 32 e 64; determina-se os valores de k para os quais em cada uma dessas instâncias o conjunto N_f possui cardinalidade zero, $\log n$ e $2\cdot\log n$. Desde

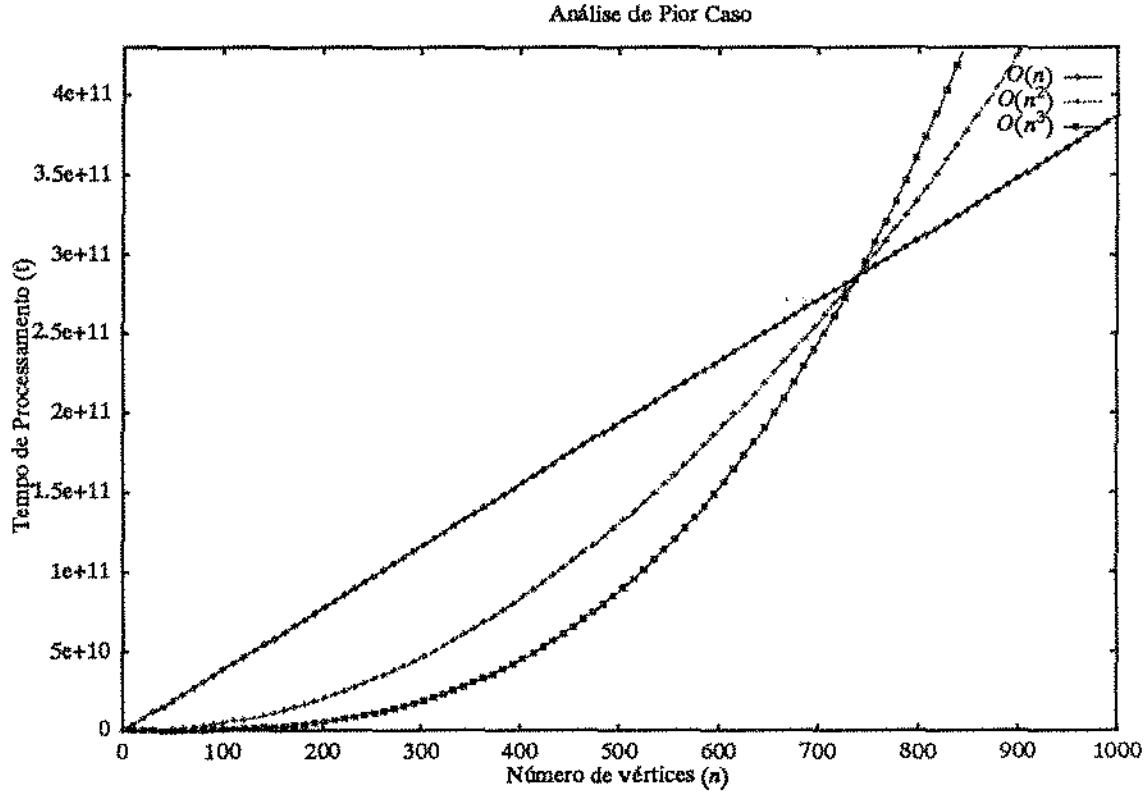


Figura 4.8 Análise de pior caso para o algoritmo ALGEN. O algoritmo possui complexidade linear para um k fixo em 18, uma complexidade quadrática para k fixado em 12 e complexidade cúbica para k fixado em 6.

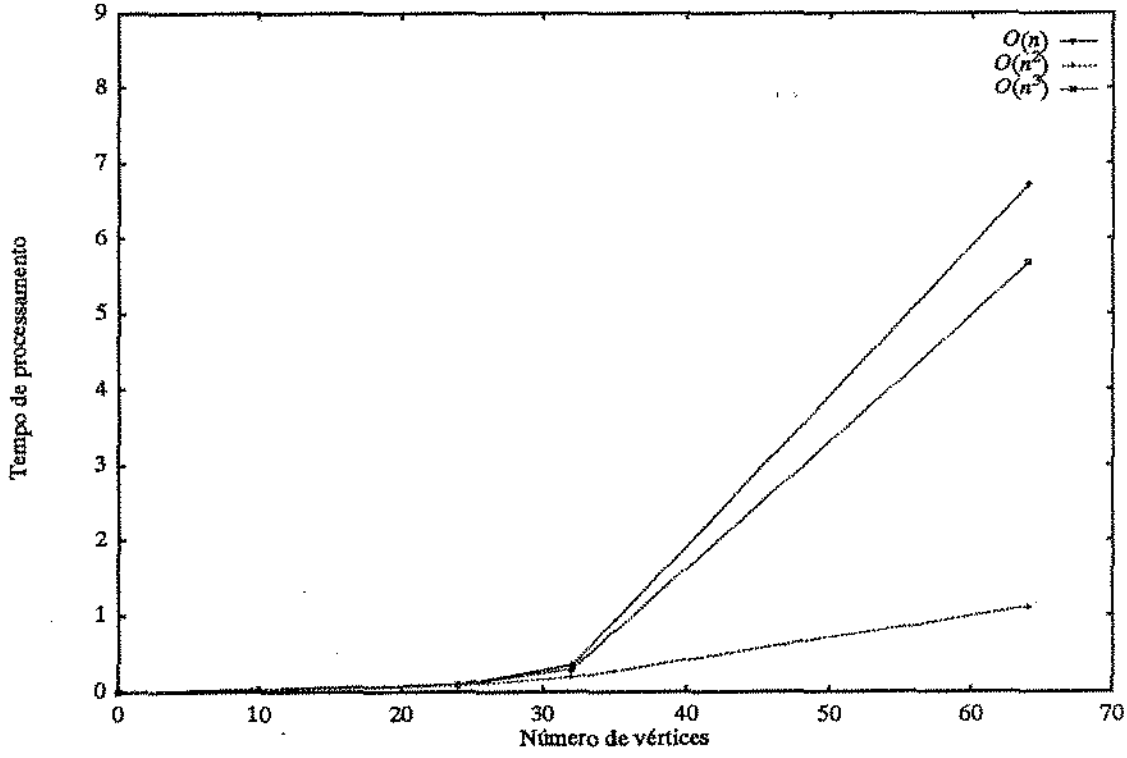
modo, satisfazem-se as condições citadas no parágrafo anterior. Os resultados da experiência são apresentados na tabela 4.5 e no gráfico da figura 4.9. Embora, os resultados não sejam conclusivos pode ser observado que o desempenho do algoritmo linear é inferior ao dos outros dois para valores de n inferiores a 70 conforme previsto.

Tabela 4.5 Qualidade das soluções produzidas para a análise de desempenho dos algoritmo segundo a variação do valor de *k*.

n	24[a]			32[b]			64[c]		
m	24			32			64		
k	7	5	3	11	5	2	18	11	6
$\|N_k\|$	0	2	4	0	5	10	0	6	12
Max tcpu (s)	0.13	0.10	0.17	1.40	0.51	1.05	18.22	1.47	10.53
Min tcpu (s)	0.03	0.05	0.04	0.03	0.05	0.02	0.24	0.37	0.58
μ *tcpu (s)*	**0.07**	**0.07**	**0.08**	**0.35**	**0.19**	**0.30**	**6.70**	**1.10**	**5.66**
σ	0.01	0.01	0.02	0.30	0.09	0.21	7.07	0.45	3.52

a. Média do tempo de cpu realizada sobre 10 instâncias
b. Média do tempo de cpu realizada sobre 5 instâncias.
c. Média do tempo de cpu realizada sobre 3 instâncias.

Figura 4.9 Desempenho dos algoritmos segundo a sensibilidade do valor de *k* a medida que o número de vértices cresce.



4.7 Conclusão

Determinar a maior k-árvore parcial no grafo de co-ocorrência é uma estratégia de grande auxílio na eficiência do algoritmo ALGEN. As experiências mostraram que a heurística $\mathbb{A}$ é melhor do que as outras ($\mathbb{B}$, *try* k) no que diz respeito ao tamanho do conjunto a ser enumerado pelo ALGEN. Pode-se observar ainda, que a determinação de um valor fixo para k não é tarefa fácil, pois para valores muito grandes de k o problema LPKT torna-se difícil de ser resolvido.

Capítulo 5

Resultados Computacionais

Este capítulo descreve os diversos testes realizados para w-MAXSAT. Primeiramente é mencionada a forma de geração dos testes. Em seguida, apresentam-se os resultados obtidos com o Algoritmo Fundamental, Algoritmo Enumerativo, ALGEN e suas diversas estratégias de combinação. É apresentada ainda a influência de uma boa solução inicial no algoritmo enumerativo, bem como a comparação dos resultados conseguidos com os algoritmos.

5.1 Introdução

O objetivo dos resultados computacionais realizados é a comparação da eficiência dos métodos propostos para a resolução do w-MAXSAT. Desde modo são analisadas as diferentes estratégias de combinação utilizadas. Os algoritmos foram implementados em linguagem C e FORTRAN77, sendo executados em um servidor Sun Sparc 1000 com oito processadores. As tabelas apresentadas neste capítulo seguem as definições abaixo:

- n: número de variáveis do problema;
- m: número de termos do problema;
- k: tamanho da k-árvore;
- *Max* $|N_f|$: número máximo de variáveis enumeradas;
- *Min* $|N_f|$: número mínimo de variáveis enumeradas;
- $\mu\,|N_f|$: número médio de variáveis enumeradas;
- *Max Nodos*: número máximo de vértices explorados na árvore de busca do algoritmo enumerativo;
- *Min Nodos*: número mínimo de vértices explorados na árvore de busca do algoritmo enumerativo;
- μ *Nodos*: número médio de vértices explorados na árvore de busca;
- *Max iter*: número máximo de iterações que o algoritmo enumerativo atingiu a folha e foi executado o algoritmo Fundamental;
- *Min iter*: número mínimo de iterações que o algoritmo enumerativo atingiu a folha e foi executado o algoritmo Fundamental;
- μ *iter*: número médio de iterações que atingiu-se a folha;
- *tcpu max Fund*: tempo máximo de cpu dispendido pelo Algoritmo Fundamental;
- *tcpu min Fund*: tempo mínimo de cpu dispendido pelo Algoritmo Fundamental;
- *tcpu* μ: tempo médio de cpu;
- *tcpu max*: tempo máximo de cpu dispendido pelo algoritmo em questão;
- *tcpu min*: tempo mínimo de cpu dispendido pelo algoritmo em questão;
- *tcpu* μ: tempo médio de cpu;
- *tcpu* σ: desvio padrão do tempo de cpu do algoritmo, indicando a dispersão dos tempos obtidos.

5.2 *Geração das instâncias para os testes*

Nos testes abaixos o número m de termos, n de variáveis, k da k-árvore são parâmetros. Os resultados apresentados referem-se a instâncias geradas da seguinte maneira:

1. Problema MAX-3SAT ponderado:
   - **(i)** Os termos contém 3 literais.
   - **(ii)** Existe uma distribuição uma uniforme dos literais nos termos.
   - **(iii)** Cada literal em cada um dos termos pode ser complementada com 50% de probabilidade.
   - **(iv)** Os coeficientes relacionados aos termos são tomados de uma distribuição uniforme entre −1000 e 1000.
2. Problema 3-SAT:

   Essa geração é semelhante ao problema anterior com exceção do item (iv), definido como:
   - **(iv)** Os coeficientes para os termos são tomados iguais à −1 (a entrada para o programa está na forma da equação 1.1, ou seja a de uma soma de produtos).
3. Problema 5-SAT
   - **(i)** Cada termo possui no máximo 5 literais.
   - **(ii)** O número de literais por termo é distribuído uniformemente entre 1, 2, 3, 4 e 5.
   - **(iii)** Os coeficientes são gerados como em 1(iv).

As diretrizes, para a geração dos problemas testes, utiliza-se dos seguintes fatos:

1. O ALGEN processa com variáveis complementadas ou não e minimiza uma função não-linear 0-1 sem restrições;
2. Para cada tamanho de problema são geradas 10 instâncias diferentes, para melhor verificação do desempenho do algoritmo.

5.3 Comparação dos Limites Superiores

Esta seção está interessada na determinação dos limites superiores durante a execução de um algoritmo de pesquisa arborescente. Neste caso, deseja-se resolver o problema com o menor número possível de nodos na árvore de busca.

Tabela 5.1

Tabela produzida pelos resultados fornecidos pelos limites superiores z_2, z_3 e z_4, para 10 instâncias *MAX-3SAT* tamanho $n = 40$ e $m = 80$. O algoritmo possui a mesma solução inicial no processamento das instâncias com cada limite.

Problema *MAX-3SAT*						
Limites Superiores	z_2		z_3		z_4	
n	30	40	30	40	30	40
m	60	80	60	80	60	80
Max Nodos	739	18780	561	9410	628	12419
Min Nodos	164	909	164	772	119	785
μ	**453**	**5258**	**350**	**3340**	**323**	**1363**
tcpu Max (*s*)	0.51	14.09	0.91	11.00	0.58	11.93
tcpu Min (*s*)	0.12	0.91	0.15	1.14	0.13	1.12
tcpu μ (*s*)	**0.32**	**4.71**	**0.39**	**4.04**	**0.33**	**3.81**
tcpu σ (%)	*0.05*	*1.79*	*0.08*	*1.33*	*0.04*	*1.29*

A tabela 5.1 apresenta os resultados produzidos por 10 instâncias de tamanhos 30x60 e 40x80 (variáveis x termos), gerados da forma descrita anteriormente. Pode-se observar que na instâncias 30x60 os resultados produzidos pelos limites possuem uma diferença, considerando o parâmetro tempo, muito pequena. Neste caso não importa o tamanho da árvore de busca, pois o custo computacional para o cálculo das penalidades de z_3 e até z_4 são superiores a de z_2. Quanto a outra instância pode-se observar uma superioridade do limite dado por z_4, tanto no tamanho da árvore de busca quanto no tempo de processamento.

5.4 Sensibilidade da Solução Inicial

O estudo dessa sensibilidade é feito através da análise da execução do algoritmo enumerativo em três situações: a solução inicial é equivalente a -infinito; é obtida por heurística de busca tabu; é a própria solução ótima.

A heurística TABU possui a seguinte configuração:

(I) número de iterações: 20; o procedimento executa até 20 iterações sem obter uma melhora na solução, caso uma solução melhor seja encontrada o contador retorna a zero.

(II) tamanho lista tabu: 60% do número de variáveis do problema.

Tabela 5.2 Resultados obtidos para se efetuar comparações de forma a perceber a sensibilidade de se possuir uma solução inicial "boa".

MAX-3-SAT - 40x80			
Solução Inicial	Ótima	Heurística	m*(-1000)
Max Nodos	12141	11231	15247
Min Nodos	690	802	885
μ	**3120**	**3753**	**4684**
tcpu max (*s*)	12.05	11.77	16.14
tcpu min (*s*)	0,92	1.05	1.59
tcpu μ (*s*)	**3.48**	**4.20**	**5.63**
tcpu σ (%)	*1.28*	*1.40*	*1.90*

Pode ser observado que a diferença entre a solução ótima fornecida ao problema e a solução aproximada, dada pela heurística, é menos de um segundo. Enquanto se a solução inicial é dada por um valor muito ruim esta diferença passa de dois segundos. A realização desta comparação foi feita sobre um problema relativamente pequeno, se for considerado um problema de maior extensão estas diferenças serão bastante significativas.

5.5 *Resultados do Algoritmo Fundamental*

O Algoritmo Fundamental foi codificado em FORTRAN 77, implementado e executado num computador SUN SPARC 10 UNIX system 5 release 4.0.

Tabela 5.3 Resultados obtidos com a execução do Algoritmo Fundamental na sua forma revisitada, ou seja, *BASIC ALGORITHM REVISITED*.

MAX-3SAT					
n	40	50	60	80	30
m	40	50	60	80	60
tcpu max (*s*)	2.06	6.55	11.09	336.07	241.81
tcpu min (*s*)	0.31	0.34	0.40	0.67	13.50
tcpu μ (*s*)	**0.58**	**1.33**	**3.13**	**40.86**	**130.14**
tcpu σ (%)	*0.18*	*0.65*	*1.29*	*34.83*	*43.99*

Este algoritmo foi aplicado a duas classes de instâncias, uma onde o número de termos é igual a n e outra onde é $2n$. Para a primeira classe o algoritmo comportou-se com boa performance, veja a tabela 5.3, isto pode ser melhor observado no gráfico da figura 5.1. Isto se dá pelo fato dessas instâncias possuírem um grafo de co-ocorrência que são k-árvores parciais para pequenos valores de k. Na segunda classe não houve um grande desempenho como na primeira, devido ao crescimento do valor de k no grafo de co-ocorrência que são k-árvores parciais. Para as instâncias 40x80 e 50x100 não foi possível encontrar nenhum resultado com tempo de cpu inferior à 15h.

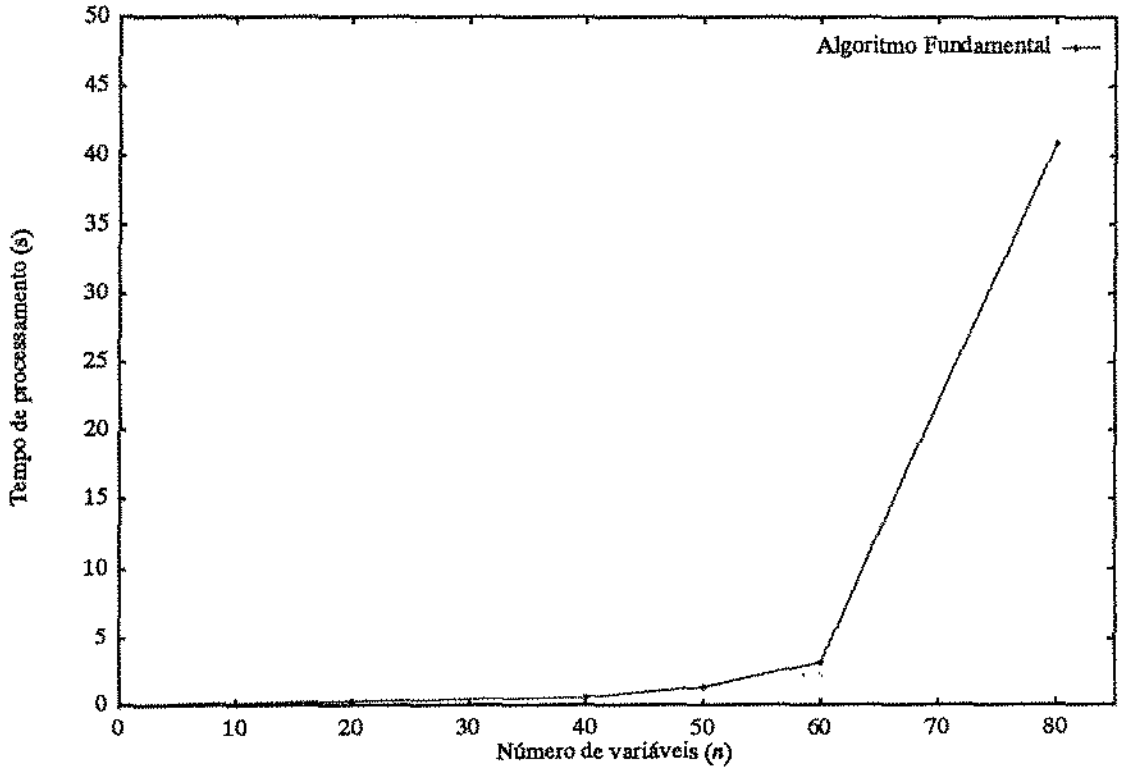


Figura 5.1 Qualidade das soluções dada pela execução do Algoritmo Fundamental, considerando o número de termos igual ao número de variáveis.

5.6 Resultados do Algoritmo Enumerativo

O algoritmo enumerativo foi codificado em C, implementado e executado num computador SUN SPARC 10 UNIX system 5 release 4.0.

Tabela 5.4 Tabela definida com os resultados obtidos com a execução do algoritmo enumerativo puro, utilizado como parte do algoritmo combinado. As linhas sombreadas significam o cálculo das médias.

MAX-3SAT						
n	40	50	60	30	40	50
m	40	50	60	60	80	100
Max Nodos	153	1285	937	634	11231	13585
Min Nodos	63	222	480	112	885	1330
μ	126	628	176	321	3371	6052
tcpu max (*s*)	0.13	1.01	1.07	0.60	12.98	39.21
tcpu min (*s*)	0.06	0.24	0.63	0.12	1.06	2.28
tcpu μ (*s*)	0.11[a]	0.55[a]	0.75[a]	0.35	4.01	11.67[b]
tcpu σ (%)	0.01	0.19	0.20	0.05	1.32	4.57

a. média obtida é inferior a 5 instâncias.
b. média obtida é inferior a 10 instâncias.

Este algoritmo foi aplicado sobre instâncias com o número de termos igual a n e 2n. No primeiro tipo de instâncias o algoritmo teve um comportamento instável, pois ou as instâncias foram resolvidas rapidamente ou o algoritmo não conseguiu resolver em com menos de 15h de cpu (o que ocorreu mais de 50% das vezes). Uma possível razão para isto é que um número pequeno de termos eventualmente não traz informação suficiente para que a árvore seja cortada significativamente. Isto é, obtém-se limites superiores que são ruins. No segundo tipo de instâncias, pode-se verificar a superioridade do algoritmo enumerativo sobre o algébrico. Entretanto, em duas das 10 instâncias de 50x100 o algoritmo enumerativo também não conseguiu resolver em menos de 15h. Isto se atribui a fato de que algumas instâncias são, consideravelmente, mais difíceis que outras.

5.7 Resultados do ALGEN

Este algoritmo foi codificado em C e FORTRAN77, implementado e executado também numa SUN SPARC 10 UNIX. Os resultados apresentados aqui demonstram a sensibilidade do tempo de processamento à largura (k) da *k*-árvore parcial e à relação número de termos x número de variáveis na instância. O cálculo dos limites superiores foram obtidos com a formulação dada por $\bar{z}_4$ (apresentada no capítulo 3).

5.7.1 Variando o tamanho da k-árvore

O objetivo deste teste é verificar o desempenho do ALGEN mediante a variação do tamanho da *k*-árvore considerando o número de variáveis e termos fixados em 80.

Tabela 5.5 Resultados obtidos com instâncias 3-SAT, três variáveis por termo. Os itens em destaque apresentam o valor médio do algoritmo quando executado sobre 10 instâncias.

	MAX 3-SAT - 80x80							
k-tree	5	7	9	11	13	15	17	19
Max $\|N_f\|$	21	17	15	13	10	9	6	4
Min $\|N_f\|$	15	12	10	7	5	3	2	0
μ	**17**	**15**	**13**	**10**	**8**	**6**	**4**	**2**
Max Nodos	23989	19169	7788	2942	670	187	62	10
Min Nodos	91	99	60	36	20	6	3	0
μ	**5064**	**3096**	**1624**	**749**	**260**	**77**	**23**	**4**
Max iter	20344	16450	6760	2532	602	162	62	8
Min iter	31	41	23	20	16	6	4	1
μ	**3732**	**2295**	**1232**	**559**	**205**	**65**	**22**	**4**
tcpu max Fund(s)	698.06	640.36	306.85	126.18	35.87	10.98	6.60	125.46
tcpu min Fund(s)	1.21	1.86	1.26	1.47	1.04	0.69	0.69	0.59
tcpu μ *(s)*	**134.54**	**93.50**	**61.91**	**30.94**	**13.09**	**5.32**	**3.04**	**13.57**
tcpu max ALGEN(*s*)	792.44	717.94	340.76	140.04	39.05	11.95	6.65	125.47
tcpu min ALGEN(*s*)	1.60	2.32	1.56	1.67	1.29	0.73	0.71	0.60
tcpu μ *(s)*	**158.09**	**106.16**	**69.11**	**34.42**	**14.31**	**5.72**	**3.18**	**13.60**
tcpu σ *(%)*	*82.21*	*72.45*	*33.35*	*14.12*	*3.90*	*1.33*	*0.78*	*13.10*

Esta instância possui um grafo de co-ocorrência com densidade de 7%, no entanto pode-se observar na tabela acima que a medida que o *k* cresce consegue-se um conjunto de vértices menor para serem enumerados e conseqüentemente uma decréscimo no tempo de execução do algoritmo.

Na execução onde o valor de *k* é igual a 19, observe que existem várias instâncias onde somente o Algoritmo Fundamental é executado ($|N_f| = 0$).

Pode-se notar ainda que o melhor valor para *k* nesta instância é 17, veja o gráfico da figura 5.2.

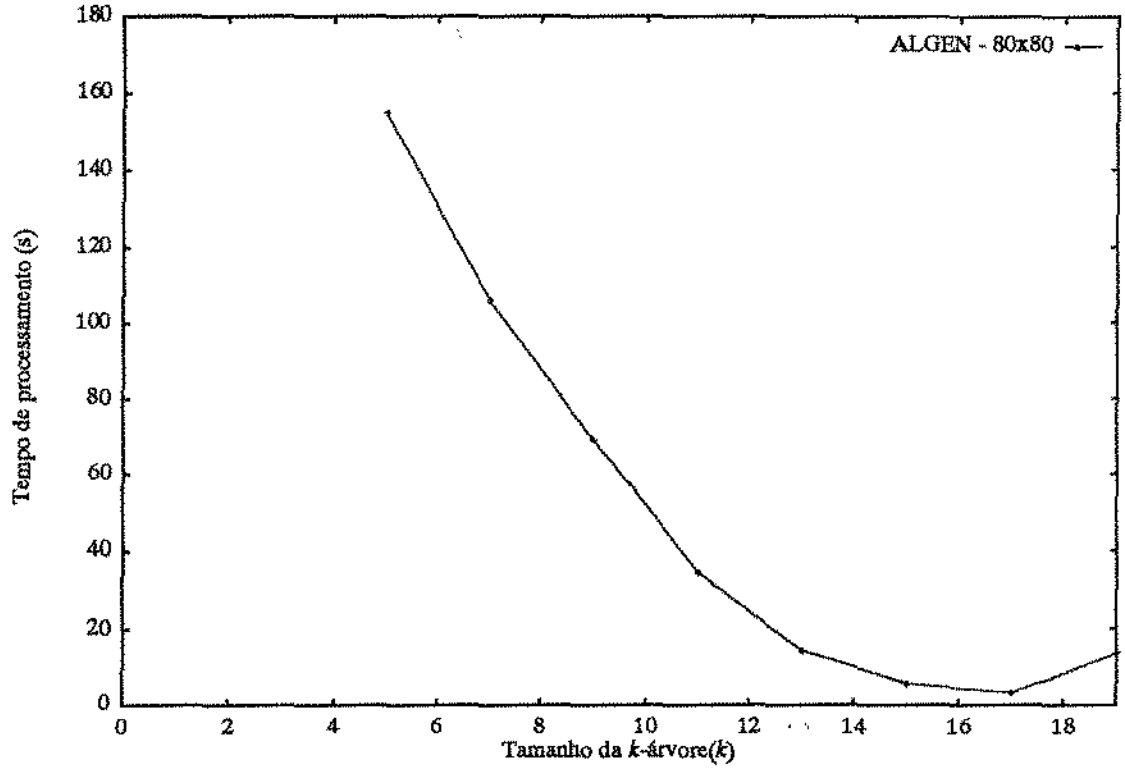


Figura 5.2 Qualidade das soluções dada pela execução do algoritmo ALGEN, resolvendo w-MAX-3SAT de 80x80 com o tamanho da k-árvore variável.

5.7.2 *Variando o tamanho do conjunto de variáveis*

Nesta seção o interesse está no desempenho do algoritmo ALGEN com relação a variação do número de variáveis.

Tabela 5.6

Tabela definida para se observar o desempenho do algoritmo a medida que aumenta-se o valor de m (número de termos) em função de n. A média dos resultados é obtida sobre 10 instâncias diferentes.

MAX-3SAT									
k-tree	7				9				
n	40	50	60	80	30	40	50	60	80
m	40	50	60	80	30	40	50	60	80
Max $\|N_f\|$	6	8	12	17	1	4	6	10	15
Min $\|N_f\|$	2	4	8	12	0	0	3	6	10
μ	**4**	**6**	**10**	**15**	**0**	**2**	**4**	**7**	**13**
Max Nodos	19	82	391	19169	1	9	44	209	7788
Min Nodos	2	10	37	99	0	0	6	18	60
μ	**10**	**35**	**133**	**3096**	**0**	**3**	**16**	**75**	**1624**
Max iter	9	48	260	16450	2	6	31	145	6760
Min iter	1	2	6	41	1	1	5	8	23
μ	**5**	**21**	**75**	**2295**	**1**	**3**	**13**	**44**	**1232**
tcpu max Fund(s)	0.19	1.30	7.28	640.36	0.18	0.47	0.84	4.80	306.85
tcpu min Fund(s)	0.03	0.06	0.18	1.86	0.03	0.04	0.19	0.28	1.26
tcpu μ *(s)*	**0.13**	**0.61**	**2.21**	**93.50**	**0.08**	**0.15**	**0.47**	**1.58**	**61.91**
tcpu max ALGEN(*s*)	0.24	1.55	8.56	717.94	0.18	0.49	0.99	5.52	340.76
tcpu min ALGEN(*s*)	0.04	0.13	0.31	2.32	0.03	0.05	0.22	0.39	1.56
tcpu μ *(s)*	**0.16**	**0.73**	**2.65**	**93.50**	**0.09**	**0.17**	**0.54**	**1.83**	**69.11**
tcpu σ (%)	*0.02*	*0.16*	*0.95*	*72.45*	*0.02*	*0.04*	*0.09*	*0.52*	*33.35*

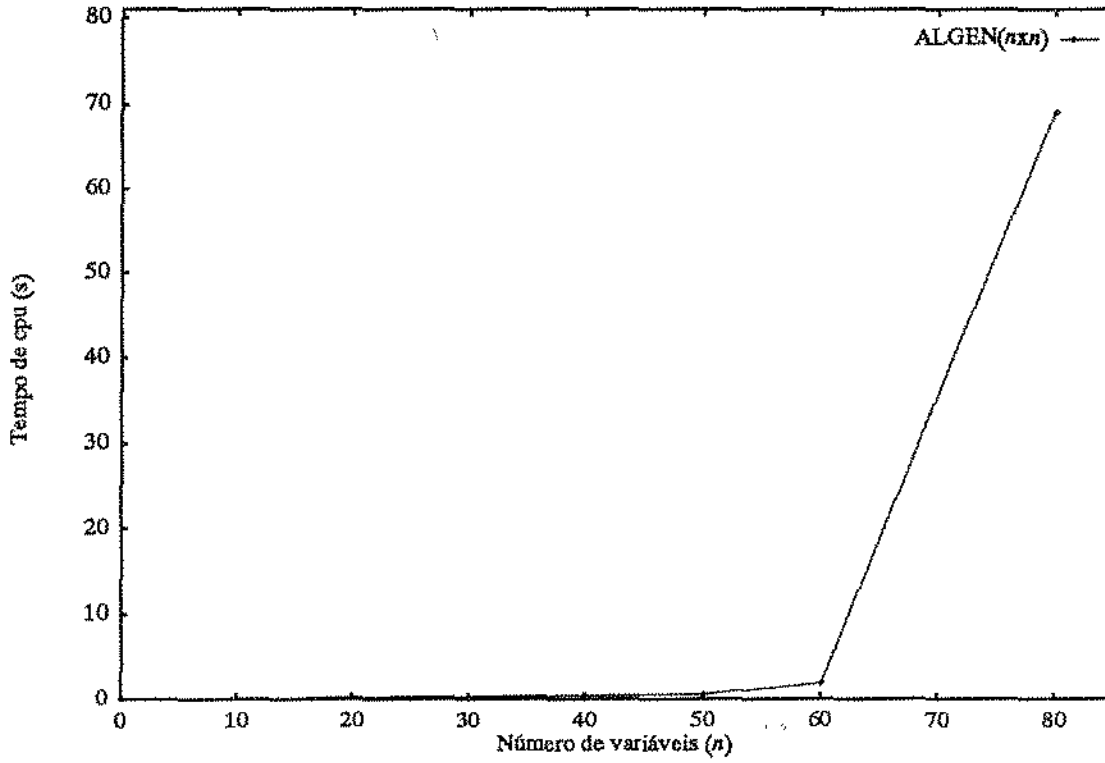


Figura 5.3 Desempenho do ALGEN para a variação do número de variáveis, tal que o número de termos é igual ao de variáveis e k-árvore fixada em 9.

5.8 Resultados do ALGEN-1

Este algoritmo foi codificado em FORTRAN77 e C, implementado e executado numa SUN SPARC 10 UNIX.

5.8.1 Verificando 3-SAT

Este algoritmo foi aplicado às instâncias do problema 3-SAT, gerado da forma apresentada na seção 5.2, para a verificação da satisfatibilidade ou não do problema. O interesse concentra-se em saber se não é SAT, uma vez que a resposta "sim" pode ser determinada por uma heurística simples.

Tabela 5.7 Resultados obtidos com a execução do ALGEN1 para verificar se o problema é satisfatível ou não, a linha em destaque é a média sobre 10 execuções.

Instâncias	percentagem de resposta não	tcpu max (s)	tcpu min (s)	tcpu μ (s)	tcpu σ (s)
40x40	*0%*	*0.12*	*0.00*	*0.05*	*0.02*
40x80	0%	0.17	0.00	0.07	0.03
40x120	0%	1.55	0.00	0.32	0.16
40x160	0%	20.60	0.00	3.94	2.30
40x200	100%	22.26	4.50	9.83	1.77
50x50	0%	0.12	0.00	0.06	0.02
50x100	0%	0.83	0.00	0.16	0.09
50x150	0%	14.58	0.00	2.08	1.49
50x200	10%	315.40	0.00	53.00	34.03
50x250	100%	51.11	18.38	33.11	3.79
60x60	0%	0.14	0.00	0.06	0.02
60x120	0%	16.53	0.0	2.01	1.72
60x180	0%	22.52	0.00	5.80	2.56
60x300	100%	546.52	66.58	222.05	56.41

5.8.2 *Maximizando 3-SAT*

Nesta seção é considerada a maximização de uma função com 3 variáveis por termo. É verificado o desempenho do algoritmo ALGEN1 segundo a variação do tamanho da *k*-árvore e o crescimento do número de termos de acordo com as variáveis.

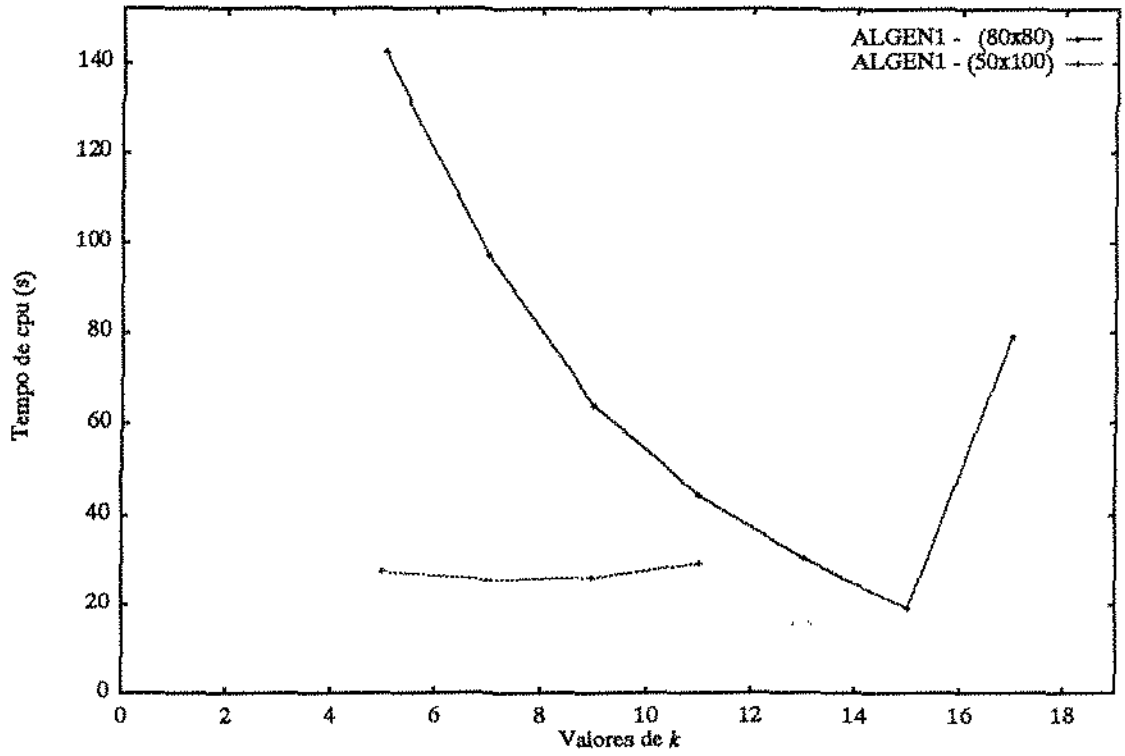
5.8.2.1 *Variando o tamanho da k-árvore*

São utilizadas duas classes de instâncias para a variação do valor de *k*. O objetivo destes testes é tentar determinar o melhor valor para esse parâmetro.

Tabela 5.8 Resultados obtidos com instâncias 3-SAT, três variáveis por termo. Os itens em destaque apresentam o valor médio do algoritmo quando executado sobre 10 instâncias.

	MAX 3-SAT - 80x80							MAX 3-SAT - 50x100			
k-tree	**5**	**7**	**9**	**11**	**13**	**15**	**17**	**5**	**7**	**9**	**11**
Max $\|N_i\|$	22	21	18	15	14	11	8	23	20	18	15
Min $\|N_i\|$	16	14	11	10	8	8	8	19	17	14	12
μ	**19**	**17**	**14**	**12**	**10**	**9**	**8**	**20**	**18**	**16**	**13**
Max Nodos	20654	13744	5468	2596	1648	721	217	18916	15323	4964	2080
Min Nodos	228	179	139	101	66	66	66	1276	1011	880	856
μ	**7413**	**4290**	**2095**	**1100**	**624**	**265**	**147**	**7093**	**4304**	**2526**	**1386**
Max iter	12593	8344	3556	2072	1283	533	181	2058	3506	1891	785
Min iter	43	19	43	34	29	29	29	46	48	60	146
μ	**3464**	**2090**	**1126**	**684**	**406**	**177**	**102**	**698**	**798**	**579**	**405**
tcpu max Fund(s)	388.62	279.54	180.21	108.46	82.80	69.82	675.60	36.11	60.78	59.51	58.20
tcpu min Fund(s)	1.82	1.32	2.68	2.38	2.76	2.64	2.79	0.64	0.97	1.70	5.54
tcpu μ *(s)*	**117.21**	**81.87**	**56.00**	**39.63**	**25.73**	**18.05**	**78.29**	**10.56**	**14.74**	**17.19**	**25.00**
tcpu max ALGEN1 *(s)*	464.89	329.26	201.55	119.90	104.86	73.17	676.43	65.53	99.62	80.67	64.78
tcpu min ALGEN1 *(s)*	2.51	1.87	3.18	2.75	3.05	3.07	3.08	3.42	3.44	4.10	8.64
tcpu μ *(s)*	**142.46**	**97.27**	**63.93**	**44.29**	**30.61**	**19.28**	**78.99**	**27.67**	**28.70**	**26.20**	**29.43**
tcpu σ *(%)*	*56.20*	*36.73*	*22.66*	*14.50*	*11.24*	*7.12*	*70.03*	*8.28*	*10.05*	*7.90*	*6.79*

Figura 5.4 Qualidade das soluções para a execução do ALGEN1 com a variação do tamanho da *k*-árvore. Soluções correspondentes às instâncias 80x80 e 50x100, apresentadas na tabela 5.8.



5.8.2.2 *Variando o conjunto de termos e variáveis*

Para este caso é considerada a variação do conjunto de termos para um número de variáveis fixadas. São consideradas quatro variações, a saber: o número de termos é igual ao número de variáveis, ao dobro das variáveis, ao triplo e ao quádruplo da cardinalidade do conjunto de variáveis. Também é considerado o valor de k fixo, neste caso é igual a 7.

Tabela 5.9 Tabela definida para se observar o desempenho do algoritmo a medida que aumenta-se o valor de m (número de termos) em função do n.

MAX-3SAT												
k-tree	7				7				7			
n	30	30	30	30	40	40	40	40	50	50	50	50
m	30	60	90	120	40	80	120	160	50	100	150	200
Max $\|N_f\|$	4	10	14	15	7	15	20	23	10	20	27	29
Min $\|N_f\|$	3	7	11	12	4	12	17	19	6	17	23	26
μ	**3**	**8**	**12**	**13**	**5**	**13**	**18**	**20**	**8**	**18**	**24**	**27**
Max Nodos	7	345	4236	4837	43	2580	11367	59412	112	15323	68612	224239
Min Nodos	3	35	131	539	6	372	751	5591	29	1011	15934	74595
μ	**5**	**123**	**682**	**1520**	**20**	**1265**	**6694**	**28317**	**71**	**4304**	**42640**	**137011**
Max iter	6	197	908	342	13	643	1722	6193	46	3506	5171	6717
Min iter	1	4	11	23	2	49	14	18	8	48	341	184
μ	**3**	**43**	**120**	**130**	**7**	**273**	**659**	**1705**	**26**	**798**	**1668**	**1945**
tcpu max Fund(s)	0.35	3.79	9.70	5.49	0.30	11.41	55.06	107.46	1.06	60.78	73.38	104.27
tcpu min Fund(s)	0.02	0.06	0.15	0.41	0.06	0.86	2.16	0.29	0.34	0.97	5.85	2.43
tcpu μ *(s)*	**0.09**	**0.84**	**1.49**	**1.94**	**0.18**	**4.22**	**4.17**	**29.16**	**0.67**	**14.74**	**24.23**	**29.16**
tcpu max (s)	0.37	4.57	18.87	15.35	0.36	15.99	46.38	337.35	1.33	99.62	226.96	747.33
tcpu min (s)	0.03	0.14	0.61	1.70	0.07	1.65	3.57	18.17	0.41	3.44	53.97	268.48
tcpu μ *(s)*	**0.11**	**1.20**	**2.98**	**5.80**	**0.23**	**7.00**	**20.21**	**117.13**	**0.85**	**25.70**	**143.55**	**471.52**
tcpu σ *(%)*	*0.03*	*0.43*	*1.87*	*1.51*	*0.03*	*1.71*	*4.85*	*34.40*	*0.11*	*10.05*	*23.08*	*49.51*

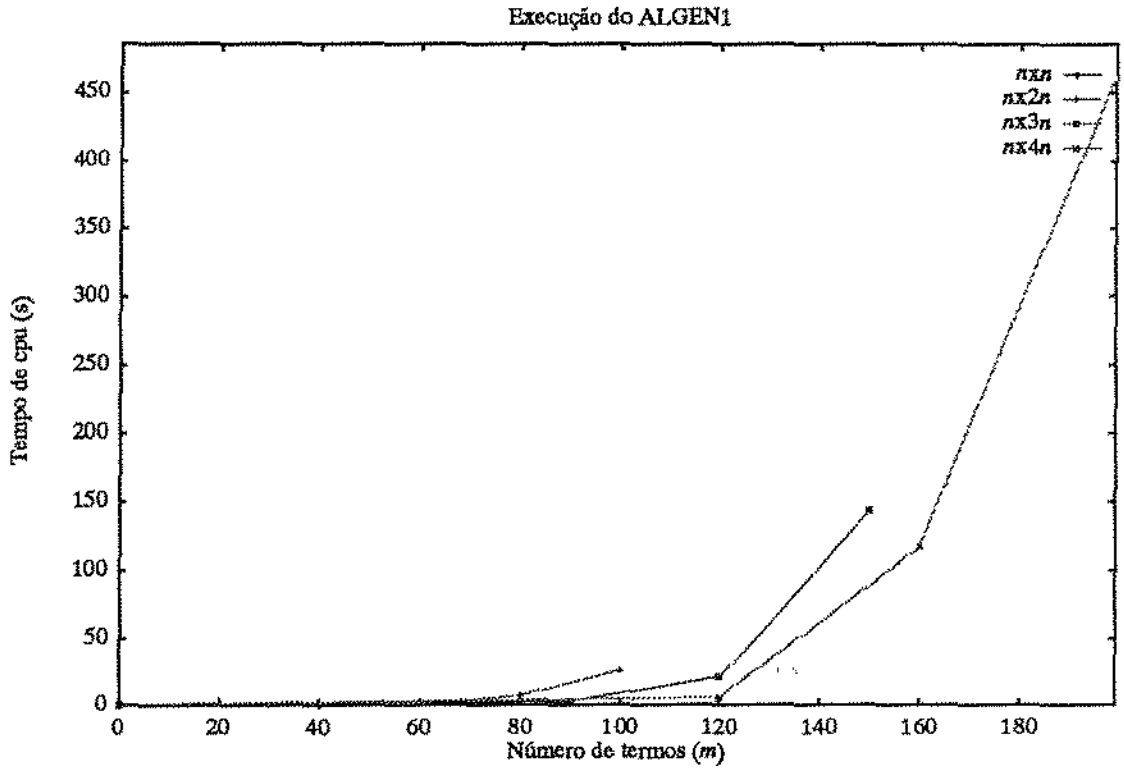


Figura 5.5 Desempenho do algoritmo durante a variação do número de variáveis. Este gráfico representa os dados apresentados na tabela 5.9. Devido o fato dos valores dos tempo de processamento da curva *nxn* serem muito próximos de zero, não é possível sua visualização no gráfico acima.

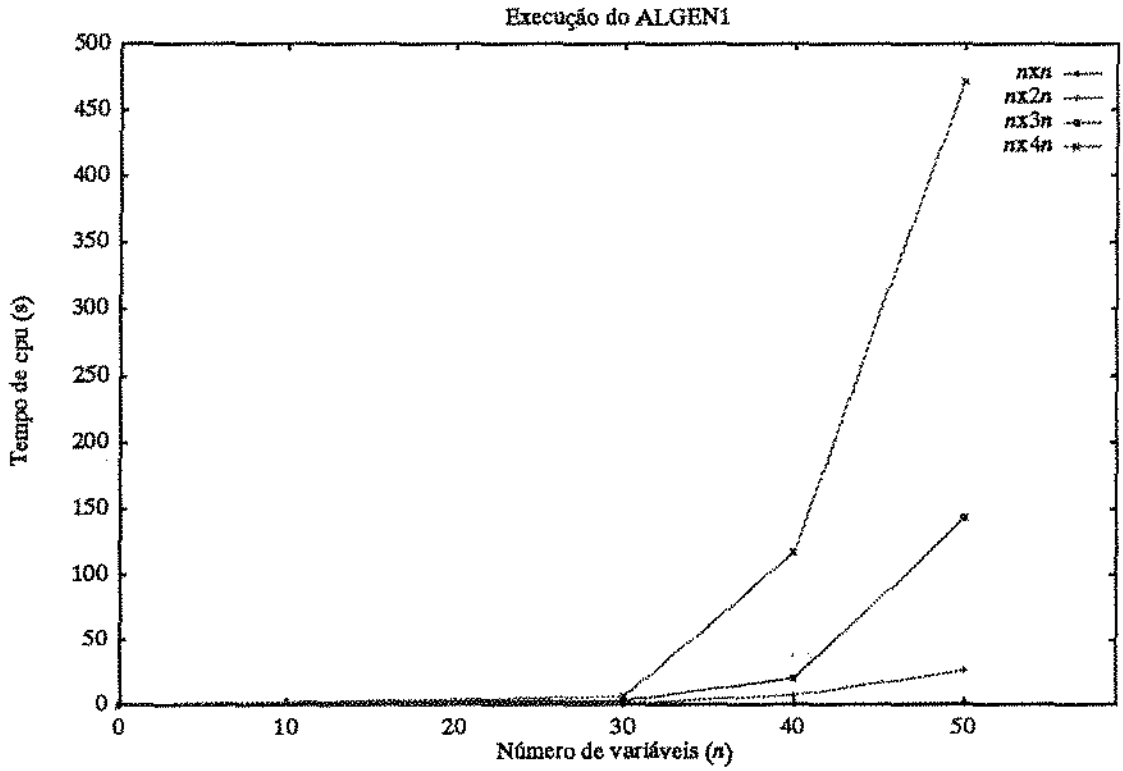


Figura 5.6 Desempenho do algoritmo para a variação do número de variáveis. Vale ressaltar que a curva *nxn* sobrepõe o eixo *n* devido o valores do eixo tempo serem muito próximos de zero.

5.8.3 Maximizando r-SAT

Nesta seção é apresentada a execução do algoritmo ALGEN1 sobre instâncias onde cada termo possuem até 5 literais.

Tabela 5.10 Qualidade das soluções na execução do ALGEN1 quando os termos possuem até 5 literais

MAX-5SAT			
k-tree	7		
n	50	100	100
m	80	100	150
Max $\|N_f\|$	22	32	48
Min $\|N_f\|$	12	20	23
μ	17	26	41
Max Nodos	7090	554718	2929189
Min Nodos	206	156	4935
μ	1599	84781	895352
Max iter	2441	531100	589818
Min iter	10	16	886
μ	368	62536	191963
tcpu max Fund(s)	32.99	18905.32	20012.39
tcpu min Fund(s)	0.16	0.69	49.04
tcpu μ *(s)*	5.61	2293.01	6038.86
tcpu max (s)	50.01	21769.75	29318.04
tcpu min (s)	0.65	1.14	69.83
tcpu μ *(s)*	9.58	2685.37	9016.38
tcpu σ *(%)*	4.89	2252.74	3107.31

5.9 Resultados do ALGEN-2

A implementação deste algoritmo segue a mesma proposta do ALGEN, com as modificações sugeridas no capítulo 3, sem pré-processamento das variáveis. Ele foi executado considerando a variação do tamanho da k-árvore, do número de variáveis e do número de termos.

5.9.1 Variando o tamanho da k-árvore

O algoritmo é executado sobre uma instância de 80 variáveis e 80 termos, considerando a variação para k entre 5 e 17, veja tabela abaixo.

Tabela 5.11 Resultados obtidos com instâncias 3-SAT, três variáveis por termo. Os itens em destaque apresentam o valor médio do algoritmo quando executado sobre 10 instâncias.

	MAX 3-SAT - 80x80						
k-tree	5	7	9	11	13	15	19
Max $\|N_f\|$	21	22	18	18	16	16	16
Min $\|N_f\|$	15	13	11	11	11	11	11
μ	**17**	**18**	**14**	**12**	**12**	**12**	**122**
Max Nodos	23989	12184	8123	4704	4413	4412	4412
Min Nodos	91	742	616	615	566	566	566
μ	**5064**	**4578**	**3214**	**2226**	**1999**	**1967**	**1967**
Max iter	20344	231	297	343	351	350	350
Min iter	31	4	7	9	10	10	10
μ	**3732**	**57**	**80**	**93**	**99**	**100**	**100**
tcpu max Fund(s)	698.06	11.03	17.61	21.26	29.27	20.79	20.65
tcpu min Fund(s)	1.21	0.16	0.34	0.41	0.57	0.57	0.56
tcpu μ *(s)*	**134.54**	**2.62**	**4.71**	**5.64**	**6.80**	**6.17**	**6.13**
tcpu max ALGEN2 *(s)*	792.44	38.86	36.02	36.50	42.07	33.70	33.49
tcpu min ALGEN2 *(s)*	1.60	1.74	1.84	1.68	1.70	1.70	1.74
tcpu μ *(s)*	**155.09**	**13.30**	**12.79**	**12.64**	**11.90**	**11.33**	**11.20**
tcpu σ *(%)*	*82.21*	*4.08*	*3.82*	*3.95*	*4.05*	*3.35*	*3.29*

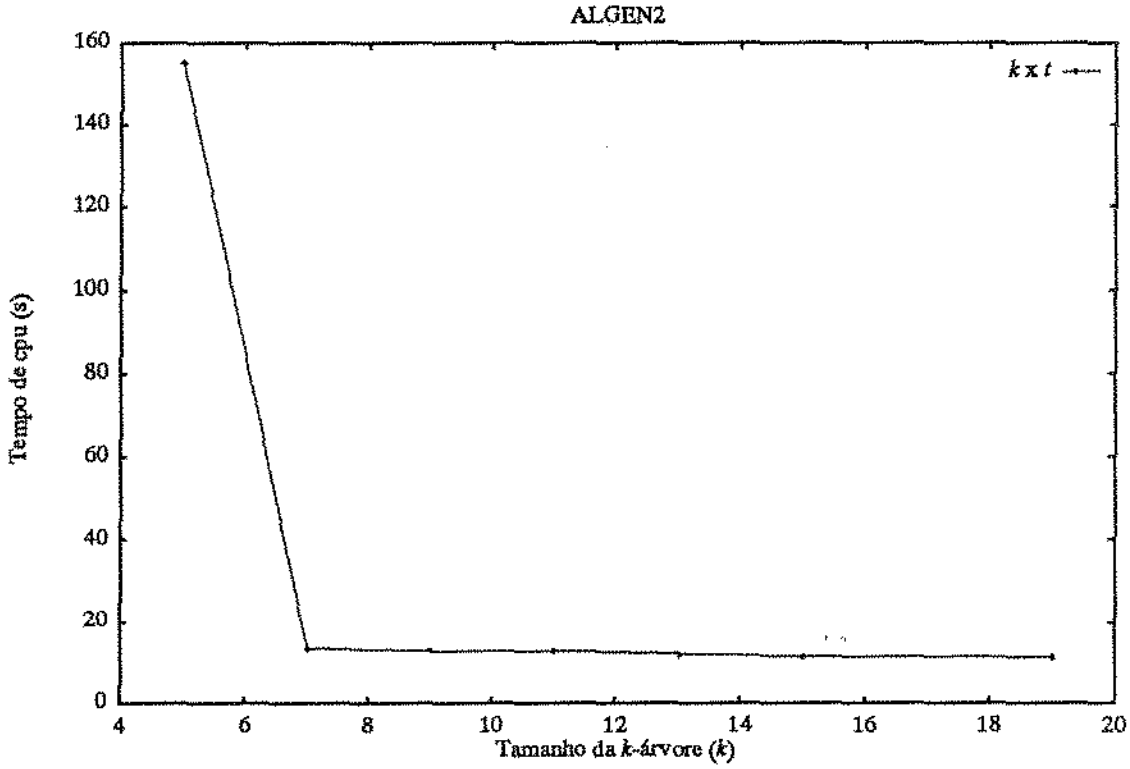


Figura 5.7 Desempenho do algoritmo perante a variação do tamanho da k-árvore. Este gráfico representa do dados apresentados na tabela 5.11.

5.9.2 *Variando o tamanho do conjunto de variáveis e termos*

Nos resultados apresentados abaixo foram consideradas instâncias com o número de termos iguais ao número de variáveis e iguais ao dobro do mesmo.

Tabela 5.12

Tabela definida para demonstrar a qualidade das soluções encontradas pelo algoritmo ALGEN2 a medida que aumenta-se o valor de *m* (número de termos) em função do *n*. Pode-se notar ainda a variação sofrida com o crescimento do número de variáveis.

	MAX-3SAT - nxn				*MAX-3SAT - nx2n*			
k-tree	7				7			
n	40	50	60	80	30	40	50	60
m	40	50	60	80	60	80	100	120
Max $\|N_f\|$	13	14	13	22	21	18	21	27
Min $\|N_f\|$	11	11	11	13	11	11	14	22
μ	**12**	**12**	**12**	**18**	**13**	**13**	**17**	**24**
Max Nodos	358	1020	985	12184	658	8267	15857	27785
Min Nodos	47	113	189	742	82	445	1200	3528
μ	**149**	**391**	**517**	**4578**	**265**	**2314**	**5655**	**14211**
Max iter	6	28	19	231	14	326	369	168
Min iter	1	4	2	4	1	13	3	13
μ	**2**	**11**	**9**	**57**	**6**	**86**	**103**	**55**
tcpu max Fund(s)	0.07	0.62	0.57	11.03	0.24	8.20	8.47	4.66
tcpu min Fund(s)	0.02	0.09	0.06	0.16	0.01	0.28	0.07	0.30
tcpu μ *(s)*	**0.04**	**0.24**	**0.28**	**2.62**	**0.11**	**2.13**	**2.67**	**1.75**
tcpu max (s)	0.32	1.79	2.10	38.86	0.94	26.97	56.88	119.37
tcpu min (s)	0.08	0.31	0.39	1.74	0.12	1.65	2.39	11.19
tcpu μ *(s)*	**0.19**	**0.75**	**1.07**	**13.30**	**0.47**	**6.82**	**18.18**	**57.39**
tcpu σ *(%)*	*0.03*	*0.18*	*0.19*	*4.08*	*0.08*	*2.68*	*5.46*	*12.40*

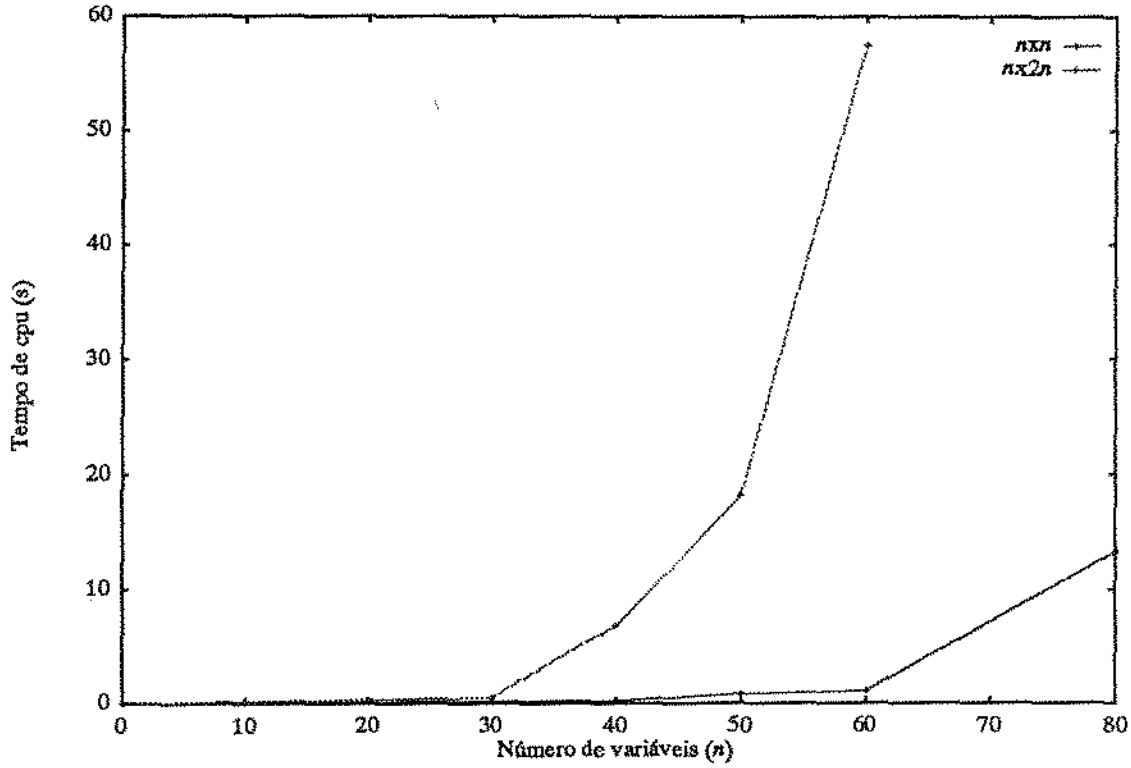


Figura 5.8 Desempenho do algoritmo ALGEN2 segundo a variação do número de variáveis, dados representados na tabela 5.12

5.10 Conclusões

Os resultados apresentados neste capítulo permitem as seguintes conclusões:

- A combinação do Algoritmo Fundamental com o algoritmo enumerativo, apresentada no capítulo 3, obteve grande sucesso.
- Os resultados obtidos por qualquer estratégia de combinação (ALGEN, ALGEN1 e ALGEN2) foram melhores que os encontrados pelos algoritmos fundamental e enumerativo separadamente.
- Os resultados não permitiram concluir com exatidão qual o melhor valor para k. Experiências mostraram que este valor está entre 5 e 9.
- Não se pode afirmar com certeza qual a melhor estratégia a adotar. Pelos resultados tem-se a intuição de que deva ser a do ALGEN2.
- Pode-se observar que nas instâncias nxn (m=n) o algoritmo enumerativo obteve baixa eficiência isto ocorre pelo fato das variáveis estarem bem distribuídas nos termos evitando que o algoritmo extraia "boas" informações (cálculo das penalidades, escolha da variável a ser fixada etc) para reduzir a árvore de busca.
- Outra observação está no mal desempenho do Algoritmo Fundamental quando as instâncias são nx$2n$ (m=$2n$), o que ocorre devido ao fato do grafo de co-ocorrência (da instância a ser resolvida) corresponder a uma k-árvore parcial somente para valores grandes de k.
- Por fim, a tabela 5.2 mostra que por menor que seja a instância, boas soluções iniciais podem ser de grande auxílio na redução da árvore de busca produzida pelo algoritmo de enumeração.

Capítulo 6

Conclusões

Esta dissertação apresenta uma Combinação Algébrico-Enumerativa, estudada e desenvolvida com o propósito de resolver o problema de Máxima Satisfatibilidade Ponderada (w-MAXSAT). Para tanto, três estratégias de combinação foram abordadas e detalhadas (representação da função objetivo, cálculo dos limites superiores, formas de ramificação na árvore de enumeração, particionamento do conjunto de variáveis e implementações).

Esta combinação demonstrou uma eficiência superior, na resolução de diferentes conjuntos de instâncias, a dos algoritmos puros. Foi observado também que para qualquer classe de instâncias em que o número de variáveis enumeradas é limitado por $c \cdot \log n$ o algoritmo tem complexidade polinomial.

6.1 Contribuições

Pode-se citar como contribuições deste trabalho:

- Proposta de um novo algoritmo exato para a resolução do problema w-MAXSAT, com análise de diversas estratégias;
- Novos limites superiores para funções não-lineares 0-1 sem restrição com variáveis complementadas ou não. Este fato tem grande importância porque a transformação acarreta um aumento significativo do número de termos na função;
- A proposição de novas estratégias para o algoritmo levou à formulação de um novo problema em teoria dos grafos (LPKT).

6.2 Extensões

Por se tratar de uma abordagem pouco estudada, ela se constitui num campo de pesquisa muito amplo. Com relação à esta dissertação pode-se citar algumas possíveis extensões:

- Proposta de um método para resolução exata para o LPKT;
- Investigação da combinação entre métodos algébricos e enumerativos, de forma que estes possam ser utilizados alternadamente;
- Utilização de algoritmos enumerativos baseados em programação linear inteira;
- Utilização de relaxações para o Algoritmo Fundamental;
- Utilização de heurísticas durante a execução do algoritmo enumerativo, para melhorar o limite superior.

Apêndice A

Limites Superiores

Este apêndice apresenta as demonstrações dos limites superiores referenciados no capítulo 3.

Preposição 3.1

$$\bar{z}_1 = \sum_{k=1}^{m} max\,(0, \omega_k) \quad \textit{é um limite superior para } f(x).$$

Dem. Claramente $\bar{z}_1$ é um limite superior de $f(x)$.

Proposição 3.2

As penalidades definidas abaixo, podem estar associadas com $\bar{z}_1$.

$$p_j^{'0} = \sum_{\substack{k=1 \\ j \in N_k}}^{m} \alpha_{jk} max\,(0, \omega_k) - \min_{\bar{N}_r = \{j\}} (0, \omega_r)$$

$$p'^{1}_{j} = \sum_{\substack{k=1 \\ j \in \bar{N}_k}}^{m} \alpha_{jk} max(0, \omega_k) - min(0, \omega r) \qquad N_r = \{j\}$$

$$0 \le \alpha_{jk} \le 1 \qquad 1 \le k \le p, \forall j \in N_k \cup \bar{N}_k$$

$$onde \quad \sum_{j \in N_k \cup \bar{N}_k} \alpha_{jk} = 1 \qquad 1 \le k \le p$$

DEM.

$$\bar{z}_1 = \sum_{k=1}^{m} max(0, \omega_k)$$

$$\max_x f(x) \le \sum_{k=1}^{m} max(0, \omega_k) T_k + \sum_{\substack{k=1 \\ |N_k| = 1}}^{m} min(0, \omega_k) T_k + \sum_{\substack{k=1 \\ |\bar{N}_k| = 1}}^{m} min(0, \omega_k) T_k$$

Para todo $k \in \{1, ..., p\}$, $T_k = \prod_{j \in N_k} x_j \prod_{j \in \bar{N}_k} \bar{x}_j \le \sum_{j \in N_k} \alpha_{jk} x_j + \sum_{j \in \bar{N}_k} \alpha_{jk} \bar{x}_j$

Se $\alpha_{jk} \ge 0$ para $j \in N_k \cup \bar{N}_k$ e $\sum_{j \in N_k \cup \bar{N}_k} \alpha_{jk} = 1$; Portanto

$$\max_x f(x) \le \sum_{k=1}^{m} \sum_{j \in N_k} \alpha_{jk} max(0, \omega_k) x_j + \sum_{k=1}^{m} \sum_{j \in \bar{N}_k} \alpha_{jk} max(0, \omega_k) \bar{x}_j +$$
$$+ \sum_{\substack{k=1 \\ |N_k| = 1}}^{m} min(0, \omega_k) T_k + \sum_{\substack{k=1 \\ |\bar{N}_k| = 1}}^{m} min(0, \omega_k) T_k$$

ou

$$\max_x f(x) \leq \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in N_k}}^{m} \alpha_{jk} max(0, \omega_k) x_j + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in N_k \\ |N_k| = 1}}^{m} \alpha_{jk} min(0, \omega_k) x_j +$$

$$+ \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in \bar{N}_k}}^{m} \alpha_{jk} max(0, \omega_k) \bar{x}_j + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in \bar{N}_k \\ |\bar{N}_k| = 1}}^{m} \alpha_{jk} min(0, \omega_k) \bar{x}_j$$

$$\text{como } \bar{z}_1 = \sum_{k=1}^{m} max(0, \omega_k) = \sum_{k=1}^{m} \left(\sum_{j \in N_k} \alpha_{jk} + \sum_{j \in \bar{N}_k} \alpha_{jk} \right) max(0, \omega_k)$$

Logo,

$$\max_x f(x) \leq \bar{z}_1 - \left[\sum_{k=1}^{m} \sum_{j \in N_k} \alpha_{jk} max(0, \omega_k) + \sum_{k=1}^{m} \sum_{j \in \bar{N}_k} \alpha_{jk} max(0, \omega_k) - \right.$$

$$- \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in N_k}}^{m} \alpha_{jk} max(0, \omega_k) x_j - \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in N_k \\ |N_k| = 1}}^{m} min(0, \omega_k) x_j -$$

$$\left. - \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in \bar{N}_k}}^{m} \alpha_{jk} max(0, \omega_k) \bar{x}_j - \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in \bar{N}_k \\ |\bar{N}_k| = 1}}^{m} min(0, \omega_k) \bar{x}_j \right]$$

$$\max_x f(x) \le \bar{z}_1 - \left[\sum_{k=1}^{m} \sum_{j \in N_k} \alpha_{jk} max(0, \omega_k)\ (1 - x_j) \ - \right.$$

$$- \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in \bar{N}_k \\ |\bar{N}_k| = 1}}^{m} \alpha_{jk} min(0, \omega_k) \bar{x}_j + \sum_{k=1}^{m} \sum_{j \in \bar{N}_k} \alpha_{jk} max(0, \omega_k)\ (1 - \bar{x}_j) -$$

$$\left. - \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in N_k \\ |N_k| = 1}}^{m} \alpha_{jk} min(0, \omega_k) x_j \right]$$

$$\max_x f(x) \le \bar{z}_1 - \left[\sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in N_k}}^{m} \alpha_{jk} max(0, \omega_k) \bar{x}_j - \sum_{\substack{j=1 \\ j \in \bar{N}_k \\ |\bar{N}_k| = 1}}^{n} min(0, \omega_k) \bar{x}_j + \right.$$

$$\left. + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \in \bar{N}_k}}^{m} \alpha_{jk} max(0, \omega_k) x_j - \sum_{\substack{j=1 \\ j \in N_k \\ |N_k| = 1}}^{n} min(0, \omega_k) x_j \right]$$

Substituindo p'^0_j e p'^1_j então teremos

$$\max_x f(x) \le \bar{z}_1 - \sum_{j=1}^{n} p'^0_j \bar{x}_j - \sum_{j=1}^{n} p'^1_j x_j$$

$$\max_x f(x) \leq \bar{z}_1 - \sum_{j=1}^{n} p'^0_j (1-x_j) - \sum_{j=1}^{n} p'^1_j x_j$$

o que mostra que as p'^0_j e p'^1_j são penalidades aditivas.

Corolário 3.2.1

$\bar{z}_2 = \bar{z}_1 - \sum_{j=1}^{n} min\,(p'^0_j, p'^1_j)$ *é um limite superior para f(x).*

Podemos melhorar $\bar{z}_2$. Considere $q_j = p'^0_j - p'^1_j \quad \forall\, j = 1, \ldots, n$.

Proposição 3.3

As penalidades, abaixo, podem ser associadas com o limite superior $\bar{z}_2$.

$$p''^0_j = r^0_j + \sum_{\substack{(k,j') \\ j \in \bar{N}_k \\ \omega_k < 0 \\ j' \in \bar{N}_k}} min\,(-\omega_k, r^1_{j'}) + \sum_{\substack{(k,j') \\ j \in \bar{N}_k \\ \omega_k < 0 \\ j' \in N_k}} min\,(-\omega_k, r^0_{j'})$$

$$p''^1_j = r^1_j + \sum_{\substack{(k,j') \\ j \in N_k \\ \omega_k < 0 \\ j' \in \bar{N}_k}} min\,(-\omega_k, r^1_{j'}) + \sum_{\substack{(k,j') \\ j \in N_k \\ \omega_k < 0 \\ j' \in N_k}} min\,(-\omega_k, r^0_{j'})$$

onde,

$$r^0_j = max\,(0, q_j)$$
$$r^1_j = -min\,(0, q_j) \quad e$$

para todo (k,j') são tal que $\omega_k < 0$ e $|N_k| = 2, |\bar{N}_k| = 2$.

DEM.

Para se demonstrar que as penalidades p''^0_j e p''^1_j estão associadas com o limite superior $\bar{z}_2$, basta considerar:

1. $\bar{z}_2$ é um limite superior para todo x_j

2. Um particonamento do conjunto de soluções

3. Se x_j (j=1, .., n) é fixado a 0 ou a 1, $\bar{z}_2$ será decrescido de no máximo p''^0_j ou p''^1_j, respectivamente.

Tem-se que $\bar{z}'_2 = \bar{z}_2 + \sum_{|N_k \cup \bar{N}_k| = 2} \omega_k T_k$ é um limite superior.

Sua demonstração pode ser considerada idêntica a de $\bar{z}_2$, onde o $\sum_{|N_k \cup \bar{N}_k| = 2} \omega_k T_k$ é incorporado no início do desenvolvimento.

Assim, basta mostra que $p''^0_j \geq \Delta z'^0_{2_j}$ e $p''^1_j \geq \Delta z'^1_{2_j}$.

Isto pode ser verificado objetivamente se $p'^0_j < p'^1_j$ tem-se q_j e os termos restantes em p'^0_j correspondem as possibilidades (exautivas) dos valores que podem ser assumidos pelas variáveis que compõem os termos de cardinalidade 2 com x_j

■

Corolário 3.3.1

$\bar{z}3 = \bar{z}_2 - \max_{j=1,\ldots,n} \min (p_j{}''^0, p_j{}''^1)$ *é um limite superior para f(x).*

Proposição 3.4

$$\bar{z}_4 = \bar{z}_2 - \sum_{\substack{k=1 \\ \omega_k < 0 \\ |N_k \cup \bar{N}_k| \geq 2}}^{m} min\Big(-\omega_k, \min_{j \in N_k} (\beta_{jk} max(0, q_j)), \min_{j \in \bar{N}_k} (-\gamma_{jk} min(0, q_j))\Big)$$

onde $0 \leq \beta_{jk} \leq 1$ *e* $\sum_{\substack{k=1 \\ |N_k| \geq 2}}^{m} \beta_{jk}$, $0 \leq \gamma_{jk} \leq 1$ *e* $\sum_{\substack{k=1 \\ |\bar{N}_k| \geq 2}}^{m} \gamma_{jk}$ *para todo j=1,...,n e k = 1, ..., m é um limite superior para f(x).*

DEM.

$$f(x) = \sum_{k=1}^{m} max(0, \omega_k) T_k + \sum_{\substack{k=1 \\ |N_k| = 1}}^{m} min(0, \omega_k) T_k + \sum_{\substack{k=1 \\ |N_k| > 1}}^{m} min(0, \omega_k) \quad \text{e}$$

$$\sum_{j=1}^{n} q_j x_j \geq \sum_{k=1}^{m} max(0, \omega_k) T_k + \sum_{\substack{k=1 \\ |N_k| = 1}}^{m} min(0, \omega_k) T_k .$$

Além disso,

$$\sum_{j=1}^{n} q_j x_j \geq \sum_{j=1}^{n} \sum_{\substack{k=1 \\ |N_k| > 1 \\ j \in N_k}}^{m} \beta_{jk} max(0, q_j) x_j + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ |\bar{N}_k| > 1 \\ j \in \bar{N}_k}}^{m} \gamma_{jk} min(0, q_j) x_j$$

$$\max_x f(x) \leq \sum_{\substack{k=1 \\ |N_k \cup \bar{N}_k| > 1}}^{m} \left(\sum_{j \in N_k} \beta_{jk} max(0, q_j) x_j + \sum_{j \in \bar{N}_k} \gamma_{jk} min(0, q_j) x_j + \right.$$

$$\left. + \; min(0, \omega_k) T_k \right)$$

como $\sum_{\substack{k=1 \\ |N_k \cup \bar{N}_k| > 1}}^{m} \sum_{j \in N_k} \beta_{jk} max(0, q_j) x_j = \sum_{j=1}^{n} max(0, q_j) = \bar{z}_3$, portanto

$$\bar{z}_3 - \left(\sum_{\substack{k=1 \\ |N_k \cup \bar{N}_k| > 1}}^{m} \left(\sum_{j \in N_k} \beta_{jk} max(0, q_j) \; (1 - x_j) + \sum_{j \in \bar{N}_k} -\gamma_{jk} min(0, q_j) x_j - \right. \right.$$

$$\left. \left. - min(0, \omega_k) T_k \right) \right) \geq \max_x f(x)$$

e para todo k tal que $|N_k \cup \bar{N}_k| > 1$,

$$min\left(-min(0, \omega_k), \min_{j \in N_k} (\beta_{jk} max(0, q_j)), \min_{j \in \bar{N}_k} (\gamma_{jk} - min(0, q_j)) \right)$$

$$\leq \sum_{j \in N_k} \beta_{jk} max(0, q_j) \; (1 - x_j) + \sum_{j \in \bar{N}_k} \gamma_{jk} - min(0, q_j) x_j - min(0, \omega_k) T_k$$

segue o resultado.

■

Referências

[ACP87] S. Arnborg, D.G. Corneil e A. Proskurowski, "Complexity of finding embeddings in *k*-tree". *Siam Journal Alg. Disc. Math.*, 8(2):277-284, 1987.

[APT79] B. Aspvall, M.F. Plass e R.E. Tarjan, "A linear-time algorithm for testing the truth of certain quantied Boolean formulas", *Information Processing Letters*, 8:121-123, 1979

[ASM85] P. Asirelli, M. de Santis e A. Martelli, "Integrity constraints in logic databases", *Journal of Logic Programming*, 3:221-232, 1985.

[Bala63] E. Balas, "Linear programming with zero-one variables", *Proceedings of the Third Scientific Session on Statistics*, Bucharest, December 5-7, 1963.

[Bell57] R. Bellman, "Dynamic Programming", (Princeton/NJ:Princeton Univ. Press), 1957.

[BF76] M.A. Breur e A.D. friedman, "Diagnosis and Reliable Design of Digital Systems", *Computer Science Press*, 1976.

[BM80] J.A. Bondy e U.S.R. Murty, "Graph Theory with Applications", (North Holland, New York), 1980.

[BS85] B.C. Buchanan e E.H. Shortliffe. "Rule-based Expert Systems - The MYCIN Experiments of the Stanford Heuristic Programming Project", Addison-Wesley, Reading, 1985.

[BW73] G.H. Brandley e P.N. Wahi, "An Algorithm for Interger Linear Programming: A Combined Algebraic e Enumeration Approach", *Operations Research*, 18(1):45-60, 1973.

[Cook71] S. Cook, "The complexity of theorem proving procedures", *Proceedings of the Third Symposium of the Association of Computing Machinery on the Theory of Computing*, 151-158, 1971

[CHJ90] Y. Crama, P. Hansen e B. Jaumard. "The Basic Algorithm for Pseudo-Boolean Programming Revisited". *Discrete Applied Mathematics*, 29:171-185, 1990.

[Drie66] N.J. Driebeek, "An algorithm for the solution of mixed integer programming problems", Management Science, 12:576-587, 1966.

[DG84] W.F. Dowling e J.H. Gallier, "Linear-time algorithms for testing the satisfiability of propositional Horn formulae", *Journal of Logic Programming*, 3:267-284, 1984.

[Fuji90] H. Fujiwara, "Computacional Complexity of Controllability / Observability Problems for Combinatorial Circuits", *IEEE Transactions on Computer*, 39(6):762-767, Jun 1990.

[Goel81] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", *IEEE Transactions on Computers*, c-30(3):215-222, 1981.

[GJ79] R.M. Garey e D.S. Johnson, "Computers and intractability, a guide to the theory of NP-completeness", *W.H. Freeman and Co.*, 1979.

[GKP88] G. Georgakopoulos, D. Kavvadias e C.H. Papadimitriou. "Probabilistic Satisfiability". *Journal of Complexity*, 4:1-11, 1988.

[GMN84] H. Gallaire, J. Minker e J.M. Nicolas, "Logic and databases: A deductive approach", *ACM Computing Surveys*, 16(2):153-185, 1984.

[GN72] R. S. Garfinkel e G. L. Nemhauser, "Integer Programming", John Wiley & Sons, New York, 1972.

[Hans79] P. Hansen, "Methods of Nonlinear 0-1 Programming", *Annals of Discrete Mathematics*, 5:53-70, 1979.

[Hans84] P. Hansen, "Networks Flows and Nonlinear 0-1 Programming", *Operations Research Proceedings 1983* (Springer, Heidelberg) 253-268, 1984.

[HJ90] P. Hansen e B. Jaumard. "Algorithms for the Maximum Satisfiability Problem". *Computing*, 44:279-303, 1990.

[HJM89] P. Hansen, B. Jaumard e V. Mathon, "Constrained Nonlinear 0-1 programming", *Cahiers du GERARD*, G-89-38, November, 1989.

[HR70] P.L. Hammer e S. Rudeanu, "Boolean methods in operations research and related areas", (Springer, Berlin, New York), 1968.

[HRR63] P.L. Hammer, I. Rosenberg e S. Rudeanu, "On the determination of the minima of pseudo-boolean functions", *Studii si cercetari Matematice*, 14:359-364, 1963.

[HWL83] F. Hayes, D.A. Waterman e D.B. Lenat, "Building expert systems", Reading, Ma., Addison-Wesley, 1983.

[John74] D. S. Johnson, "Approximation algorithms for combinatorial problems", *Journal of Computer and System Sciences*, 9:256-278, 1974

[Knuth75] D. E. Knuth, "The art of computer programming", vol. 3: Sorting and Searching, (Addison-Wesley Reading, MA, 1975, 2nd edition).

[KGV83] S. Kirkpatrick, C.D. Gelatt e M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, 220(4598):671-680, 1983.

[Lieb82] K.J. Liebeherr, "Algorithmic extremal problems in combinatorial optimization", *Journal of Algorithms*, 3:225-244, 1982.

[LB66] E.L. Lawler e M.D. Bell, "A method for solving discrete optimization problems", *Operations Research*, 14:1098-1112, 1966.

[MR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller e E. Teller, "Equation of state calculations by fast computing machines", *Journal Chem. Phys.*, 21:1087-1092, 1953.

[MW68] J.C.T. Mao e B.A. Wallingford, "An extension of Lawler and Bell's method of discrete optimizations with examples from capital budgeting", *Management Science*, 15:51-60, 1968.

[MW69] _____, "Corrections and ments on an extension of Lawler and Bell's method of discrete optimizations with example from capital budgeting", *Management Science*, 15:481, 1969.

[NP+85] T. A. Nguyen, W.A. Perkins, T.J. Laffey e D. Pecora, "Checking an expert systems knowledge bas for consitency and completeness", *IJCAI 85, Arvind Joshi* (ed.), Los Altos, California, 519-524, 1985.

[NW88] G. L. Nemhauser e L. A. Wolsey. "Integer and Combinatorial Optimization", John Wiley, Chichester, UK, 1988.

[Rose74] D. Rose, "On simple characterization of k-trees", Discrete Mathematics, 7:317-322, 1974.

[Roth66] J. P. Roth, "Diagnosis of automata failures: A calculus and a method", *IBM Journal Research Development*, 10:278-291, 1966.

[Taha75] H. A. Taha, "Integer Programming: Theory, Applications and Computations", Academic Press, 1975.