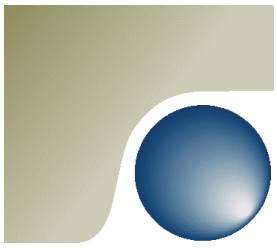


XXXIV Simpósio Brasileiro de Pesquisa Operacional

Carlos Alberto Martinhon

ALGORITMOS RANDÔMICOS EM OTIMIZAÇÃO COMBINATÓRIA

mart@dcc.ic.uff.br

UNIVERSIDADE FEDERAL FLUMINENSE - UFF

Instituto de Computação – IC

ÍNDICE

À minha esposa Priscila.

INTRODUÇÃO

Embora se conheça aplicações envolvendo algoritmos randômicos desde épocas primitivas (Shallit[1992]) os primeiros artigos sobre este assunto datam do final da década de 70 com os trabalhos de Rabin[1976] e Solavay e Strassen[1977] para o problema do reconhecimento de números primos (*Primality Test*). As décadas de 1980 e 1990 testemunharam, a partir de então, um enorme crescimento da área de algoritmos randômicos. Eles emergiram de aplicações voltadas unicamente à teoria dos números e geometria computacional para problemas nas mais diversas áreas de interesse. Uma gama enorme de pesquisadores tem utilizado, cada vez mais, técnicas e ferramentas oriundas de modelos probabilísticos, sejam eles seqüenciais ou paralelos. Como exemplo, pode-se citar aplicações em algoritmos *on-line*, otimização combinatória, criptografia, geometria computacional, teoria dos números, estrutura de dados, processamento paralelo e distribuído entre outras (Karp[1991], Gupta et. al. [1994], Motwani&Raghavan [1995]).

Ao executar um algoritmo determinístico repetidas vezes para uma mesma entrada, obtém-se sempre, uma mesma saída com tempo de processamento sempre constante. Isto não ocorre com os algoritmos randômicos ou probabilísticos onde cada execução poderá produzir, dependendo da aplicação, uma saída diferente baseada em eventos aleatórios. Nestas situações, os tempos de processamento e/ou os resultados obtidos definem uma "função randômica" da entrada. Surpreendentemente, para uma grande quantidade de problemas, a utilização de algoritmos randômicos se constitui na forma mais simples e/ou mais rápida de implementação! Nestes casos, sua utilização implica em uma melhora de desempenho quando comparada a algoritmos puramente determinísticos !

Em um algoritmo randômico, além da entrada de dados associada a um problema, uma fonte de *bits* randômicos é utilizada com o propósito de realizar escolhas aleatoriamente. Desta forma, é inevitável que a análise de desempenho de tais algoritmos utilize conceitos e ferramentas oriundos de teoria de complexidade e probabilidade. Com o propósito de auxiliar o leitor ainda não familiarizado com estes tópicos, faremos, na primeira parte do trabalho, uma breve introdução destes conceitos.

Na parte I do trabalho (Capítulo I), discutiremos, alguns dos modelos de máquina utilizados (máquina RAM determinística e probabilística). Definiremos complexidade local e assintótica, bem como algumas de suas propriedades associadas. Será dada também, uma atenção especial à complexidade de algoritmos recursivos, em particular, ao algoritmo *Quicksort* para o problema de ordenação. Este exemplo será utilizado novamente na segunda parte do trabalho (capítulo III), quando sua versão probabilística for apresentada (*Quicksort randômico*). De maneira geral, o conceito de recorrência é mais amplo e pode ser estendido a modelos probabalísticos (Recorrência Probabilística-Karp [94]).

No Capítulo II, daremos uma atenção especial ao estudo de distribuições de probabilidade discretas. A máquina RAM probabilística ou máquina de Turing probabilística, analogamente aos modelos determinísticos, trabalham apenas com valores inteiros. Assim, solução gerada e tempo de processamento de um algoritmo são quantidades enumeráveis e podem ser representadas convenientemente por variáveis aleatórias discretas.

Na segunda parte do trabalho (Capítulo III), fazemos uma breve introdução aos algoritmos randômicos. Definimos os métodos de *Monte Carlo e Las Vegas* e apresentamos algumas aplicações associadas. No método de Las Vegas, estudaremos o *Quicksort* Randômico e problema da Geração de Orientações Acíclicas em Grafos. No método de Monte Carlo, consideramos várias aplicações de interesse, entre elas, abordamos o problema de Corte Mínimo em Grafos, o problema da determinação de Emparelhamentos Perfeitos em um Grafos, o problema de Multiplicação de Matrizes (versão decisão) entre outras. Finalizamos o capítulo definindo algumas classes de complexidade no modelo probabilístico que são de fundamental importância na classificação de problemas de decisão.

No Capítulo IV, estudamos os algoritmos randômicos aproximativos, técnica que vem sendo cada vez mais utilizada na solução de problemas combinatórios NP-Árduos. Começamos tratando dos algoritmos aproximativos determinísticos, apresentando seus principais conceitos e definições. Em seguida, aplicamos os algoritmos randômicos aproximativos ao problema *MAX-SAT* e ao problema Geral de Recobrimento (*General Covering Problem*), mais especificamente, ao problema de Recobrimento de Conjuntos (*Set Covering Problem*). Na abordagem aqui apresentada, formula-se inicialmente uma relaxação linear do problema. Estas soluções (relaxadas) irão definir probabilidades para algum procedimento randômico, utilizado logo a seguir. Trata-se, na verdade, de uma aplicação do *Método Probabilístico*, desenvolvido inicialmente por Erdös&Spencer[1974]. Nele, o objetivo principal será garantir a existência de objetos combinatórios. Em seguida, utilizando-se o método das probabilidades condicionais, pode-se construir uma versão determinística para o problema original (*Derandomization techniques*). Em nosso trabalho, as versões determinísticas para os problemas MAX-SAT e Recobrimento serão implementadas através do método das Expectâncias Condicionais.

Vários tópicos interessantes sobre o assunto, ainda não foram considerados nesta primeira versão do trabalho. Entre eles, podemos citar a utilização de algoritmos randômicos em Programação Linear (Motwani e Raghavan[1995]), a utilização de Programação de Semidefinida na relaxação de problemas combinatórios (Goemans&Willianson [1994/95]), Amostra Randômica (*Random Sampling*) em Grafos (Karger[1995]), percursos aleatórios em grafos, entre outros.

Os modelos probabilísticos de computador podem ser vistos como uma extensão dos modelos puramente determinísticos. Sua utilização e aplicação na resolução de problemas algorítmicos constitui, sem dúvida alguma, um tema fascinante de pesquisa e que vem despertando, cada vez mais, o interesse da comunidade científica.

PARTE I

CONCEITOS BÁSICOS

E

FERRAMENTAS

Capítulo I

Complexidade de Algoritmos

"Deus fez os números inteiros,
todo o resto é obra do homem"
Leopold Kronecker

I.1 - INTRODUÇÃO:

Informalmente falando, um algoritmo se caracteriza, essencialmente, por uma seqüência finita de "passos elementares" voltados para a resolução de um determinado problema. Este algoritmo pode ser visto como uma função *f(.)* onde *S = f(E),* sendo *E* uma entrada qualquer do problema, e *S*, uma solução deste problema. Segundo Knuth, a palavra "algoritmo" é derivada do nome "*al-Khowârizmî*", um matemático persa do século IX. Apesar do primeiro computador ter sido construído na década de 1940, os algoritmos já existiam há muitos anos e eram dedicados, primordialmente, à solução de problemas aritméticos.

Pode-se identificar, entre os modelos de computador atuais, *três* tipos de algoritmos que se diferenciam, basicamente, na maneira como resolvem e tratam seus problemas: os algoritmos determinísticos, randômicos ou probabilísticos e os algoritmos não-determinísticos.

O tempo de execução de um algoritmo determinístico para uma mesma entrada *E* é sempre fixo, ou seja, sucessivas repetições do algoritmo aplicadas a *E* resultam sempre, em uma mesma saída sem que ocorra variação de seu tempo de processamento. Existem situações entretanto, onde a saída e/ou o tempo de processamento poderão ser distintos a cada repetição do algoritmo. Trata-se dos algoritmos randômicos (ou probabilísticos) estudados mais adiante no capítulo III. Nos algoritmos randômicos um número *finito* processadores geram valores aleatoriamente, permitindo, desta forma, que diferentes respostas sejam geradas a cada repetição. Nos algoritmos não-determinísticos, assume-se que um número *infinito* de processadores seja utilizado, cada um deles, gerando valores aleatoriamente. Como abordado mais adiante (seção I.9), esse conceito abstrato de algoritmo será importante na classificação dos problemas de decisão quanto a seu grau de dificuldade. Salvo indicação contrária, consideraremos apenas algoritmos determinísticos.

Suponha que sejam dados um problema π e um algoritmo determinístico que o resolva após um número finito de passos. Normalmente, deseja-se responder questões relativas à eficiência deste algoritmo. Entretanto, o que significa dizer que um algoritmo *é* ou *não* eficiente? Ou ainda, quais serão os parâmetros observados na avaliação de seu desempenho? Pode-se, evidentemente, considerar o tempo e o espaço (memória) exigidos no processamento como importantes fatores na análise de eficiência desse algoritmo.

O processo mais utilizado na avaliação de algoritmos até meados da década de 60 foi a chamada "análise de comportamento médio", que consiste fundamentalmente, em avaliar o tempo de CPU necessário na resolução de determinadas instâncias de um problema. O comportamento médio é extremamente útil, pois fornece informações precisas sobre o desempenho do algoritmo

para aquelas instâncias em particular. Esta análise entretanto se torna bastante inadequada se for considerada uma entrada qualquer para o problema. Outras dificuldades são também incorporadas, além de depender (na maioria dos casos) de uma infinidade de dados de entrada, o tempo de processamento dependerá evidentemente da máquina utilizada, do código gerado pelo compilador, e ainda, de como evolui o tempo de processamento quando instâncias cada vez maiores para o problema forem consideradas. Pode-se constatar por exemplo, que a taxa de variação entre o tempo e o tamanho de uma entrada qualquer será fator determinante no desempenho de um dado algoritmo.

Neste capítulo serão apresentadas algumas ferramentas utilizadas na análise de desempenho (ou complexidade) de um algoritmo e que visam contornar as dificuldades expostas acima. A idéia será buscar uma medida de complexidade (função de complexidade) que independa do computador e da natureza dos dados de entrada. Como discutido posteriormente, essa função será definida no tamanho de uma instância qualquer e retornará valores no tempo. Mais formalmente, espera-se obter uma função de complexidade (de tempo) $T:tam(\pi)\rightarrow Z^+$ onde $tam(\pi)$ representa o tamanho de uma instância I qualquer (definida na seção I.4) e Z^+ representa o número total de unidades de tempo.

Analogamente, pode-se fazer uma analise da complexidade de espaço de um determinado algoritmo. Nestes casos, deseja-se obter uma função $E:tam(\pi)\rightarrow Z^+$, onde $tam(\pi)$ representa o tamanho do problema e Z^+ o número de posições de memória exigida pela máquina na execução do algoritmo.

I.2 - ALGORITMOS E PERFORMANCE

Em função do grande desenvolvimento no *hardware* dos computadores modernos digitais somos levados, mesmo que inconscientemente, a ignorar a importância dada à performance de algoritmos. Na verdade, com a evolução tecnológica e a possibilidade de se trabalhar com problemas cada vez maiores a preocupação com o desempenho dos algoritmos se torna fundamental. Nesta seção, são apresentados alguns exemplos onde a taxa de variação entre o tempo de processamento e o tamanho do problema são considerados. Pode-se constatar que, em um casos, o tempo de processamento cresce exponencialmente enquanto o tamanho do problema cresce apenas linearmente. Nestas situações, o ganho obtido com o equipamento pode se tornar irrelevante.

Suponha que 5 algoritmos distintos sejam elaborados para resolução de um determinado problema. A complexidade de cada algoritmo é apresentada na Figura I.1. O parâmetro n representa o tamanho do problema. As funções de complexidade associadas (coluna 2), indicarão o número de unidades de tempo utilizado para se processar uma entrada de tamanho n. Deseja-se responder, qual o tamanho máximo do problema (para cada algoritmo) de forma a resolvê-lo em um intervalo de tempo pré-determinado (p. ex., *1seg, 1min* o*u 1 hora*). Suponha que um *milisegundo* seja a unidade de tempo utilizada.

Note que, no algoritmo exponencial, apenas problemas pequenos são considerados dentro da faixa de tempo estipulada. Os demais algoritmos (Figura I.1) são *polinomiais*. Cabe notar que, embora a função de complexidade de tempo $T(n) = n\log n$ não seja polinomial (algoritmo A2), ela também será considerada "polinomial" já que pode ser limitada superiormente por um polinômio de grau $k \geq 2$.

Como discutido mais adiante, a grande maioria dos algoritmos considerados são executados em tempo polinomial ou exponencial no tamanho da entrada. Há funções entretanto, onde a taxa de crescimento é superior à polinomial mas inferior à exponencial (p. ex., $f(n)=n^{\log n}$). Pode-se ter também funções onde a taxa de crescimento seja superior à exponencial (p. ex., $f(n) = O(n!)$).

ALGORITMOS	COMPLEXID. DE TEMPO	TAMANHO MÁXIMO DO PROBLEMA		
		1 Seg.	1 Min.	1 Hora
A1	n	1000	$6x10^4$	$3{,}6x10^6$
A2	$nlogn$	140	4893	$2x10^5$
A3	n^2	31	244	1897
A4	n^3	10	39	153
A5	2^n	9	15	21

Figura I.1: Tamanho máximo de um problema

Considere agora uma geração futura de computadores dez vezes mais rápido que os atuais. O quadro da Figura I.2 mostra como cresce o tamanho desses problemas à medida que a velocidade de processamento é aumentada.

Observe que no algoritmo A_5 apenas um acréscimo de 3,3 unidades no tamanho do problema para um computador 10 vezes mais rápido! Esses dados nos mostram a importância que a análise de complexidade exerce sobre o tempo de processamento. Deve-se lembrar entretanto que, algoritmos exponenciais quando aplicados a problemas pequenos podem ter um desempenho melhor que um algoritmo polinomial. Suponha por exemplo *dois* algoritmos de complexidades 10^5n e 2^n respectivamente. Para $n \leq 21$, o algoritmo exponencial terá um desempenho superior ao algoritmo polinomial. Verifique!

ALGORITMOS	COMPLEXID. DE TEMPO	Tam. Máx. Comp. Atuais	Tam. Máx. Comp. Futuros
A1	n	S_1	$10.S_1$
A2	$nlogn$	S_2	$\approx S_2$
A3	n^2	S_3	$3{,}16.S_3$
A4	n^3	S_4	$2{,}15.S_4$
A5	2^n	S_5	$S_5+3{,}3$

Figura I.2: Tamanho máximo de um problema

Outros fatores devem ser discutidos na avaliação da complexidade como, por exemplo, o desempenho médio do algoritmo (complexidade de caso médio). Um algoritmo de complexidade exponencial nem sempre executará um número exponencial de passos para qualquer entrada. Um exemplo clássico é o algoritmo simplex para o problema de programação linear (para maiores detalhes vide Bazaraa et al.[1990]) . O algoritmo simplex tem em média, processamento polinomial, embora possua complexidade teórica exponencial.

Antes de passar a uma discussão mais formal sobre o tamanho de um problema e das funções complexidade é fundamental definir, mais precisamente, um modelo teórico de computador. Nele, serão abstraídas as principais características de uma máquina mais sofisticada.

I.3 - MÁQUINA RAM (Random Access Machine)

Como avaliar o tempo de execução de um algoritmo, sem executá-lo propriamente em uma determinada máquina? Antes de contar o número total de passos realizados por um algoritmo, deve-se, inicialmente, observar que esses passos são diferentes entre si. O tempo de uma adição por exemplo, será diferente do tempo gasto em uma multiplicação, o tempo de uma divisão será

diferente de uma comparação e assim por diante. Portanto, é importante que se considere o tempo de cada instrução separadamente somando-as apenas ao final do processo. Deve-se lembrar ainda, que o tempo de processamento de uma determinada instrução não será o mesmo em máquinas diferentes. Portanto, é fundamental que se construa um modelo teórico de computador e que seja capaz de gerar informações aproximadas sobre o tempo de processamento independentemente do computador utilizado.

O modelo *RAM* (*Random Access Machine*) é um modelo hipotético de máquina constituído de duas fitas (para entrada e saída de dados respectivamente), uma unidade de controle e processamento e uma memória (Figura I.3). A natureza exata das instruções não é relevante, sendo semelhantes às encontradas em um computador real. Haverá operações de entrada e saída, transferência de informação entre memória e registradores, endereçamento indireto, operações aritméticas e desvios. Cada registrador ou posição de memória poderá armazenar e acessar valores inteiros como uma unidade, sem ter acesso à representação do número.

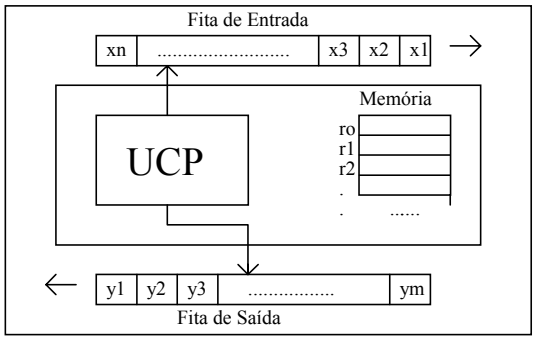


Figura I.3: Modelo de uma máquina RAM

As operações aritméticas permitidas são +, -, x, /. Um algoritmo poderá ainda comparar dois valores e avaliar a raiz quadrada de um inteiro positivo.

Obviamente outros modelos teóricos de computador podem ser utilizados como, por exemplo, a máquina de Turing. Entretanto, uma função de complexidade obtida em qualquer outro modelo de computador, deverá ser expressa equivalentemente em uma máquina RAM (Hopcroft[1979]). Assim, funções polinomiais/exponenciais de tempo nos demais modelos, deverão ser polinomiais/exponenciais no modelo RAM.

O modelo RAM pode ainda ser subdividido em *dois* tipos de máquina. No primeiro, máquina RAM com custo uniforme, cada instrução é executada em uma unidade de tempo. Este modelo é considerado excessivamente poderoso já que não pode ser simulado polinomialmente por uma máquina de Turing. Isto ocorre pois, com a operação de multiplicação, inteiros extremamente grandes podem ser gerados (Motwani&Raghavan[1995]). Eliminando-se as demais operações aritméticas, além da soma e subtração, o modelo RAM se torna polinomialmente equivalente à máquina de Turing. No segundo modelo, máquina RAM com custo logarítmico, cada instrução é processada em um tempo proporcional ao logaritmo de seus operandos. Neste caso, o novo modelo incluindo todas as operações aritméticas descritas acima, será polinomialmente equivalente à máquina de Turing (Motwani&Raghavan[1995]).

Por razões de simplicidade e clareza, utiliza-se normalmente a máquina RAM com custo uniforme. Assim, as fitas de entrada e saída serão formadas por uma seqüência de células, cada uma delas podendo armazenar um número inteiro de tamanho arbitrário. Além disso, não haverá limite para o número de células na memória. Esta abstração será possível quando:

- o tamanho do problema for suficientemente pequeno para ser armazenado na memória principal,
- os inteiros utilizados nos cálculos forem suficientemente pequenos para serem armazenados em uma única palavra do computador, em outras palavras, o tamanho dos operandos será limitado polinomialmente no tamanho da entrada,

Adicionalmente, a máquina RAM poderá gerar, em um único passo, valores aleatórios distribuídos uniformemente em um conjunto limitado no tamanho do problema. (máquina RAM probabilística). Para maiores detalhes sobre este assunto vide Papadimitriou[1994].

Formulado nosso modelo teórico de máquina passemos agora para a definição de tamanho de um problema:

I.4 - TAMANHO DE UM PROBLEMA

Considere π um problema qualquer, ou seja, um enunciado que indique quais propriedades a solução deverá satisfazer e uma descrição formalizada de todos os seus parâmetros (dados de entrada). Fazendo-se uma associação de valores (numéricos ou não) a esses parâmetros define-se uma *instância* de π. Considere o seguinte exemplo:

Exemplo I.1: (Problema da Mochila)

Neste problema, o objetivo será encher uma mochila de capacidade M com no máximo n objetos distintos. Sabendo-se que cada objeto tem um valor diferente, como encher a mochila (sem ultrapassar sua capacidade máxima) maximizando o valor dos objetos presentes em seu interior? Mais formalmente, tem-se o problema:

$$\textit{maximizar} \sum_{j=1}^{n} c_j x_j$$

$$\textit{sujeito a}: \begin{cases} \sum_{j=1}^{n} p_j x_j \le M \\ x_j = 0 \ \textit{ou}\ 1 \qquad \textit{onde} \quad j = 1,..,n \end{cases}$$

onde n, b, c_j e a_j são parâmetros inteiros positivos e x_j é variável de decisão. Os parâmetros são especificados abaixo:

c_j - custo (valor) da peça j
M - capacidade total da mochila (peso)
n - número total de objetos
p_j - peso da peça j

Assim, uma instância I para o problema da mochila será definida por uma $(2n+1)$-upla de inteiros positivos $(c_1,...,c_n,p_1,...,p_n,M)$ onde $p_j \le M$ e $j=1,..,n$. •

O conceito de tamanho de problema, será fundamental na definição de função de complexidade. Assim, dado um problema π e uma instância I qualquer associada, o *tamanho* desse problema (denotado por $tam(\pi)$), será definido como um número inteiro positivo que representa a quantidade dos dados de entrada presentes em I.

A noção exata de quantidade dependerá do problema considerado. Em algumas situações, deseja-se determinar o número total de elementos presentes em uma dada entrada sem se preocupar com os valores assumidos por cada elemento dessa instância. Em outras, o valor de cada elemento

a) Complexidade de Pior Caso: $T_W = \max_{1 \le k \le m} \{T(E_k)\}$

b) Complexidade de Melhor Caso: $T_B = \min_{1 \le k \le m} \{T(E_k)\}$

c) Complexidade de Caso Médio: $T_M = \sum_{1 \le k \le m} \Pr(E_k).T(E_k)$

A complexidade de melhor caso representa, evidentemente, a situação ideal dos dados de entrada do algoritmo não sendo considerada, portanto, uma medida apropriada de complexidade na maioria dos casos.

A complexidade de caso médio como definido acima tem obviamente grande interesse prático. Entretanto, para vários problemas, sua estimativa se torna bastante trabalhosa já que é importante que se defina, de antemão, uma distribuição estatística associada aos dados de entrada. Além disso, vários problemas admitem uma quantidade infinita (enumerável ou não) de dados de entrada e a determinação dos tempos de processamento a elas associadas se torna, por vezes, impraticável.

Na complexidade de pior caso, uma cota superior para o tempo de processamento será determinada. Será também de grande interesse prático e poderá ser calculada mais facilmente que a complexidade de caso médio.

Note que, escolhendo um dos três itens acima, será possível determinar o tempo de processamento em função do tamanho do problema.

Notação: Para simplificar a notação acima, salvo indicação contrária, assumi-se que $T=T_W$, $T=T_B$ ou $T=T_M$ na análise de complexidade. •

Antes de tentar determinar uma expressão para a função de tempo *T(.)*, deve-se avaliar, individualmente, o tempo de cada instrução σ_j do algoritmo *A*. Supondo $t(\sigma_j)$, o número de unidades de tempo gasto na execução de σ, o tempo total na execução do algoritmo será dado por:

$$T = \sum_j k_j . t(\sigma_j) \qquad \text{(I)}$$

onde:

σ_j - *j-ésima* instrução do algoritmo (máq. *RAM*),
k_j - número de execuções de σ_j
$t(\sigma_j)$ - tempo de execução de σ_j

Note que o número de execuções k_j da instrução σ_j será uma função do tamanho do problema (ou seja, $k_j = k_j\ (tam(\pi))$). Na complexidade de pior caso, k_j irá representar o número máximo de execuções (passos) de σ_j. É fácil ver portanto que o tempo *T(.)* será, de fato, função do tamanho do problema. Assim, a complexidade de tempo, será o maior número de passos possível (no pior caso) para execução completa de *A*.

No critério de custo uniforme cada instrução RAM requer uma unidade de tempo e cada registro requer uma unidade de espaço. Portanto, a função de tempo (I) pode ser rescrita de maneira mais simples como abaixo:

$$T = \sum_{j=1}^{q} k_j , \qquad \text{(II)}$$

onde *q* é o número total de instruções do algoritmo.

Será que a simplificação introduzida acima (onde $t(\sigma_j) = 1, \forall\ \sigma_j$) acarretará uma grande distorção no tempo de processamento? A proposição I.1 abaixo (deixada como exercício) mostra que esta distorção será de apenas uma constante.

Proposição I.1: Sejam $\sigma_1,\sigma_2,...,\sigma_m$ instruções de um algoritmo *A* com seus respectivos tempos de processamento $t(\sigma_1), t(\sigma_2),....,t(\sigma_m)$. Seja T_0, o tempo de processamento calculado com a utilização da função *(I)* acima. Fazendo-se $t(\sigma_1)=t(\sigma_2)=....=t(\sigma_m)=1$ obtêm-se um novo tempo de processamento *T* onde $T=c.T_0$ e *c* é constante. •

A função de complexidade *T(.)* (representada por (I) ou (II)) será chamada de função de *complexidade local de tempo*. Analogamente, a *complexidade local de espaço* de um algoritmo indica a quantidade total de memória exigida por um algoritmo durante sua execução. No critério de Custo Uniforme essa memória irá corresponder ao número total de registros (células) utilizadas. Lembre-se que, neste caso, cada registro armazena um inteiro de tamanho arbitrário. Pode-se representar a complexidade de espaço por uma função $E:tam(P)\rightarrow Z^+$ onde *tam(P)* representa o tamanho de nosso problema e Z^+, o número total de unidades de espaço utilizados.

O exemplo I.5 a seguir, trata das complexidades locais de tempo e espaço no pior caso.

Exemplo I.5: (Pior Caso)

Considere $p(x) = a_nx^n + a_{n-1}x^{n-1}+...+a_0$ um polinômio qualquer de grau *n* com coeficientes inteiros. Dado $x_0 \in Z^+$, deseja-se calcular simplesmente o valor $p(x_0)$.

Considere inicialmente o seguinte algoritmo:

Algoritmo I.1: Cálculo de $p(x_0)$:
Início
leia $(n, a_n, a_{n-1},..,a_0 , x_0)$;
$p \leftarrow a_0$;
para $j:=1$ **até** n **faça**
$p \leftarrow p + a_j x_0^j$;
fim;
imprima (p);
fim.

Figura I.4: Cálculo de $p(x_0)$

No cálculo de x^j (potenciação), *j-1* multiplicações são executadas. Tem-se ainda, na estrutura de repetição mais 4 operações a saber: incremento de *j*, comparação com *n*, adição e multiplicação. Dessa forma o "*loop*" interno será executado *n* vezes e a cada execução serão feitas *(3+j)* operações. No final das repetições quando $j=n$ (no *para... faça...*), mais um incremento de *j* e uma comparação com *n* serão realizadas. Assim o tempo total será:

$$T(n) = 2 + \sum_{j=1}^{n}(3 + j) = n\left(\frac{n+1}{2}\right) + 3n + 2 = \frac{n^2 + 7n + 4}{2}$$

Na complexidade de espaço são necessários *n+1* registros para armazenar as constantes a_n, $a_{n-1},..,a_0$, *três* registros para se armazenar n,x_0 *e p*, e um acumulador. Necessita-se portanto, de um total de *n+5* posições de memória. Assim, $E(n)=n+5$.

Pode-se obter uma melhora na complexidade local de tempo do algoritmo utilizando-se o método de Horner. Neste método, o polinômio *p(.)* será rescrito na forma de um "ninho" de

binômios. Abaixo é apresentado um quadro para o cálculo de $p(x_0)$ considerando-se um polinômio de grau *3:*

x_0	a_3	a_2	a_1	a_0
	$\downarrow$	a_3x_0	$a_2x_0 + a_3x_0^2$	$a_1x_0 + a_2x_0^2 + a_3x_0^3$
	a_3	$a_2 + a_3x_0$	$a_1 + a_2x_0 + a_3x_0^2$	$a_0 + a_1x_0 + a_2x_0^2 + a_3x_0^3$

Figura I.5: Tabela de Horner

Tem-se portanto: $p(x_0) = ((a_3x_0+a_2)x_0+a_1)x_0+a_0$. O algoritmo apresentado abaixo resolve o problema para um polinômio de grau *n*.

Algoritmo I.2: Método de Horner;
Início
leia $(n, a_n, a_{n-1},..,a_0, x_0)$;
$p \leftarrow a_n$;
para $i:=n-1$ **até** 0 **faça**
$p \leftarrow a_i + p.x_0$;
fim;
imprima (p);
fim.

Figura I.6: Cálculo de p(x_0) através do método de Horner

Observe agora que *4* operações serão realizadas dentro da estrutura de repetição. Note ainda que ao final do *loop* serão executadas mais 2 operações (decremento e comparação). Logo, a complexidade local de tempo será igual a $T(n)=4n+2$.

Analogamente ao algoritmo anterior tem-se um comportamento linear para a complexidade local de espaço do algoritmo I.2 acima (verifique).

Na análise de complexidade, considera-se normalmente a complexidade de tempo de um determinado algoritmo. As conclusões obtidas sobre a complexidade de espaço são análogas e seguem diretamente. O próximo exemplo ilustra uma aplicação da complexidade de caso médio:

Exemplo I.6: (Complexidade de caso médio)

Suponha que se deseje encontrar o tempo médio de busca de um elemento *x* em uma lista $A=\{a_1,a_2,...,a_n\}$ com *n* elementos. Considere o algoritmo I.3 a seguir:

Algoritmo I.3: Busca;
Início
leia (A,x);
$a_{n+1} \leftarrow x$;
$i \leftarrow 1$;
enquanto $a_i \neq x$ **faça**
$i \leftarrow i+1$;
imprime solução (A, i);
fim.

Figura I.7: Busca elemento em uma lista desordenada

Note que se *x* não pertence a lista então $x = a_{n+1}$. Como discutido acima, a função de complexidade de caso médio será:

$$T_M = \sum_{k=1}^{m} \Pr(E_k).T(E_k)$$

onde *m* é o número total de entradas de tamanho *n.*

Apesar de se ter um número infinito de entradas, será possível classificá-las em *n+1* classes distintas a saber:

$$E_1 = \{A / a_1 = x\}$$
$$E_2 = \{A / a_2 = x\}$$
$$\ldots\ldots\ldots\ldots$$
$$E_n = \{A / a_n = x\}$$
$$E_{n+1} = \{A / x \notin A\}$$

Note que E_i para $i=1,2,..,n$ representa o conjunto de todas as listas A tal que $a_i = x$ e E_{n+1}, o conjunto de todas as listas A tal que $x \notin A$. Observe portanto que na função de complexidade de caso médio tem-se $m=n+1$. Deve-se determinar agora as probabilidades e os tempos de execução de cada uma das $n+1$ entradas. Suponha por exemplo que $Pr(E_i)=Pr(E_j)$ para $i=1,..,n$ e $j=1,..,n$ (distribuição uniforme). Se q é a probabilidade de se ter $x \in A$ então:

$$\Pr(E_i) = q/n \text{ , para } i = 1,..,n$$
$$\Pr(E_{n+1}) = 1 - q$$

Analisando-se o algoritmo de busca acima conclui-se diretamente que $T(E_i) = i$ será o número de comparações para cada entrada E_i. Substituindo $Pr(E_i)$ e $T(E_i)$ na expressão da complexidade média tem-se:

$$T = T_M = \sum_{i=1}^{n+1} \Pr(E_i).t(E_i) = \sum_{i=1}^{n} \frac{q}{n}.i + (1-q).(n+1) = (n+1).\frac{(2-q)}{2}$$

Observe que se $q=1$ tem-se em média $(n+1)/2 \approx n/2$ comparações! •

A maior dificuldade existente na análise do comportamento médio de um algoritmo é definir a distribuição da entrada de dados associada ao problema. Freqüentemente, assume-se que as instâncias sejam igualmente prováveis, entretanto, a determinação de uma amostra representativa da entrada pode, em algumas casos, não ser muito precisa. Observe ainda que, no cálculo de cada $T(E_i)$, contabilizou-se apenas o número de comparações realizadas ($T(E_i) = i$). Como discutido a seguir, este tipo de abordagem irá simplificar enormemente o cálculo de complexidade. A complexidade local de tempo (ou espaço) será substituída pela complexidade assintótica.

I.6 - COMPLEXIDADE ASSINTÓTICA

A determinação de uma expressão exata para as complexidades locais de tempo ou espaço de um determinado algoritmo são freqüentemente desgastantes e desnecessárias. Em um modelo teórico de máquina, a complexidade local já é uma aproximação grosseira de uma máquina real, não sendo necessário portanto, um grande esforço analítico para obtê-la. Por que não observar apenas o comportamento assintótico das funções de complexidade quando o tamanho do problema se torna arbitrariamente grande? Neste caso, será suficiente provar se as complexidades obtidas tem um comportamento linear, quadrático, exponencial, etc.

Na determinação da complexidade assintótica faz-se, freqüentemente, uma simplificação das constantes que aparecem na expressão da complexidade local. Neste caso, pretende-se

determinar apenas a taxa de variação do tempo de processamento em relação ao tamanho do problema. Em outras palavras deseja-se obter a ordem de grandeza da função de complexidade local. Abaixo é apresentada uma definição mais formal de complexidade assintótica:

Definição I.1: (Complexidade Assintótica)

Uma função de complexidade local *T(n)* será de ordem *O(f(n))* se, e somente se, existirem constantes positivas *c* e n_0 tal que $0 \leq T(n) \leq c.f(n), \ \forall n \geq n_0$. •

Note que *c.f(n)* define um *limite superior* para *T(n)*. Graficamente, tem-se a seguinte situação representada na figura I.8.

Pode-se representar por *O(f(n))*, o conjunto de todas as funções *T(n)* tais que $0 \leq T(n) \leq c.f(n), \ \forall n \geq n_0$.. Neste caso, pode-se dizer simplesmente que $T(n) \in O(f(n))$[1].

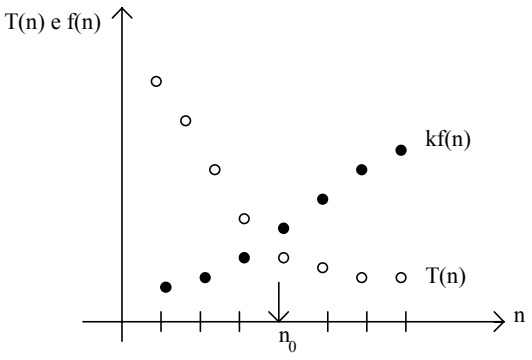


Figura I.8: Complexidade Assintótica

Portanto, para mostrar que uma dada função *T(n)* pertence a *O(f(n))* deve-se exibir ou garantir a existência de duas constantes *c* e n_0 tal que a definição descrita acima se verifique. Como determinar entretanto as constantes *c* e n_0? Para responder a essa pergunta considere inicialmente a seguinte definição formal de limite:

Definição I.2: (Limite):

Diz-se que $\lim_{n\to\infty} h(n) = k$ se, e somente se, $\forall \ \varepsilon > 0, \ \exists \ n_o \ tal \ que \ \|h(n) - k\| \leq \varepsilon, \ \forall \ n \geq n_0$. •

Assim, se $lim_{n\to\infty} T(n)/f(n) = k < \infty$ tem-se, da definição de limite, que: $\forall \ \varepsilon > 0, \ \exists \ n_0$ tal que $||T(n)/f(n) - k|| \leq \varepsilon, \ \forall \ n \geq n_0$. Ou seja, tem-se $k - \varepsilon \leq T(n)/f(n) \leq k + \varepsilon, \ \forall \ n \geq n_0$. Tomando esta última desigualdade e fazendo $c = k + \varepsilon$, conclui-se que: $\exists$ *c* e n_0 tal que $T(n) \leq c.f(n), \ \forall \ n \geq n_0$, ou seja *T(n) é O(f(n))* (ou simplesmente $T(n) \in O(f(n))$). Note que a existência de n_0 na definição de complexidade assintótica é garantida pela definição formal de limite. Este tipo de abordagem é interessante, já que ele nos exige, da talvez árdua tarefa de determinar n_0 explicitamente.

Considere o seguinte exemplo:

Exemplo I.6: Seja $T(n) = 5n^2 + n$. A função *T(n)* é $O(n^2)$? Note que fazendo *c = 6* tem-se $5n^2 + n \leq 6n^2$. Calculando $lim_{n\to\infty} T(n)/f(n)$:

$$\lim_{n\to\infty} \frac{5n^2 + n}{n^2} < 6 < \infty .$$

[1] Alguns autores utilizam a notação: *T(n) = O(f(n))* (vide Cormem et al. [1990]).

Pode-se garantir portanto, da definição de limite, a existência de um inteiro n_0 tal que $5n^2 + n \leq 6n^2,\ \forall\, n \geq n_0$. •

Pode-se mostrar também que $T(n)= 5n^2+n$ é $O(n^3)$ ou $O(n^4)$. Entretanto, estaremos interessados no menor limite superior possível. Para se evitar conflitos dessa natureza, diz-se simplesmente que $T(n)$ é $o(n^3)$ ou $o(n^4)$. Mais formalmente se $T(n)$ é $o(f(n))$ então $lim_{n\to\infty} T(n)/f(n) = 0$.

Exemplo I.7: $T(n) = 3^n$ é $O(2^n)$? Suponha que existam k e n_0 tais que $3^n \leq k.2^n,\ \forall\, n \geq n_0$. Segue que:

$$\left(\frac{3}{2}\right)^n \leq k, \quad \forall\ n \geq n_0$$

Calculando o limite:

$$\lim_{n\to\infty}\left(\frac{3}{2}\right)^n \to +\infty \leq k \qquad \text{(absurdo!)}$$

Chegando portanto a uma contradição. Logo, $T(n)=3^n$ não será $O(2^n)$. Pode-se mostrar entretanto que 2^n é $O(3^n)$, ou mais precisamente, que 2^n é $o(3^n)$. Verifique! •

Pode-se estabelecer, através da notação O(.), uma hierarquia de funções de complexidade. Abaixo são apresentados alguns exemplos:

$$O(1) \subset O(\log n) \subset O(n) \subset O(n\log n) \subset O(n^2) \subset \ldots \subset O(2^n) \subset O(n!) \subset \ldots$$

Da mesma forma, se $T(.)$ representa a complexidade de melhor caso de um algoritmo pode-se definir um *limite inferior* para $T(.)$. O limite inferior será representado por $\Omega(.)$.

Definição I.3: (Limite Inferior)

Seja $T(n)$ a complexidade local (de melhor caso) de um dado algoritmo. A função de complexidade $T(n)$ é $\Omega(f(n))$ se existem constantes c e n_0 *tais que* $T(n) \geq cf(n),\ \forall\, n \geq n_0$. •

Exemplo I.8: Seja $T(n) = 2n^2 + n$ a complexidade de melhor caso de um dado algoritmo. Deseja-se mostrar que $T(n)$ é $\Omega(n^2)$. De fato, bastará fazer k=1 e n_0=1. Assim: $2n^2 + n \geq n^2,\ \forall\, n \geq 1$. Pode-se mostrar também que $T(n)$ é $\Omega(n)$ ou $\Omega(1)$, entretanto, deve-se buscar normalmente, o maior limite inferior possível. Analogamente à notação o(.), pode-se utilizar ω(.) para $\Omega(.)$. Assim, em nosso exemplo, pode-se dizer que $T(n)$ é $\omega(n)$ ou $\omega(1)$. •

Considere agora a seguinte definição:

Definição I.4: (Limite):

Diz-se que $\lim_{n\to\infty} h(n) = +\infty$ se, e somente se, $\forall A > 0, \exists\ n_o tal que\ h(n) > A,\ ,\ \forall\, n \geq n_0$. •

Observe por exemplo que, se $lim_{n\to\infty} T(n)/f(n) = +\infty$ então $T(n)$ é $\omega(f(n))$. Verifique!

Em determinadas situações, um algoritmo de complexidade $T(n)$ será $O(f(n))$ e $\Omega(f(n))$ simultaneamente. Neste caso, $T(n)$ será $\theta(f(n))$. Mais formalmente, tem-se:

$$\theta(g(n)) = \{T(n) : \exists\, c_1, c_2, n_0 > 0 \;\; tais\ que \;\; 0 \le c_1 g(n) \le T(n) \le c_2 g(n), \;\; \forall \;\; n \ge n_0\}$$

Exemplo I.9:

a) Para encontrar o maior elemento em uma lista desordenada de *n* elementos pode-se construir facilmente um algoritmo de complexidade local *T(n)* e ordem *O(n)* e *Ω(n)* respectivamente. Neste caso, tem-se um algoritmo de ordem *θ(n)*. Um algoritmo com essa característica será chamado *algoritmo ótimo.*

b) Para encontrar um elemento *x* em uma lista desordenada de *n* elementos, tem-se obviamente *O(n)* passos no pior caso. No melhor caso entretanto, apenas uma comparação será realizada. Assim, *T(n)* terá limite inferior igual a *Ω(1)*. Neste caso o algoritmo não será ótimo.•

Suponha agora um algoritmo que se divida em duas ou mais partes independentes. A seguir, serão apresentadas algumas regras que nos ajudarão no cálculo de complexidade assintótica de um determinado algoritmo:

Proposição I.2 (Regra das Somas):

Um algoritmo *A* se divide em duas partes independentes A_1 e A_2 onde A_1 de complexidade é $T_1(n)$ é de ordem *O(f(n))* e A_2 de complexidade $T_2(n)$ é de ordem *O(g(n))* então $T(n) = T_1(n)+T_2(n)$ e A será de ordem *O(max(f(n),g(n)))*.

Prova: Sejam C_1, C_2, n_1, n_2 constantes positivas tais que:

$$T_1(n) \le C_1 . f(n), \qquad \forall\, n \ge n_1$$
$$T_2(n) \le C_2 . g(n), \qquad \forall\, n \ge n_2$$

fazendo-se $n_0 = max(n_1, n_2)$ e adicionando as duas equações acima tem-se:

$$T_1(n) + T_2(n) \le C_1 . f(n) + C_2 . g(n) \le (C_1 + C_2) . max(f(n), g(n)), \quad \forall\, n \ge n_0 .$$

Portanto, da definição tem-se que *T(n)* é de ordem *O(max(f(n),g(n)))* •

Proposição I.3 (Regra do Produto): Se um algoritmo A contém dois "aninhamentos" A_1 *e* A_2, de complexidade $T_1(n)$ *e* $T_2(n)$ com ordem *O(f(n))* e *O(g(n))* respectivamente, então *A* será de complexidade local $T(n) = T_1(n).T_2(n)$ e de ordem *O(f(n).g(n))*.

Prova: Sejam [illegible]

a) Se *f(n)* é de ordem *O(g(n))* e *g(n)* é de ordem *O(h(n))* então *f(n)* é de ordem *O(h(n)).*
b) *g(n)* é de ordem *O(f(n))* e *f(n)* é de ordem *O(g(n))* se, e somente se, *O(f(n))=O(g(n)).*
c) *g(n)* é de ordem *Of((n))* e *f(n) não* é de ordem *O(g(n))* se, e somente se, *O(f(n))⊂O(g(n)).*

Prova:
a) Tem-se da definição de complexidade assintótica que:

$$f(n) \le C_1 . g(n), \qquad \forall\, n \ge n_1$$
$$g(n) \le C_2 . h(n), \qquad \forall\, n \ge n_2$$

Fazendo $n_0 = max(n_1, n_2)$ e $C = C_1 . C_2$ *para* $n \ge n_0$ conclui-se que : $f(n) \le C_1 . g(n) \le C . h(n)$ $\forall\, n \ge n_0$. Logo *f(n)* é de ordem *O(h(n)).*

b) **(→)** Considere uma função *T(n)* qualquer pertencente a *O(f(n)).* Como *f(n)* pertence a *O(g(n))* tem-se, da propriedade transitiva que *T(n)* pertence a O(g(n)). Assim, qualquer que seja *T(n)* em *O(f(n))* tem-se *T(n)* pertencente a *O(g(n)),* ou seja, *O(f(n))⊆O(g(n)).* De maneira análoga, pode-se mostrar que *O(f(n))⊇O(g(n)).* Segue então que *O(f(n))=O(g(n)).*
(←) É fácil mostrar que *f(n)* pertence a *O(f(n)).* Como *O(f(n))=O(g(n))* segue que *f(n)* pertence a *O(g(n)).* De maneira análoga pode-se mostrar que *g(n)* pertence a *O(f(n)).*

c) (→) Seja *T(n)* qualquer tal que *T(n) ∈ O(f(n)).* Como *f(n) ∈ O(g(n)),* tem-se do item (a) que *T(n) ∈ O(g(n))* (prop. transitiva). Logo *O(f(n))⊆O(g(n)).* Suponho por absurdo que, *O(f(n)) = O(g(n)).* Como *g(n) ∈ O(g(n))* segue então que *g(n) ∈ O(f(n))* (contradição!). Logo *O(f(n)) ⊂ O(g(n)).*
(←) Temos que *f(n) ∈ O(f(n)).* Como *O(f(n)) ⊂ O(g(n))* então *f(n)∈O(g(n)).* Se *g(n) ∈ O(f(n))* segue da proposição anterior (item b) que *O(g(n)) = O(f(n))* o que é absurdo! Portanto *g(n) ∉ O(f(n)).* •

Proposição I.5: Se $lim_{n\to\infty} f(n)/g(n) = k$ onde $0 < k < \infty$ então *O(f(n)) = O(g(n)).* Se *k = 0* então *O(f(n)) ⊂ O(g(n)).*

Prova: Considere inicialmente o caso onde *k >0.* Da definição de limite tem-se: $\forall\, \varepsilon>0, \exists\, n_0$ tal que, $k-\varepsilon \le f(n)/g(n) \le k+\varepsilon, \forall\, n \ge n_0$.. Ao escolher $k > \varepsilon$ tem-se: *(a)* $f(n) < (k+\varepsilon)\, .g(n), \forall\, n \ge n_0$. e *(b)* $g(n) < (1/(k-\varepsilon)).f(n), \forall\, n \ge n_0$ (verifique). Logo, de *(a)* e *(b)* respectivamente temos *f(n) ∈ O(g(n))* e *g(n) ∈ O(f(n)).* Da proposição anterior, segue que *O(f(n)) = O(g(n)).*

Suponha agora *k = 0.* Da definição de limite tem-se: $\forall\, \varepsilon>0, \exists\, n_0$ tal que, , $-\varepsilon \le f(n)/g(n) \le \varepsilon, \forall\, n \ge n_0$. Assim, da segunda parte da desigualdade: $\forall\, \varepsilon>0, \exists\, n_0$ tal que, $f(n) \le \varepsilon.g(n), \forall\, n \ge n_0$, ou seja, *f(n) ∈ O(g(n)).* Ainda, da segunda parte da desigualdade tem-se que: $\forall\, \varepsilon>0, \exists\, n_0$ tal que $g(n) > (1/\varepsilon).f(n), \forall\, n \ge n_0$. Da definição I.4, $lim_{n\to\infty} f(n)/g(n) = +\infty$. Conclui-se portanto que *g(n) ∉ O(f(n)).* Como *f(n) ∈ O(g(n)),* tem-se da proposição anterior (item 3) que *O(f(n)) ⊂ O(g(n)).*•

DefiniçãoI.5: A definição de complexidade assintótica pode ser estendida facilmente para o caso onde o tamanho do problema seja representado por *dois* ou mais parâmetros. Dada uma função *g(n,m),* representa-se por *O(g(n,m))* o seguinte conjunto de funções:

$$O(g(n,m) = \{ f(n,m) : existem constantes\ positivas\ c, n_0 e\ m_0\ tais\ que\ 0 \le f(n,m) \le cg(n,m)$$
$$\forall\, que\ n \ge n_0 em \ge m_0 \}$$

A definições de $\Omega(g(n,m))$ e $\theta(g(n,m))$ são análogas. Este tipo de representação é bastante útil quando se trabalha com algoritmos em grafos. Neste caso, n representa o número de vértices e m é o número de arestas.

Quando se trabalha com outros modelos de máquina (diferentes da máquina RAM), pode ocorrer que a expressão de complexidade assintótica obtida nestes modelos seja distinta. Ao mudar de uma máquina para outra é importante que se preserve algumas características do modelo original. Considere então a seguinte definição:

Definição I.6: (Funções relacionadas polinomialmente)

Duas funções $f_1: N \rightarrow \Re$ e $f_2: N \rightarrow \Re$ são *relacionadas polinomialmente* se existirem funções $p_1: \Re \rightarrow \Re$ e $p_2: \Re \rightarrow \Re$ tais que $f_1(n) \leq p_1(f_2(n))$ e $f_2(n) \leq p_2(f_1(n)) \ \forall n \geq 0$. •

Como exemplo, considere *duas* funções $f_1(n) \leq 2n^2$ e $f_2(n)$. Fazendo $p_1(x) = 2x$ na primeira desigualdade tem-se que $2n^2 \leq p_1(n^5) = 2n^5$, $\forall n \geq 0$. Na segunda desigualdade, fazendo $p_2(x) = x^5$ chega-se a $n^5 \leq p_2(2n^2) = 8n^6$, $\forall\ n \geq 0$.

Ao se trabalhar com uma máquina de Turing, máquina RAM ou máquina RASP (Aho et. al.[1974]) tem-se funções de complexidade relacionadas polinomialmente. Isto significa dizer que, se uma função de complexidade é polinomial (ou exponencial) em um dado modelo ela será também polinomial (ou exponencial) nos demais modelos. Maiores detalhes sobre este assunto podem ser encontrados em Aho et. al.[1974].

I.7 - COMPLEXIDADE DE ALGORITMOS RECURSIVOS

Um procedimento recursivo consiste, basicamente, na utilização direta ou indireta, de um procedimento dentro do mesmo procedimento que o define. Normalmente, todo processo recursivo exigirá a definição de uma fórmula de recorrência e de uma base da recursão. Esta técnica é amplamente utilizada em métodos do tipo divisão e conquista (para maiores detalhes vide Cormen et al. [1990]).

O exemplo seguinte ilustra uma aplicação da recursividade ao problema de ordenação.

I.7.1 - Algoritmo Quicksort:

Suponha que se queira ordenar uma lista A qualquer com n valores inteiros positivos. Baseado na estratégia de divisão e conquista, Hoare[1962], propôs um algoritmo de ordenação diferente do *Mergesort*. No *Mergesort* dividi-se a lista original A em partes iguais A_1 e A_2 respectivamente. Faz-se então uma ordenação de cada uma das partes seguida de uma concatenação das sub-listas. No *Quicksort* a divisão em duas sub-listas é feita utilizando-se um elemento *chave* (ou pivô) de forma que, no final do processo, as sub-listas ordenadas não precisem mais ser concatenadas! Na sub-lista A_1 são armazenados todos os elementos menores que *chave* enquanto que, na sub-lista A_2 apenas os elementos maiores que *chave*. Após a divisão, repeti-se o processo para A_1 e A_2 separadamente.

O exemplo seguinte ilustra as principais etapas presentes no *Quicksort.* Considere uma lista desordenada A como abaixo:

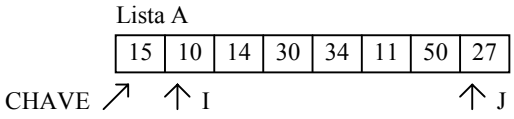


Figura I.9: Lista A desordenada

Note que são criados dois índices *I* e *J* apontando para o segundo e o último elementos respectivamente. Além disso, separa-se o primeiro elemento *A[1]=15* (denominado elemento *chave* ou *pivô*). O objetivo inicial será determinar a posição correta dessa *chave* na lista *A*. Assim, incrementa-se *I* sempre que algum elemento *A[I]*, menor do que *chave* for encontrado. Analogamente, decrementa-se *J* sempre que um elemento *A[J]* maior do que *chave* for encontrado. Essa situação pode ser expressa como na figura seguinte:

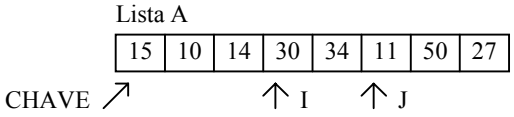


Figura I.10: Atualizações de I e J

Caso se tenha $I < J$ os elementos *A[I]* e *A[J]* são trocados de posição e o processo repetido até que $I \geq J$. A nova situação é apresentada abaixo:

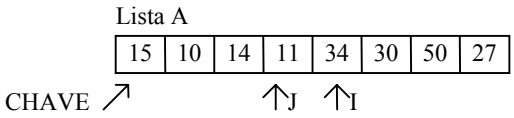


Figura I.11: I > J

Note que a posição correta do elemento *chave* é indicada pelo índice *J*! O próximo passo será trocar a posição do elemento *chave* com *A[J]*. Assim:

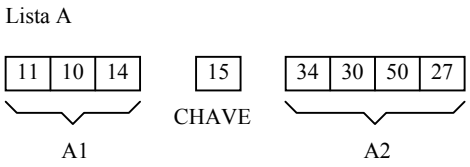


Figura I.12: Divisão da Lista A

Bastará aplicar agora o mesmo processo para as sub-listas A_1 e A_2 separadamente através de chamadas recursivas. Finalmente, tem-se a seguinte situação:

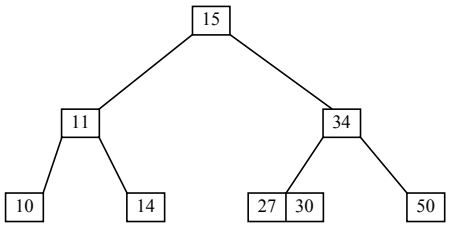


Figura I.13: Lista A ordenada

Observe que as chamadas recursivas são executadas até que *0, 1* ou *2* elementos sobre em cada sub-lista (base da recursão). Caso se tenha apenas *um*, ou nenhum elemento ao final do

processo, nenhuma operação será realizada. Entretanto, se a *A* contiver *2* elementos, apenas uma comparação será realizada. Resumindo, tem-se o seguinte procedimento:

```
Procedimento I.4: Quicksort (inf, sup);
Início
    Se (sup - inf < 2) então
        Se (sup - inf = 1) então                    {temos 2 elementos}
            Se (A[inf]>A[sup]) então
                troca (A[inf], A[sup]);
    senão
        I ← inf;
        J ← sup;
        chave ← A[inf];
        Enquanto J > I faça
            I ← I + 1;
            Enquanto (A[I] < chave) faça
                I ← I + 1;
            Enquanto (A[J] > chave) faça
                J ← J - 1;
            Se (J > I) então
                troca(A[I], A[J]);
        fim Enq;
        troca (A[inf],A[J]);
        Quicksort (inf, I-1);
        Quicksort (J+1, sup);
    fim;
fim.
```

Figura I.14: Algoritmo *Quicksort*

Note no procedimento *Quicksort*, que se o elemento *chave* for o maior de todos os elementos de *A*, o índice *I* será incrementado até *n+1*. Para evitar um erro decorrente de um acesso indevido a esta posição (inexistente na lista *A*), bastará criar um novo elemento fazendo $A(n+1)=\infty$ no programa principal. Verifique!

I.7.1.1 - Complexidade do Quicksort (Pior Caso)

Observando a algoritmo acima pode-se eleger (sem perda de generalidade) a instrução *A[inf]>A[sup]* como operação elementar na base da recursão. Note que *3* situações distintas serão possíveis. Se *sup-inf < 0,* a chamada recursiva irá corresponder a uma sub-lista vazia. Neste caso a operação elementar não será realizada. O mesmo ocorre se *sup-inf =0* (sub-lista com apenas *1* elemento). Segue então que *T(0)=0* e *T(1)=0* respectivamente. Se *sup-inf =1* tem-se uma sub-lista com 2 elementos. Neste caso, *T(2)=1*.

No cálculo da fórmula de recorrência ($n \geq 3$) deve-se atualizar os índices *I* e *J* até que se tenha $I \geq J$. Só então, as chamadas recursivas serão realizadas (etapa de divisão). Observe novamente que *3* etapas devem ser analisadas.

Na etapa de atualização dos índices *I* e *J* conclui-se facilmente que *O(cn)* passos serão executados. Os índices *I* e *J* deverão percorrer a lista de tamanho *n* até se cruzarem.

Nas chamadas recursivas, deve-se estabelecer inicialmente como o vetor original *A* será dividido. Essa divisão entretanto dependerá da natureza dos dados de entrada. Na pior divisão

possível, pode-se construir uma configuração onde *n-1* elementos estarão presentes em uma das sub-listas e 0 elementos na outra. Neste caso, *T(n)* se divide em *T(n-1)* e *T(0)* respectivamente. Isto ocorrerá sempre que o vetor original *A* já estiver ordenado! É fácil ver que esta é a pior divisão possível já que, na chamada de complexidade *T(n-1),* o tamanho do problema original é reduzido em apenas uma unidade! Assim, um maior número de chamadas deverão ser executadas até que se chegue à base da recursão. O mesmo não ocorreria se *A* fosse dividida em duas partes de mesmo tamanho (Verifique!).

Tem-se portanto a seguinte situação:

$$\begin{cases} T(0) = 0,\ T(1) = 0 \ e\ T(2) = 1, & n \le 2 \quad (\textit{base da recursão}) \\ T(n) = cn + T(0) + T(n-1), & n > 2 \quad (\textit{fórmula de recorrênci a}) \end{cases}$$

Desenvolvendo a recorrência acima conclui-se que:

$$\begin{aligned} T(n) &= cn + T(n\text{-}1) \\ &= cn + c(n\text{-}1) + T(n\text{-}2) \\ &= cn + c(n\text{-}1) + c(n\text{-}2) + T(n\text{-}3) \end{aligned}$$

Após *k* passos tem-se: $T(n) = cn + c(n\text{-}1) + c(n\text{-}(k\text{-}1)) + T(n\text{-}k).$ (I)

Espera-se obter, ao final do processo *n-k=2* (base da recursão). Assim, substituindo $n-k=2$ em (I) tem-se:

$$T(n) = c(n + (n\text{-}1) + (n\text{-}2) + \ldots + 4 + 3) + T(2)$$

$$= c\sum_{i=3}^{n} i + T(2) = c\left(\left(\sum_{i=1}^{n} i\right) - 3\right) + 1 = c\left(\frac{n(n+1)}{2} - 3\right) + 1$$

Segue portanto que *T(n)* é de ordem $O(n^2)$. A seção seguinte trata da análise de complexidade média do algoritmo *Quicksort*.

I.7.1.2 - Complexidade do Quicksort (Caso Médio):

Apesar da complexidade teórica de pior caso do *Quicksort* ter se mostrado inferior ao *Mergesort*, testes realizados empiricamente mostram um bom desempenho do *Quicksort* quando comparado ao *Mergesort* (vide Knuth[1973]). Pode-se mostrar em média (para uma entrada gerada aleatóriamente) que o *Quicksort* (assim como o *Mergesort*) é processado em *O(nlogn)* passos.

Assim, suponha inicialmente que a probabilidade do elemento *chave* ser o *i-ésimo* elemento seja *1/n* (distribuição uniforme). Para calcular a análise de performance média deve-se analisar todas as chamadas recursivas possíveis, ou seja, todas as possíveis divisões do vetor original *A*. Utilizando a média ponderada conclui-se que:

$$\frac{1}{n}\sum_{k=1}^{n}\left(T_M(k-1) + T_M(n-k)\right)$$

representa o compartamento médio de todas chamadas recursivas (lembre-se que $T_M(.)$ representa o tempo médio para se processar uma entrada de tamanho *n*). Resumindo todos os casos, tem-se:

$$\begin{cases} T_M(0)=0,\ T_M(1)=0 \ \ e \ \ T_M(2)=1, & n \le 2 \quad (\textit{base da recursão}) \\ T_M(n)=cn+\dfrac{1}{n}\sum_{k=1}^{n}\left(T_M(k-1)+T_M(n-k)\right), & n>2 \quad (\textit{fórmula de recorrênci a}) \end{cases}$$

Desenvolvendo a fórmula de recorrência tem-se que:

$$\begin{aligned} T_M(n) &= cn+\frac{1}{n}.\sum_{k=1}^{n}\left(T_M(k-1)+T_M(n-k)\right) \\ &= cn+\frac{2}{n}.\left(T_M(0)+T_M(1)+..+T_M(n-1)\right) \end{aligned}$$

Multiplicando por *n* ambos os lados tem-se:

$$nT_M(n)=cn^2+2\sum_{i=0}^{n-1}T_M(i) \qquad \text{(I)}$$

Trocando *n* por *n-1* em *(I)*:

$$(n-1)T_M(n-1)=c(n-1)^2+2\sum_{i=0}^{n-2}T_M(i) \qquad \text{(II)}$$

Subtraindo (II) de (I) segue:

$$\begin{aligned} nT_M(n)-(n-1)T_M(n-1) &= c(n^2-(n-1)^2)+2T_M(n-1) \\ &= c(2n-1)+2T_M(n-1) \end{aligned}$$

Fazendo as devidas simplificações na expressão acima chega-se a:

$$nT_M(n)=(n+1)T_M(n-1)+c(2n-1)$$

Multiplicando ambos os lados por $\dfrac{1}{n(n+1)}$:

$$\frac{T_M(n)}{(n+1)}=\frac{T_M(n-1)}{n}+c.\frac{(2n-1)}{n(n+1)}<\frac{T_M(n-1)}{n}+\frac{2c}{n+1}$$

Assim:

$$\begin{aligned} \frac{T_M(n)}{n+1} &< \frac{2c}{n+1}+\frac{T_M(n-2)}{n-1}+\frac{2c}{n} \\ &< \frac{2c}{n+1}+\frac{2c}{n}+\frac{2c}{n-1}+\frac{T_M(n-3)}{n-2} \\ &< \frac{2c}{n+1}+\frac{2c}{n}+\frac{2c}{n-1}+\ldots+\frac{2c}{n-k+2}+\frac{T_M(n-k)}{n-k+1} \end{aligned}$$

Espera-se obter *n-k=2* (base da recursão). Como $T_M(2)=1$ tem-se:

$$\frac{T_M(n)}{n+1}<2c\sum_{j=4}^{n+1}\frac{1}{j}+\frac{T_M(2)}{3}$$

Note que o somatório acima pode ser majorado pela integral de *1/j* (vide Figura I.15). Assim:

$$\frac{T_M(n)}{n+1} < 2c\int_3^{n+1}\frac{1}{j}.dj + \frac{1}{3} = 2c(\ln(n+1) - \ln 3) + \frac{1}{3}$$
$$< 2c.\ln(n+1) - 2c.\ln 3 + \frac{1}{3}$$

Fazendo-se as devidas simplificações chega-se a:

$$T_M(n) < 2cn\ln((n+1)/3) + d$$

onde **c** e **d** são constantes.

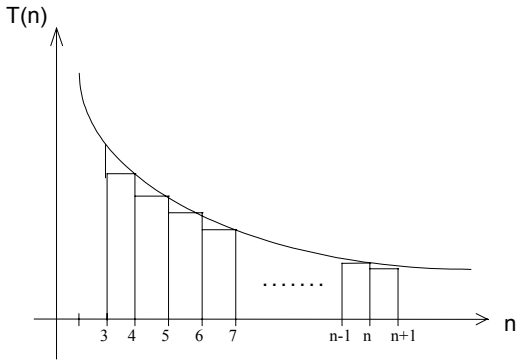


Figura I.15: Integral de 1/j

Observe que, fazendo a mudança de base e simplificando novamente a expressão conclui-se finalmente que $T_M(n)$ é *O(nlog n).* •

I.8 – LIMITE INFERIOR DE PROBLEMAS

Em algumas situações, deseja-se determinar qual a complexidade inerente a um determinado problema (limite inferior). Sua complexidade deverá ser independente dos algoritmos existentes ou não em determinado momento. A obtenção do limite inferior entretanto, pode ser bastante difícil para uma grande quantidade de problemas. O que se faz nestes casos é iniciar a análise do problema com um limite inferior qualquer e ir aumentando (se possível) esse limite até que não seja mais possível aumentá-lo. Teremos chegado, neste caso, a um limite inferior desejado para o problema. Mais formalmente tem-se a seguinte definição:

Definição I.7: (Limite Inferior de um Problema)

Diz-se que *Ω(f(n))* define um limite inferior de um problema π quando qualquer algoritmo que resolver π requerer, pelo menos, *O(f(n))* passos no pior caso. •

Exemplo I.10: Se $\Omega(n^2)$ é um limite inferior de um problema π não poderá existir nenhum algoritmo com complexidade assintótica de pior caso inferior a $O(n^2)$. •

Note neste caso que o limite *Ω(f(n))* é o melhor (maior) possível. Pode-se dizer ainda que um algoritmo ótimo tem menor a complexidade (de pior caso) entre todos os algoritmos possíveis na resolução de um dado problema.

Como comentado anteriormente, o algoritmo *Quicksort* (para uma grande quantidade de entradas) possui desempenho computacional superior quando comparado ao *Mergesort* (sendo ambos $\Omega(nlogn)$). Pode-se mostrar na verdade, que $\Omega(nlogn)$ é o limite inferior para o problema de ordenação (para maiores detalhes vide Horowitz&Sahni[1978]).

I.9 – PROBLEMAS DE DECISÃO E ALGORITMOS NÃO-DETERMINÍSTICOS

Como classificar os problemas resolvidos ou não pelos algoritmos e ferramentas disponíveis em dado momento? Uma primeira classificação diz respeito aos problemas que podem ou não ser resolvidos algoritmicamente (problemas computáveis e não-computáveis). Entretanto, considerando-se apenas os problemas computáveis, como classifica-los em função do grau de dificuldade presente em sua resolução? Note por exemplo que se não resolvemos um problema em tempo polinomial (no tamanho de sua entrada), isto não significa que se possa faze-lo posteriormente! Informalmente falando, como classificar tais problemas independentemente dos algoritmos existentes ou não em dado momento? Esta pergunta começou a ser respondida nos primórdios de 1970 quando uma nova área teve origem na ciência da computação. A teoria de complexidade computacional, buscava, a partir de então, categorizar e classificar problemas computáveis e assim melhor avaliar a qualidade dos algoritmos atualmente disponíveis. Problemas com limite inferior exponencial por exemplo podem ser classificados como *intratáveis,* já que, comprovadamente, será impossível a obtenção de algoritmos (nos modelos de máquina tradicionais) que nos dêem uma solução exata em tempo polinomial. Problemas que possam ser resolvidos por algoritmos polinomiais no tamanho de sua entrada serão chamados de "tratáveis" e pertencerão à classe *P* dos problemas computáveis (Garey&Jonhson[1979]).

Apesar do incansável esforço de uma enorme quantidade de pesquisadores nas mais diversas áreas de atuação, alguns problemas computáveis permanecem sem uma resolução satisfatória. Décadas de trabalho envolvendo abordagens diferenciadas não demonstraram ser suficientes na resolução de determinados problemas em um tempo de processamento aceitável (tempo polinomial). Esse grande número tentativas frustradas levaram a maioria dos pesquisadores a acreditar que tais problemas são inerentemente difíceis e que não poderão ser resolvidos eficientemente. A teoria da NP-completude (Garey&Jonhson[1979]) resolve em parte tais problemas. Apesar dela não provar se tais problemas podem ou não ser resolvidos em tempo polinomial, ela garante que problemas NP-completo se equivalem quanto a seu grau de dificuldade, no sentido que, se um problema desta classe puder ser resolvido em tempo polinomial todos os demais problemas também serão! Esse resultado surpreendente garante que cada um dos problemas desta classe será tão difícil quanto os demais. Neste sentido pode-se dizer que tais problemas são computacionalmente equivalentes. Fica a dúvida entretanto, se existem ou não algoritmos polinomiais para esses problemas.

Apesar dos aspectos negativos levantados acima, a teoria da NP-completude tem aspectos positivos que merecem ser ressaltados. O fato de sabermos a *priori* que um determinado problema é NP-Completo poderá obviamente refletir favoravelmente na estratégia de resolução a ser adotada. Aqueles problemas que, devido sua natureza, exijam um pequeno tempo de resposta poderão ser implementados com a utilização de heurísticas especialmente adaptadas (de tempo polinomial) e cuja solução obtida seja a mais próxima possível de uma solução ótima do problema original. Imagine por exemplo um sistema que gerencie o tráfego de veículos em uma cidade, e que um operador deseje simular, em tempo real, situações distintas que o auxiliem na tomada de decisões. Problemas de planejamento estratégico em uma companhia que envolvam altos custos financeiros de implantação, poderão ser resolvidos através de ferramentas mais robustas e de complexidade exponencial no tamanho do problema desde o tempo total de processamento esteja dentro de um horizonte aceitável pela empresa. Neste caso, o elevado tempo de processamento necessário para

resolução do problema poderá ser perfeitamente aceitável em detrimento de uma maior qualidade na resposta apresentada. A escolha de uma estratégia de resolução, dependerá portanto, de uma combinação de fatores que incluem desde uma avaliação prévia do grau de dificuldade intrínseco ao problema até a elaboração de alternativas de resolução oriundas das características do problema e da qualidade desejada pela solução.

Com o objetivo de melhor classificar problemas computáveis independentemente dos algoritmos disponíveis em dado momento, utiliza-se o conceito de *algoritmo não-determinístico.* Intuitivamente falando, em um algoritmo não-determinístico, a solução do problema pode ser apresentada através de um "oráculo" (simulado através de um paralelismo ilimitado). O que se faz neste caso, será apenas verificar a natureza da solução apresentada. Este tipo de abordagem nos exime da talvez frustrante, apresentação de um algoritmo determinístico polinomial para o problema. Como observado anteriormente, o fato de não conseguirmos resolver um problema através de um algoritmo determinístico polinomial não nos dá um atestado da não existência de tal algoritmo (para maiores detalhes vide Cormem et. al. [1990], Garey&Jonhson[1979]).

I.9.1 - PROBLEMAS DE DECISÃO

Uma grande quantidade de problemas algorítmicos podem ser classificados, basicamente, em 3 categorias distintas a saber: *decisão*, *localização* e *otimização*. Esta classificação se refere, especialmente, a determinadas características observadas na solução do problema.

Nos *problemas de decisão*, deseja-se responder apenas *sim* ou *não* a uma determinada pergunta. Como exemplo, suponha que se deseje saber se existe ou não, em um grafo G qualquer, um circuito hamiltoniano com custo inferior a k (para k inteiro positivo). Já nos *problemas de localização*, deseja-se determinar uma certa estrutura safisfazendo as restrições do problema. Assim, no problema do circuito hamiltoniano, deve-se exibir explicitamente (se existir) um circuito hamiltoniano com custo inferior a k. Nos *problemas de otimização* espera-se encontrar uma determinada estrutura (solução viável) de forma a minimizar ou maximizar uma função objetivo associada. No caso do problema do caixeiro viajante, deve-se determinar, dentre todos os um circuitos hamiltonianos, aquele que tiver menor custo.

Observe que ao resolver um problema de otimização, resolve-se implicitamente os problemas de localização associados. Ou seja, estaremos localizando possíveis estruturas (soluções viáveis) que minimizem uma função objetivo. Obviamente neste caso, a resposta ao problema de decisão também será afirmativa. Pode-se afirmar portanto que um problema de otimização será pelo menos tão difícil quanto um problema de localização ou decisão. O surpreendente neste caso é que a resolução de um problema de decisão também será tão difícil quanto os problemas de localização ou otimização associados! Em outras palavras, pode-se dizer que um problema de decisão terá o mesmo grau de dificuldade que um problema de localização ou otimização! Dessa forma, a análise do grau de dificuldade inerente a cada problema poderá ser simplificada observando-se apenas os problemas de decisão! Assim, se um determinado problema de decisão não puder ser resolvido em tempo polinomial, o problema de otimização associado também não poderá ser resolvido em tempo polinomial e vice-versa.

O problema do Caixeiro Viajante (apresentado abaixo) ilustra bem esta interessante equivalência entre os problemas de decisão e otimização.

I - Caixeiro Viajante (Otimização):

Considere um grafo direcionado $G=(V,A)$ onde para cada arco $(i,j) \in A$ tem-se um custo de transporte c_{ij} associado. Encontre um circuito hamiltoniano W t. q. o custo total de transporte z^* seja o mínimo possível.

II - Caixeiro Viajante (Decisão):

Considere um grafo direcionado G=(V,A) onde para cada arco $(i,j) \in A$ temos um custo de transporte c_{ij} associado. Além disso seja k^*, um inteiro positivo. Existe em G algum circuito hamiltoniano W tal que o custo total de transporte seja inferior ou igual a k^*?

Observe que se for encontrado um algoritmo polinomial para o problema do caixeiro viajante (apresentado em *(I)*) resolve-se também, em tempo polinomial, o problema de decisão associado. Neste caso, bastará comparar o valor da solução ótima z* do problema de otimização com o valor inteiro k^*. Logo, a resolução de (*I)* implica diretamente na resolução de (II). Pode-se mostrar facilmente que a recíproca também é verdadeira, ou seja, caso se tenha um algoritmo polinomial para o problema de decisão, será possível utilizá-lo na construção de um novo algoritmo, também polinomial para o problema de otimização. Para isto, determina-se inicialmente uma cota superior para o valor da solução ótima z* no problema do caixeiro viajante (otimização). É fácil ver que, se C, representa o custo da maior aresta em do grafo então *nC* será um limite superior para *z**. A idéia agora será fazer uma busca binária pelo elemento z* no intervalo *[0,nC]*. Primeiramente, resolve-se o problema de decisão fazendo $k^* = nC/2$. Caso a resposta a esse problema de decisão seja *sim*, continua-se a busca de z* restrita ao intervalo *[0, nC/2],* caso contrário, busca-se *z** no intervalo *[nC/2, nC]*. O processo é repetido até que se tenha um intervalo contendo apenas o elemento *z** (valor da solução ótima no problema de otimização). Observe que a complexidade total do problema de otimização será de ordem *O(f(n).log(nC))* onde *f(n)* representa a complexidade do problema de decisão associado.

I.9.2 - ALGORITMOS NÃO-DETERMINÍSTICOS

Em um algoritmo não-determinístico (para problemas de decisão) pode-se encontrar, além dos comandos usuais de repetição, decisão, atribuição , as seguintes operações:

- *Escolha*(S) → Escolhe um elemento do conjunto S
- *FRACASSO* → Sinaliza término com fracasso
- *SUCESSO* → Sinaliza término com sucesso

A função X ← *Escolha(S),* armazena em X qualquer um dos elementos presentes em S. Não existe uma regra que especifique como tal escolha é realizada. As funções *SUCESSO* e *FRACASSO* sinalizam o final de processamento. Sempre que existir um conjunto de escolhas que nos leve a um término com *SUCESSO*, ela será realizada. Falando informalmente, essa função pode ser encarada como um "oráculo" que sempre visualiza a melhor opção. Um algoritmo não-determinístico termina em *FRACASSO* se, e somente se, não existir nenhum conjunto de escolhas que nos levem a um término com *SUCESSO*. O tempo de processamento para as funções: *Escolha(S), SUCESSO* e *FRACASSO* será de *O(1)* passos.

Uma interpretação determinística de um algoritmo não-determinístico pode ser feita imaginando-se um paralelismo ilimitado. Cada vez que a função *Escolha(S)* for executada, o algoritmo será capaz de gerar cópias de si próprio, cada uma delas correspondendo a uma possível escolha. Assim, tem-se a execução de várias cópias simultâneamente. A primeira cópia que obtiver *SUCESSO* (caso isto venha a ocorrer) cessa imediatamente o processamento de todas as outras cópias e retorna *sim* ao problema de decisão associado. Uma cópia que retorne *FRACASSO* interrompe o processamento apenas daquela cópia em funcionamento. Neste caso, a resposta ao problema de decisão associado será *não*, apenas se todos os demais processadores finalizarem com *FRACASSO.* Pode-se afirmar que, mesmo que a probabilidade de encontrar uma solução correta seja extremamente pequena (mas positiva), ela será determinada se o número de chutes ou

tentativas tende a infinito. Como exemplo, considere um jogo de azar ou loteria. A probabilidade de uma pessoa em particular acertar toda uma sequência correta de valores e ganhar o tão almejado prêmio é sem dúvida bastante pequena. Apesar disso, a probabilidade de se encontrar um ganhador aumenta consideravelmente se milhões de apostadores jogarem simultâneamente! Embora estranho, este tipo de abordagem nos exime da apresentação imediata de um algoritmo determinísto para o problema considerado. O que se faz, é apenas reconhecer a solução apresentada. Informalmente falando, ao se imaginar um paralelismo ilimitado, simula-se, de certo modo, um "oráculo" ou "bola de cristal" que sempre nos retorna uma resposta correta! Desta forma, se a resposta ao problema de decisão considerado for *sim,* um dos infinitos processadores deverá ter finalizado com *SUCESSO!*

Mais adiante no Capítulo III, veremos que a implementação de um algoritmo não-determinístico poderá ser "realizada" através de um algoritmo probabilístico (Método de Las Vegas). Neste caso entretanto, o tempo de processamento poderá tender a infinito. Informalmente falando, um único processador irá cumprir o papel dos infinitos processadores no algoritmo determinístico. Esta abordagem será interessante na prática quando o tempo esperado de processamento for polinomial no tamanho do problema.

Na verdade, máquinas não-determinísticas, como discutidas acima, não são interessantes do ponto de vista prático (de implementação). Elas nos ajudarão apenas na compreensão e classificação dos problemas algorítmicos. Assim, quando determinado problema não puder ser resolvido "eficientemente" através dos mecanismos conhecidos, utiliza-se as funções não-determinísticas detendo-se apenas na natureza da solução apresentada. Esta etapa do algoritmo será chamada *reconhecimento* da solução.

Os algoritmos não-determinísticos são fundamentais na definição da classe *NP* dos problemas de decisão. Para verificar se um dado problema de decisão π, pertence à classe *NP,* duas etapas deverão ser executadas: *exibição* e *reconhecimento*. Na etapa de *exibição*, uma justificativa à resposta *sim* é apresentada. O *reconhecimento* desta solução deverá ser realizada em tempo determinístico polinomial no tamanho da entrada. Vale ressaltar que, se o problema não tem reconhecimento polinomial ele poderá ou não pertencer a *NP.*

Observe que um algoritmo determinístico pode ser visto como um caso particular de um algoritmo não-determinístico, em outras palavras $P \subseteq NP$. Será entretanto que *P* está contido propriamente em *NP*? Ou seja, será possível afirmar que $P \neq NP$*?* Na verdade, se uma subclasse de *NP*, denominada *NP-Completo,* admitir um único problema que possa ser resolvido por um algoritmo polinomial então todos os problemas em *NP* admitirão também um algoritmo polinomial em sua resolução. Neste caso, tem-se $P=NP$! Existe uma forte tendência em se acreditar que não existem algoritmos polinomiais para problemas NP-Completos. Apesar disso, ainda não foi obtida nenhuma prova conclusiva e o problema continua em aberto. Maiores detalhes sobre este assunto podem ser encontrados em Garey&Jonhson[1979].

O exemplo seguinte mostra que o problema do Caixeiro Viajante (decisão) pertence a NP. Toda vez que um algoritmo não determinísticos encontrar *SUCESSO* ou *FRACASSO* durante o processamento ele para imediatamente.

Exemplo I.12: (Ciclo Hamiltoniano)

Suponha $G(V,E)$ um grafo completo não-orientado (onde $|V|=n$ e $|E|=m$) e k um inteiro positivo. Deseja-se determinar se G admite ou não um ciclo hamiltoniano de tamanho menor ou iqual a k Suponha que os custos entre cada para de vértices i,j sejam armazenadas em uma matriz C quadrada nxn. Caso não exista a conexão entre os vértices i e j o custo associado será *infinito.* O ciclo hamiltoniano definindo a sequência de visitas realizadas será armazenado em um vetor S de n elementos. No algoritmo não-determinístico seguinte são apresentadas as etapas de exibição e reconhecimento. Suponha que o vértice *1* do grafo seja o vértice de partida e chegada:

Procedimento I.5: Ciclo Hamiltoniano não-determinístico – Grafo Completo;
Início

```
S ← ∅;
S[1] ← 1;
Para i:=2 até n faça                {exibição}
        j ← Escolha{2,..,n};
        S[i]:=j;
fim;
Para i:=1 até n-1 faça              {começa reconhecimento}
        para j:=i+1 até n faça
                se S[i]=S[j] então FRACASSO;
fim para;
custo_total ← 0;
Para i:=1 até n faça
        custo_total ← custo_total + C[S[i],S[i+1]];
custo_total ← custo_total + C[S[n],S[1]];
Se C[S[i],S[i+1]] > k então FRACASSO;
SUCESSO;
```

fim.

Figura I.16: Circuito Hamiltoniano Não-determinístico

A função *Escolha(.)* acima, retorna passo a passo os véstices presentes no ciclo hamiltoniano. Informalmente falando, tem-se infinitos processadores executando uma cópia deste procedimento. O primeiro processador a resolver o problema, passa essa informação a todos os demais. Observe que a etapa de reconhecimento é realizada por um algoritmo polinomial de ordem $O(n^2)$. Logo, o problema do ciclo hamiltoniano pertencerá à classe *NP* dos problemas de decisão.

Capítulo II

Tópicos em Teoria de Probabilidade

"Como podemos falar em leis do acaso?
Não seria o acaso uma antítese da lei?"
Bertrand Russel

II.1 - INTRODUÇÃO

Neste capítulo serão apresentados alguns dos conceitos e ferramentas necessários à compreensão dos algoritmos randômicos. Será dada uma atenção especial ao estudo das variáveis aleatórias discretas. A máquina RAM probabilística ou máquina de Turing probabilística, analogamente aos modelos determinísticos, trabalham apenas com valores inteiros. Assim, solução gerada e tempo de processamento são quantidades enumeráveis e podem ser representados convenientemente por variáveis aleatórias discretas.

Um *espaço de probabilidade* consiste de um conjunto (possivelmente infinito) de resultados de um experimento, cada um com uma probabilidade de ocorrência associada. Mais formalmente, um espaço de probabilidade será definido em termos de um *espaço amostral* com uma estrutura algébrica (álgebra ou σ-álgebra) e uma *medida de probabilidade* imposta sobre este espaço. A partir destes conceitos define-se probabilidade condicional, eventos independentes, variáveis aleatórias discretas e as principais distribuições de probabilidade associadas.

Maiores detalhes sobre os resultados aqui apresentados (definições, demonstrações, etc) podem ser encontrados em Meyer[1983] ou James[1981].

II.2 - ESPAÇO DE PROBABILIDADE

Ao executar um experimento, especifica-se não somente o procedimento a ser realizado, mas também, aquilo que se deseja observar. Define-se então um *espaço amostral* Ω como sendo o conjunto de todos os resultados possíveis de um dado experimento. Um subconjunto $A \subseteq \Omega$ será chamado *evento.* Se A contiver apenas um elemento então A será chamado *evento elementar.* Como exemplo, considere o seguinte experimento: jogar um dado e observar o resultado. Neste caso, o conjunto $\Omega = \{1,2,3,4,5,6\}$ representará o espaço amostral associado aos possíveis resultados do experimento e os subconjuntos $A_1 = \{2,4,6\}$ e $A_2 = \{1,3,5\}$ são eventos indicando o subconjunto dos números pares e ímpares respectivamente. Além disso, cada um dos valores de Ω representam eventos elementares.

Na realização de um experimento deve-se determinar, em algumas situações, se os objetos envolvidos no experimento são *equilibrados* ou *viciados*. Por exemplo, uma moeda ou dado podem estar viciados se algum de seus eventos elementares ocorrer com maior probabilidade em relação aos demais. Caso contrário, se os resultados do experimento ocorrem com mesma freqüência os objetos estarão equilibrados.

É comum considerar situações onde o espaço amostral Ω é apenas finito ou enumerável (pode-se ter também Ω infinito e não-enumerável). Um conjunto X será *enumerável* quando for finito ou quando existir uma bijeção $f:N\rightarrow X$ (onde $N=\{1,2,3..\}$ representa o conjunto dos números naturais). Quando existir a bijeção e X for infinito então X será chamado *infinito enumerável*. Representa-se por 2^{Ω}, o conjunto de todas as partes de Ω. As vezes pode ser interessante apenas a geração de uma subcoleção de eventos distintos de 2^{Ω}. Entretanto, nem todas as possíveis subcoleções de 2^{Ω} conduzem a um espaço de probabilidade bem definido (James[1981]). A definição de álgebra (ou σ-álgebra) apresentada a seguir irá determinar que tipo de subcoleções de 2^{Ω} conduzirão a uma caracterização precisa de espaço de probabilidade.

Definição II.1: (Álgebra)

Seja Ψ uma coleção de subconjuntos de Ω não vazio. Diz-se que Ψ define uma *álgebra* de subconjuntos de Ω se os seguintes axiomas forem satisfeitos:

(i) $\Omega\in\Psi$

(ii) Se $A\in\Psi$ então $A^c\in\Psi$ (onde A^c é o conjunto complementar de A em relação a Ω).

(iii) Se $A\in\Psi$ e $B\in\Psi$ então $A\cup B\in\Psi$. •

As propriedades básicas e intuitivas acima (conceitos primitivos) serão essenciais ao desenvolvimento da teoria do cálculo de probabilidades. Para exemplificá-las considere um espaço amostral Ω com 4 elementos.

Exemplo II.1: (Álgebra)

Seja 2^{Ω} o conjunto de todas as partes de $\Omega=\{0,1,2,3\}$ e duas subcoleções Ψ_1 e Ψ_2 como descritas a seguir:

2^{Ω} = {∅; {0}; {1}; {2}; {3}; {0,1}; {0,2}; {0,3}; {1,2}; {1,3}; {2,3}; {0,1,2}; {0,1,3}; {1,2,3}; {0,1,2,3}}
Ψ_1 = {∅; {0}; {1}; {0,1}; {2,3}; {0,2,3}; {1,2,3}; {0,1,2,3}}
Ψ_2 = {∅; {0}; {1}; {0,1}; {1,3}; {2,3}; {0,2,3}; {1,2,3}; {0,1,2,3}}

Note que a subcoleção Ψ_1 define uma álgebra de conjuntos já que as propriedades (i), (ii) e (iii) se verificam. Ao contrário, a subcoleção Ψ_2 não define uma álgebra já que o complementar de $\{1,3\}$ não pertence a Ψ_2. •

Se Ψ é uma álgebra de subconjuntos de Ω então as seguintes propriedades são satisfeitas:

P1) $\varnothing\in\Psi$;

P2) Sejam $A_1, A_2, .., A_n \in \Psi$ e $n \geq 0$ um inteiro qualquer. Então $\bigcup_{i=1}^{n} A_i \in \Psi$ e $\bigcap_{i=1}^{n} A_i \in \Psi$.

P3) Se $A\in\Psi$ e $B\in\Psi$ então $A\text{-}B\in\Psi$ (onde $A\text{-}B=A\cap B^c$).

As propriedades P1, P2 e P3, cujas demonstrações são deixadas como exercício, decorrem diretamente da definição de álgebra.

Definição II.2: (σ-álgebra)

Seja Ψ uma coleção de subconjuntos de Ω não-vazio. Então Ψ define uma σ-álgebra de subconjuntos de Ω se, além das propriedades (i) e (ii) a propriedade (iv) descrita abaixo também for satisfeita, ou seja:

iv) Se $A_i \in \Psi$, para i=1,2,... então $\bigcup_{i=1}^{\infty} A_i \in \Psi$. •

Seja Ψ uma σ-álgebra de subconjuntos de Ω. A propriedade seguinte (decorrente de (i), (ii) e (iv)) garante que a interseção enumerável de eventos de Ψ também pertence a Ψ.

P4) Se $A_i \in \Psi$, para i=1,2,... então $\bigcap_{i=1}^{\infty} A_i \in \Psi$.

De maneira geral, uma σ-álgebra é fechada[1] para uma quantidade enumerável de aplicações das operações *união*, *interseção* e *complementar*.

Uma vez definida uma álgebra (ou σ-álgebra) sobre Ω pode-se definir agora uma *medida de probabilidade* sobre Ψ.

Definição II.3: (Medida de probabilidade)

Considere uma função *Pr(.)* definida em uma álgebra Ψ satisfazendo os seguintes postulados ou axiomas (Kolmogorov[1933]):

a) $Pr(A) \geq 0, \quad \forall A \in \Psi$.

b) $Pr(\Omega)=1$.

c) (aditividade-finita) Se $A_1, A_2, .., A_n \in \Psi$ são mutuamente exclusivos (i.e. $A_i \cap A_j = \varnothing$, $\forall\, i \neq j$) então:

$$\Pr\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} \Pr(A_i)$$

Note que $Pr: \Psi \rightarrow [0,1]$. Diz-se neste caso que *Pr(.)* define uma *medida de probabilidade sobre* Ψ ou simplesmente uma *probabilidade em* Ψ.

Caso o número de eventos em Ψ seja infinito e Ψ defina uma σ-álgebra com eventos mutuamente exclusivos pode-se substituir (c) pelo seguinte axioma:

c') (σ-aditividade) $\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \Pr(A_i)$. •

Seja *Pr(.)* uma medida de probabilidade imposta sobre uma σ-álgebra ψ. Se $A \in \Psi$ as seguintes propriedades são satisfeitas:

P5) $\Pr(A^c) = 1 - \Pr(A)$. Note, como conseqüência que $\Pr(\varnothing) = 1 - \Pr(\Omega)$.

P6) $0 \leq Pr(A) \leq 1$.

P7) $A_1 \subseteq A_2 \Rightarrow \Pr(A_1) \leq \Pr(A_2)$.

P8) $\Pr\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} \Pr(A_i)$, p/ algum *n* inteiro positivo.

P9) $\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) \leq \sum_{i=1}^{\infty} \Pr(A_i)$.

A demonstração das propriedades *P5,..,P9* (deixadas como exercício) decorrem diretamente dos axiomas acima (definição II.3).

Tem-se então a seguinte definição de espaço de probabilidade:

Definição II.4: (Espaço de probabilidade)

O trio *(Ω, ψ, Pr(.))* define um *espaço de probabilidade* se e somente se:

[1] Um conjunto X é fechado sobre uma operação $\oplus$ se e somente se $a \oplus b \in X$, $\forall$ a, b $\in$ X.

(a) Ω for um espaço amostral não-vazio.
(b) ψ é uma σ-álgebra de subconjuntos de Ω.
(c) *Pr(.)* é uma medida de probabilidade em ψ. •

Os conceitos e definições apresentados nas seções seguintes relacionam eventos e probabilidades de um espaço de probabilidade:

II.3 - PROBABILIDADE CONDICIONAL

Nesta seção, discute-se algumas das relações existentes entre os eventos de um dado experimento. Define-se probabilidade condicional e fórmula de Bayes.

Em determinadas situações, é possível que se tenha um conhecimento prévio de alguns dos eventos associados a um experimento. Por exemplo, suponha que duas moedas equilibradas sejam jogadas e o resultado obtido seja observado. O espaço amostral associado a este experimento será Ω = *{KK, CK, KC, CC}* onde *K* e *C* representam *cara* e *coroa* respectivamente. Suponha, entretanto, que se deseje determinar a probabilidade de duas *caras* sabendo-se que pelo menos uma *cara* tenha ocorrido. Neste caso, esta informação adicional (garantindo a existência de pelo menos uma cara), exclui a possibilidade de 2 coroas, reduzindo portanto, o conjunto dos resultados observáveis no experimento. Para responder perguntas deste tipo faz-se uso da definição de probabilidade condicional.

Definição II.5: (Probabilidade condicional)
Seja *(Ω, ψ, Pr(.))* um espaço de probabilidade. Se $B \in \psi$ e *Pr(B)>0,* a *probabilidade condicional* de *A* dado *B* é definida por:

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}, \quad A \in \Psi. \qquad \bullet$$

No exemplo acima, pode-se fazer *A = {KK}* (duas caras ocorrem) e *B = {KK, KC, CK}* (pelo menos uma cara ocorre). Assim, conclui-se diretamente que *Pr(A|B) = (1/4)/(3/4) = 1/3.*

Como observado a seguir, o conceito de probabilidade condicional poderá ser útil na determinação da probabilidade de interseção de 2 ou mais eventos.

Proposição P10: (Teorema da Multiplicação ou Probabilidade composta)
Considere *(Ω, ψ, Pr)* um espaço de probabilidade. Além disso, seja *n* é inteiro positivo e $A_1, A_2, .., A_n \in \Psi$. Então:

$$\Pr(A_1 \cap A_2 \cap ... \cap A_n) = \Pr(A_1).\Pr(A_2 \mid A_1).\Pr(A_3 \mid A_2 \cap A_1).....\Pr(A_n \mid A_1 \cap ... \cap A_{n-1}) \qquad \bullet$$

Exemplo II.2: (Probabilidade Composta)
Para exemplificar uma aplicação da propriedade P10 considere que 3 cartas de um baralho devam ser selecionadas sem reposição. Qual a probabilidade que exatamente 3 reis sejam retirados? Para resolver este problema considere A_i o evento "tirar um rei na i-ésima remoção". Se A representa o evento "tirar 3 reis", tem-se que $A = A_1 \cap A_2 \cap A_3$. Utilizando a fórmula de probabilidade composta conclui-se que: $\Pr(A) = \Pr(A_1).\Pr(A_2 \mid A_1).\Pr(A_3 \mid (A_1 \cap A_2)) =$ (4/52) .(3/51).(2/50) = 1/5525. O que ocorre se as cartas forem selecionadas com reposição? •

Em determinadas situações deseja-se particionar o espaço amostral em um subconjunto enumerável de eventos:

Definição II.6: (Partição)

Seja $A_1, A_2, \ldots$ um conjunto enumerável de eventos pertencentes a ψ. Os eventos $A_1, A_2, \ldots$ definem uma *partição* do espaço amostral Ω, quando:

a) $A_i \cap A_j = \varnothing, \ \forall\ i \neq j$.

b) $\bigcup_i A_i = \Omega$ •

Para todo evento B∈ψ, tem-se $B = \bigcup_i (B \cap A_i)$ (vide figura II.1). Como os A_i's são disjuntos então $B \cap A_i$ também serão disjuntos e:

$$\Pr(B) = \sum_i \Pr(B \cap A_i) = \sum_i \Pr(A_i).\Pr(B \mid A_i) \cdot$$

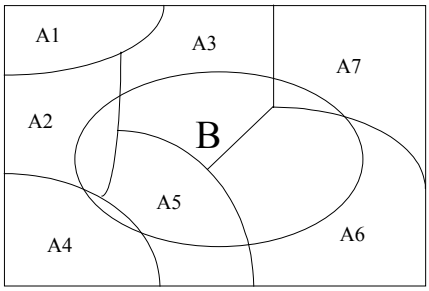


Figura II.1: Partição de um espaço amostral

Mais formalmente, tem-se a seguinte proposição:

Proposição P11: (Probabilidade absoluta)

Seja *(Ω, ψ, Pr(.))* um espaço de probabilidade. Se a seqüência enumerável de eventos aleatórios $A_1, A_2, \ldots$ (pertencentes a *ψ*) formar uma partição de *Ω* então:

$$\Pr(B) = \sum_{i=1}^{\infty} \Pr(A_i).\Pr(B \mid A_i), \quad \forall\ B \in \Psi. \qquad \bullet$$

Este tipo de resultado é interessante quando a probabilidade de um evento $B \in \psi$ não for diretamente determinada. Neste caso, a probabilidade de *B* é obtida em função de eventos com probabilidade conhecida.

Exemplo II.3: (Probabilidade Absoluta)

Para ilustrar este resultado suponha que, em um lote de 100 peças, 20 sejam defeituosas e 80 sejam *não* defeituosas. Deseja-se retirar 2 peças em seguida e sem reposição. Qual a probabilidade da segunda peça ser defeituosa? Para resolver este problema considere os seguintes eventos A={a primeira peça é defeituosa } e B={a segunda peça é defeituosa}. Utilizando a fórmula da probabilidade absoluta tem-se:

$$\Pr(B) = \Pr(B \mid A).\Pr(A) + \Pr(B \mid A^c).\Pr(A^c) = \frac{19}{99}.\frac{1}{5} + \frac{20}{99}.\frac{4}{5} = \frac{1}{5} \qquad \bullet$$

Considere agora o seguinte resultado. Da expressão de probabilidade condicional (definição II.5) tem-se que:

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}, \quad A \in \Psi.$$

De maneira análoga, pode-se concluir que $Pr(A \cap B)=Pr(B|A).Pr(A)$. Além disso, da expressão de probabilidade absoluta: $Pr(B)=Pr(B|A).Pr(A)+Pr(B|A^c).Pr(A^c)$. Substituindo $Pr(A \cap B)$ e $Pr(B)$ na expressão de probabilidade condicional chega-se à seguinte igualdade, também conhecida como *fórmula de Bayes*:

$$\Pr(A \mid B) = \frac{\Pr(B \mid A).\Pr(A)}{\Pr(B \mid A).\Pr(A) + \Pr(B \mid A^c).\Pr(A^c)}$$

O exemplo seguinte ilustra uma aplicação da fórmula de Bayes:

Exemplo II.4: (Fórmula de Bayes)

Considere duas moedas onde uma é equilibrada e outra é viciada. A moeda viciada sempre retorna *cara*. Neste experimento uma moeda será selecionada aleatóriamente e jogada duas vezes. Deseja-se saber qual a probabilidade da moeda selecionada ser viciada sempre que forem observadas duas caras. Suponha que 3 eventos sejam considerados ao final deste experimento: A = {uma moeda viciada é selecionada}, A^c ={uma moeda equilibrada é selecionada} e B = {duas caras são selecionadas}. Assim, aplicando-se diretamente a fórmula de Bayes tem-se:

$$\Pr(A \mid B) = \frac{\Pr(B \mid A).\Pr(A)}{\Pr(B \mid A).\Pr(A) + \Pr(B \mid A^c).\Pr(A^c)} = \frac{1.(1/2)}{1.(1/2) + (1/4).(1/2)} = \frac{4}{5}$$

Logo, a probabilidade que uma moeda viciada seja retirada, caso duas caras sejam observadas, será igual a 80%. •

Suponha que os eventos $A_1, A_2,...$ (pertencentes a ψ) definam uma partição de Ω e $B \in \Omega$ seja um evento qualquer. De maneira geral, a fórmula de Bayes pode ser representada através da seguinte expressão:

$$\Pr(A_i \mid B) = \frac{\Pr(A_i).\Pr(B \mid A_i)}{\sum_j \Pr(A_j).\Pr(B \mid A_j)}$$

A fórmula de Bayes é útil quando as probabilidades de cada um dos eventos A_1 e as probabilidades de B dado A_j ($p/\ j=1,2,...$) são conhecidas sem que se conheça diretamente a probabilidade do evento B.

II.4 - INDEPENDÊNCIA ENTRE EVENTOS

É comum situações onde dois ou mais eventos associados a um experimento ocorram independentemente um do outro. Neste caso, a ocorrência ou não de um evento qualquer B não irá influenciar na ocorrência ou não de outro evento A. Formalmente, considere a seguinte definição de eventos independentes:

Definição II.7: (Eventos Independentes)

Seja $(\Omega, \psi, Pr(.))$ um espaço de probabilidade. Dois eventos aleatórios A e B pertencentes a ψ são *independentes* se e somente se $\Pr(A \cap B) = \Pr(A).\Pr(B)$. Em outras palavras, se $Pr(A) \neq 0$ então $Pr(B|A) = Pr(B)$. •

Exemplo II.5: (Independência entre eventos)

Suponha que um dado equilibrado seja jogado 2 vezes consecutivas. Considere que os seguintes eventos estejam associados a este experimento:

A = {o primeiro dado retorna par}
B = {o segundo dado retorna 5 ou 6}

Se *x* e *y* representam os valores obtidos no primeiro e segundo dados respectivamente então $\Omega = \{(x,y): 1 \le x \le 6 \text{ e } 1 \le y \le 6\}$. É fácil ver neste caso que: $Pr(A) = 18/36 = 1/2$ e $Pr(B) = 12/36 = 1/3$. Ainda $Pr(A \cap B) = 6/36 = 1/6$. Segue então que $Pr(A \cap B) = Pr(A).Pr(B) = 1/6$. Conclui-se portanto que *A* e *B* são eventos independentes. Note ainda que $Pr(B|A) = 12/36 = 1/3$, ou seja, $Pr(B|A) = Pr(B) = 1/3$ (a ocorrência do evento A não influencia em nada a probabilidade de *B*). •

Definição II.8: (Eventos *dois*-a-*dois* independentes)

Uma coleção de eventos $A_1, A_2, A_3, \ldots, A_n$ pertencentes a um espaço amostral Ω é dita *dois-a-dois independente* se e somente se $Pr(A_i \cap Aj) = Pr(A_i).Pr(A_j), \ \forall\, 1 \le i \le j \le n$.

Definição II.9: (Eventos mutuamente independentes)

Uma coleção de eventos $A_1, A_2, A_3, \ldots, A_n$ pertencentes a um espaço amostral Ω é *mutuamente independente* (ou simplesmente *independente*) se, e somente se, qualquer subcoleção $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$, onde $2 \le k \le n$ e $1 \le i_1 \le i_2 \le \ldots \le i_k \le n$ a seguinte igualdade for satisfeita:

$$\Pr(A_{i_1} \cap A_{i_2} \cap \ldots \cap A_{i_k}) = \Pr(A_{i_1}).\Pr(A_{i_2}) \ldots \Pr(A_{i_k}).$$ •

Obviamente, se uma coleção de eventos é mutuamente independente, ela será também dois-a-dois independente, entretanto, a recíproca não é verdadeira como pode ser observado no exemplo seguinte:

Exemplo II.6: (Independência entre eventos)

Para ilustrar os conceitos acima considere o experimento onde duas moedas equilibradas são jogadas e 3 eventos distintos são observados.

A_1 = {a primeira moeda é *cara*},
A_2 = {a segunda moeda é *cara*},
A_3 = {as duas moedas são distintas}.

É fácil ver que $Pr(A_1) = Pr(A_2) = Pr(A_3) = 1/2$, $Pr(A_1 \cap A_2) = Pr(A_1).Pr(A_2)$, $Pr(A_1 \cap A_3) = Pr(A_1).Pr(A_3)$ e $Pr(A_2 \cap A_3) = Pr(A_2).Pr(A_3)$ sendo portanto dois-a-dois independentes. Entretanto, $Pr(A_1 \cap A_2 \cap A_3) \neq Pr(A_1).Pr(A_2).Pr(A_3)$. Note que, $Pr(A_1 \cap A_2 \cap A_3) = 0$ e $Pr(A_1).Pr(A_2).Pr(A_3) = 1/8$, logo os eventos A_1, A_2 e A_3 não são mutuamente independentes! •

II.5 - VARIÁVEIS ALEATÓRIAS DISCRETAS:

Em probabilidade, associa-se freqüentemente valores numéricos aos possíveis resultados ou eventos de um experimento. Estes resultados podem ser representados convenientemente através de variáveis aleatórias. Aqui, será dada uma atenção especial às variáveis aleatórias discretas. Mais formalmente, considere a seguinte definição:

Definição II.10: (Variável Aleatória)

Seja ε um experimento e *(Ω,ψ,Pr(.))* um espaço de probabilidade associado. Uma função *X* que associa a cada elemento w∈Ω um número real *X(w)* é denominada *variável aleatória.* Se $X:\Omega \rightarrow R_X$ é variável aleatória e o contradomínio R_X for finito ou infinito enumerável então *X* será uma *variável aleatória discreta.* Note por exemplo que, se R_X é infinito enumerável então $R_X=\{x_1,x_2,...\}$, onde $x_i \in R$ para $i=1,2,....$ •

Exemplo II.7: (Variáveis Aleatórias)

(a) Novamente, considere um experimento onde *duas* moedas equilibradas são jogadas. Suponha ainda que se deseje observar o número de caras obtidos neste experimento. Como $\Omega = \{KK, CK, KC, CC\}$, uma variável aleatória X pode ser obtida fazendo-se $X(KK) = 2$, $X(CK)= X(KC) = 1$ e $X(CC) = 0$. Uma outra variável *Y* pode ser gerada sobre o mesmo experimento fazendo-se, por exemplo, $Y(KK) = Y(CC) = 1$ e $Y(KC) = Y(CK) = 0$, ou seja, a imagem de *Y* retorna 1 se as moedas forem e iguais e retorna 0 caso contrário.

(b) Em outro experimento, um dado equilibrado é jogado e seu resultado observado. O experimento deve ser repetido até que o número 6 seja obtido. Suponha que este problema seja resolvido por um computador onde cada escolha aleatória consuma uma unidade de tempo (p. ex., 1 segundo) . Neste caso, pode-se definir uma variável aleatória *X* representando o tempo total de processamento até que a propriedade P, desejada pelo algoritmo, seja satisfeita (neste caso, sortear o número 6). É fácil ver neste exemplo que $R_X=\{1,2,3,...\}$, ou seja, $x_i=i$ para $i=1,2,3....$ •

Definição II.11: (Função Densidade de Probabilidade)

Seja X uma variável aleatória discreta assumindo um número infinito enumerável de valores $x_1,x_2,x_3,....$ A cada número x_i será associado um número $Pr(x_i) = Pr(X= x_i)$, denominado probabilidade de x_i. Os valores $Pr(x_i)$, para $i=1,2,...$ deverão satisfazer às seguintes condições:

a) $Pr(x_i) \geq 0$, para $i = 1,2,3,...$

b) $\sum_{I=1}^{\infty} \Pr(x_i) = 1$

A função $Pr:R_X \rightarrow [0,1]$ é denominada *função densidade de probabilidade.* A coleção $[x_i, Pr(x_i)]$ é algumas vezes chamada de *distribuição de probabilidade.* •

Note que, se *X* é variável aleatória e x∈ℜ, o evento $X=x$ estará associado ao conjunto $\{w \in \Omega: X(w)=x\}$. Assim:

$$\Pr(X = x) = \sum_{w \in \Omega} \Pr(w)$$

Considere o seguinte exemplo:

Exemplo II.8: (Função densidade de probabilidade)

(a) Considere novamente o exemplo II.7.(a) acima. Como as *duas* moedas envolvidas no experimento são equilibradas, é fácil ver que $Pr(X=2) =1/4$, $Pr(X=1) =1/2$ e $Pr(X=0) =1/4$. Na segunda variável conclui-se diretamente que $Pr(Y=1) = Pr(Y=0) =1/2$.

(b) No experimento II7.(b) pode-se fazer, por exemplo, $Pr(X=n) = (1-p)^{n-1}.p$ onde $n=1,2,3,....$ Neste caso, *n* representa o tempo total de execução em segundos do algoritmo randômico (número de "chutes") e p=1/6 representa a probabilidade de ocorrência da propriedade P (sortear o número 6). Enquanto o número 6 não for obtido (evento complementar) um novo "chute" será realizado.

Observe ainda que, nestes *dois* experimentos as condições (a) e (b) da definição II.11 são atendidas (verifique). •

Definição II.12: (Função Densidade de Probabilidade Conjunta)

Suponha que X e Y sejam duas variáveis aleatórias discretas associadas a um espaço amostral Ω. A *função densidade de probabilidade conjunta* $f_{XY}:R_X x R_Y \rightarrow [0,1]$ nas variáveis *X* e *Y* é definida por: $f_{XY}(x,y)=Pr(X=x \text{ e } Y=y)$. •

Da definição acima tem-se que, se $y \in R_Y$ é fixo então:

$$\Pr(Y = y) = \sum_x \Pr(X = x \ e \ Y = y).$$

Analogamente, se $x \in R_X$ é fixo então:

$$\Pr(X = x) = \sum_y \Pr(X = x \ e \ Y = y).$$

Da definição II.5, de probabilidade condicional, conclui-se que:

$$\Pr(X = x \mid Y = y) = \frac{\Pr(X = x \ e \ Y = y)}{\Pr(Y = y)}$$

Assim, duas variáveis aleatórias discretas *X* e *Y* associadas a um espaço amostral Ω serão independentes se, e somente se, $Pr(X=x \text{ e } Y=y) = Pr(X=x).Pr(Y=y)$, qualquer que seja $x \in R_X$ e $y \in R_Y$. Equivalentemente, *X* e *Y* serão independentes se $Pr(X=x|Y=y) = Pr(X=x)$.

A seguir define-se *valor esperado* ou *expectância* de uma variável aleatória:

Definição II.13: (Valor Esperado)

Seja *X* uma variável aleatória discreta. Se $R_X=\{x_1,x_2,,...\}$ e $Pr(X=x_i)=Pr(x_i)$, para $i=1,2,3....$, o *valor esperado E(X)* será denotado por:

$$E(X) = \sum_{i=1}^{\infty} x_i \Pr(x_i)$$

se a série $\sum_{i=1}^{\infty} x_i \Pr(x_i)$ convergir absolutamente, isto é, se $\sum_{i=1}^{\infty} | x_i | \Pr(x_i) < \infty$. •

Definição II.14: (Valor esperado de uma função de uma variável aleatória)

Seja X uma variável aleatória discreta assumindo valores $x_1, x_2, x_3, ...$ e seja $Y = H(X)$ uma de função de *X*. O *valor esperado E(Y)* será dado por:

$$E(Y) = E(H(X)) = \sum_{i=1}^{\infty} H(x_i).\Pr(x_i)$$

se a série $\sum_{i=1}^{\infty} H(x_i).\Pr(x_i)$ convergir absolutamente. •

Definição II.15: (Valor esperado condicionado)

Seja *(X,Y)* uma variável aleatória discreta bidimensional. Define-se *valor esperado condicionado* de *X* para um dado $Y=y_i$ como sendo:

$$E(X \mid y_i) = \sum_{i=1}^{\infty} x_i \Pr(x_i \mid y_i)$$

O valor esperado condicionado de Y, para um dado X, é definido analogamente. •

Sejam *X* e *Y* variáveis aleatórias definidas sobre um espaço amostral *Ω*. Algumas propriedades importantes do valor esperado podem ser obtidas com base nas definições acima:

P12) Se *c* é constante então *E(cX+Y) = cE(X)+E(Y).*
P13) *E(X.Y) = E(X).E(Y),* se *X* e *Y* são independentes.

De maneira geral, se $X_1, X_2, ..., X_n$ são variáveis aleatórias sobre o espaço amostral *Ω* então valem as seguintes propriedades:

P14) $E(X_1 + X_2 + ... + X_n) = E(X_1) + E(X_2) + ... + E(X_n)$;
P15) $E(X_1.X_2.....X_n) = E(X_1).E(X_2).....E(X_n)$, se $X_1, X_2, ..., X_n$ forem mutuamente independentes;
P16) *E(E(X|Y))=E(X)* e *E(E(Y|X))=E(Y).*

As demonstrações das propriedades P12 a P16 são deixadas como exercício. O exemplo seguinte ilustra uma utilização da propriedade (P14):

Exemplo II.9: (Linearidade do valor esperado)
Um navio atraca em um porto marítimo com 40 marinheiros a bordo. Todos saem a noite para comemorar a chegada. Ao voltarem para o navio, todos eles embriagados em função da grande comemoração, escolhem aleatóriamente uma cabine para dormir. Qual o número esperado de marinheiros que conseguem retornar para sua própria cabine?

Seja X_i uma variável aleatória que indica se o *i-ésimo* marinheiro encontrou ou não sua própria cabine. Assim, $X_i = 1$ em caso afirmativo e $X_i = 0$, caso contrário. Desta forma, tem-se que 1/40 é a probabilidade do i-ésimo marinheiro encontrar sua própria cabine. Logo, $E(X_i) = 1.(1/40) + 0.(39/40)$. Da linearidade do valor esperado tem-se que:

$$E\left(\sum_{i=1}^{40} X_i\right) = \sum_{i=1}^{40} E(X_i) = \sum_{i=1}^{40} \frac{1}{40} = 1.$$

Espera-se então que um marinheiro encontre sua própria cabine. •

A *variância* de uma variável aleatória *X*, assim como o *desvio padrão*, representam medidas de variação de *X* em torno de seu valor esperado *E(X).* Mais formalmente:

Definição II.16: (Variância e Desvio Padrão)
Seja *X* uma variável aleatória. A *variância* de X é definida como: $\sigma_X^2 =$ *V(X)* = $E\left[(X - \mu_X)^2\right]$, onde E(X) = μ_X. O *desvio-padrão* de X será igual a $\sigma_X = \sqrt{\sigma_X^2}$. •

São listadas a seguir algumas propriedades da envolvendo a variância de uma ou mais variáveis aleatórias:

P17) Se *c* é constante então $V(cX) = c^2 V(X)$;
P18) Se *c* é constante então $V(X + c) = V(X)$;
P19) $V(X) = E(X^2) - (E(X))^2$;

A demonstração dessas propriedades são deixadas como exercício.

Proposição P20: Sejam $X_1, X_2, .., X_N$ variáveis aleatórias independentes. Se $X = X_1 + ... + X_N$ então $V(X) = V(X_1) + ... + V(X_N)$.

Prova: Considere inicialmente a seguinte notação:

$$\mu_i = E(X_i) \quad \text{e} \quad \mu = E(X) = \sum_{i=1}^{m} E(X_i) = \sum_{i=1}^{m} \mu_i$$

Assim,

$$V(X) = E[(X-\mu)^2] = E\left[\left(\sum_{i=1}^{n} X_i - \sum_{i=1}^{n} \mu_i\right)^2\right] = E\left[\left(\sum_{i=1}^{n} (X_i - \mu_i)\right)^2\right].$$

Fazendo $Y_i = X_i - \mu_i$ segue da linearidade do valor esperado que:

$$E\left[\left(\sum_{i=1}^{n} Y_i\right)^2\right] = \sum_{i=1}^{n} E(Y_i^2) + 2\sum_{i<j} E(Y_i Y_j)$$

Como $X_1, X_2, .., X_n$ são independentes os pares X_i, X_j juntamente com os pares $X_i - \mu_i$ e $X_j - \mu_j$ também o são.

Como *E(XY) = E(X).E(Y)* (proposição P13) então:

$$V(X) = E[(X-\mu)^2] = E\left[(X_i - \mu_i)^2\right] + 2\sum_{i<j} E(X_i - \mu_i).E(X_j - \mu_j)$$

Mas $E(X_i - \mu_i) = E(X_i) - \mu_i = 0.$ Assim: $V(X) = V(X_1) + ... + V(X_N)$ •

O resultado seguinte mostra de maneira mais precisa a variabilidade da variável aleatória X em relação a seu valor esperado *E(X).*

Proposição P21: (Desigualdade de Tchebyshev)

Se X é variável aleatória com expectância μ_X e desvio padrão σ_X então:

$$\Pr\left(|X - \mu_X| \geq t\sigma_X\right) \leq \frac{1}{t^2}, \quad \forall\, t \in \Re^+. \qquad \bullet$$

A Figura II.2 ilustra melhor a utilização da desigualdade de Tchebyshev:

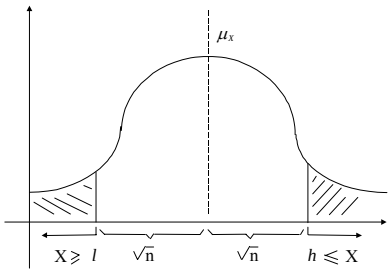


Figura II.2: Desigualdadede Tchebyshev

II.6 - ALGUMAS DISTRIBUIÇÕES DE PROBABILIDADE IMPORTANTES

Apresenta-se a seguir algumas distribuições de probabilidade bastante comuns no estudo das variáveis aleatórias discretas:

Definição II.17: (Distribuição de Bernoulli)

Uma variável aleatória *X* satisfaz a distribuição de Benoulli com parâmetro *p* se, e somente se, *Pr(X = 0) = 1-p e Pr(X = 1) = p.* É fácil ver neste caso que *E(X) = p* e $V(X) = p - p^2$ (verifique).

Definição II.18: (Distribuição Binomial)

Seja ε um experimento e A algum evento associado com probabilidade *Pr(A)=p* (portanto, $p(A^c)=1-p$). Para *n* repetições de ε, o espaço amostral será formado por todas as seqüências possíveis $(x_1,x_2,...,x_n)$ onde $x_i \in A$ ou $x_i \in A^c$ *p/ i=1,2,..,n.* Neste caso, a variável aleatória *X,* representando o número total de ocorrências do evento *A*, define uma distribuição binômial com parâmetros *n* e *p.*

Se *X=k* então:

$$\Pr(X = k) = \binom{n}{k}.p^k.(1-p)^{n-k} \qquad \text{onde k=0,1,2,...,n}$$ •

Proposição P22: Se uma variável aleatória *X* com parâmetros *n* e *p* é binômialmente distribuída então *E(X)=n.p* e *V(X)=n.p(1-p).* •

Exemplo II.10: (Distribuição Binomial)

Suponha que uma urna contenha um conjunto de bolas azuis (A) e vermelhas (V). Dessas, 20% são azuis e 80% são vermelhas. Considere um experimento onde 3 bolas são selecionadas ao acaso e com reposição. Seja X uma variável aleatória indicando o número de bolas azuis selecionadas. É fácil ver que $\Re_X = \{0,1,2,3\}$ define o conjunto dos possíveis valores de X. Qual o valor de *Pr(X=2)?* Observe neste caso que a variável X define uma distribuição binomial com parâmetros n=3 e p=0,2. Assim:

$$\Pr(X = 2) = \binom{3}{2}.(0,2)^2.(0,8) = 0,096.$$

O valor esperado para o número de bolas azuis selecionadas será *E(X)=0,6* e *V(X) = 0,48.* •

Definição II.19: (Distribuição Geométrica)

Considere um experimento obtido através de uma seqüência de ensaios de Bernoulli. Seja *p* a probabilidade de sucesso e *1-p* a probabilidade de fracasso. Quantos passos (valor esperado) serão executados até um evento com sucesso tenha sido encontrado?

Seja *X=k* o número de repetições até que um sucesso tenha sido obtido. Assim:

$$\Pr(X = k) = (1-p)^{k-1}.p \quad \text{onde k=1,2,3,...}$$ •

Proposição P23: Seja X uma variável aleatória com distribuição geométrica e parâmetro *p.* Então *E(X)=1/p* e $V(X) = (1-p)/p^2$. •

Prova: Fazendo *q =1-p,* segue da definição de valor esperado que:

$$E(X) = \sum_{i=1}^{\infty} i.p.q^{i-1} = p\sum_{i=1}^{\infty} \frac{d}{di} q^i \cdot$$

Assim, da linearidade da derivada e da derivada do quociente entre duas funções (vide Lima[1976]) conclui-se que:

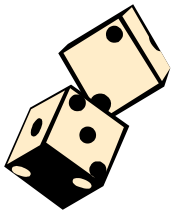
$$E(X) = p\sum_{i=1}^{\infty} \frac{d}{di}.q^i = p.\frac{d}{di}\left(\sum_{i=1}^{\infty} q^i\right) = p.\frac{d}{di}\left(\frac{q}{1-q}\right) = \frac{1}{p}.$$

De maneira análoga, prova-se que $V(X) = (1-p)/p^2$. •

Exemplo II.10: (Distribuição Geométrica)

Suponha que um dado equilibrado seja jogado e seu resultado observado (exemplo II7(b)). Qual o valor esperado para o número de jogadas até que o número 6 seja obtido? Como *p=1/6, X* define uma variável com distribuição geométrica e probabilidade de sucesso *p = 1/6.* Segue então de proposição P23 que *E(X) = 6* e *V(X) = 30.* •

PARTE II



ALGORITMOS RANDÔMICOS

E

OTIMIZAÇÃO COMBINATÓRIA

Capítulo III

Uma Introdução aos Algoritmos Randômicos

"Será que Deus joga dados ?"
Ian Stewart.

III.1 - INTRODUÇÃO

Embora se conheça aplicações envolvendo algoritmos randômicos desde épocas primitivas (Shallit[1992]) os primeiros artigos sobre este assunto datam do final da década de 70 com os trabalhos de Rabin[1976] e Solavay e Strassen[1977] para o problema do reconhecimento de números primos (*Primality Test*). As décadas de 1980 e 1990 testemunharam, a partir de então, um enorme crescimento da área de algoritmos randômicos. Eles emergiram de aplicações voltadas unicamente à teoria dos números e geometria computacional para problemas nas mais diversas áreas de interesse. Uma gama enorme de pesquisadores tem utilizado, cada vez mais, técnicas e ferramentas oriundas de modelos probabilísticos, sejam eles seqüenciais ou paralelos. Como exemplo, pode-se citar aplicações em algoritmos *on-line*, otimização combinatória, criptografia, geometria computacional, teoria dos números, estrutura de dados, processamento paralelo e distribuído entre outras (Motwani&Raghavan [1995], Karp[1991], Gupta et al.[1994]).

Pode-se dizer que um algoritmo determinístico se caracteriza, basicamente, por uma seqüência finita de "passos elementares" voltados para a resolução de um determinado problema. Este algoritmo pode ser caracterizado por uma função $f(.)$ onde $S = f(E)$, sendo E uma entrada qualquer do problema e S uma solução do problema. Tipicamente, o tempo de execução de um algoritmo determinístico é sempre fixo, ou seja, sucessivas repetições do algoritmo aplicadas a uma mesma entrada E resultam sempre em uma mesma saída com igual tempo de processamento. O mesmo não ocorre, por exemplo, com os algoritmos randômicos ou probabilísticos onde cada execução poderá produzir uma saída diferente baseada em eventos aleatórios. Nestas situações, o resultado obtido e/ou o tempo de processamento definem uma "função randômica" da entrada. Surpreendentemente, para uma grande quantidade de problemas, a utilização de algoritmos randômicos se constitui na forma mais simples e/ou mais rápida de implementação! Nestes casos, sua utilização implica em uma melhora de desempenho quando comparada aos algoritmos puramente determinísticos.

Outra característica importante dos algoritmos randômicos pode ser ilustrada através do seguinte exemplo. Imagine um jogo entre dois oponentes onde um deles desenvolva um algoritmo determinístico e o outro proponha a pior instância para aquele algoritmo. Cada modificação no algoritmo por um dos jogadores implicará na escolha de outra instância por parte de seu oponente, sempre recaindo no pior caso possível. Nos algoritmos randômicos, entretanto, escolhe-se um procedimento cujo desempenho independa da instância estabelecida por um dos oponentes. Neste caso, o tempo esperado de processamento (complexidade) se aplica para qualquer instância do problema e não apenas para a pior instância selecionada (complexidade de pior caso). O

comportamento de um algoritmo randômico poderá variar mesmo quando aplicado, repetidas vezes para uma mesma entrada! Desta forma, o tempo de processamento se torna uma variável aleatória e a análise de tempo de processamento implica em uma maior compreensão da distribuição de probabilidade associada. Finalmente, pode-se dizer que, em um algoritmo randômico, o valor esperado irá depender de escolhas aleatórias feitas no algoritmo e não de distribuições impostas sobre a entrada de dados. Este comportamento diferencia o tempo esperado de processamento, presente nos algoritmos randômicos, da complexidade de caso médio, normalmente utilizada na análise de algoritmos determinísticos.

Neste capítulo, foram selecionados alguns tópicos visando uma breve explanação sobre o tema. Alguns dos exemplos apresentados não tratam especificamente de problemas de otimização. Isto não impede entretanto, que as mesmas idéias e conceitos sejam igualmente aplicadas a problemas combinatórios.

Inicialmente na seção III.2, é feita uma apresentação dos métodos de Las Vegas e Monte Carlo. Aplicações utilizando estes dois tipos de abordagem são apresentadas nas seções III.3 e III.4 respectivamente. No método de Monte Carlo será dada uma atenção especial à técnica de Impressão Digital (*Fingerprinting*) e Amostra Randômica (*Random Sampling*). As principais classes de complexidade associadas a modelos probabilísticos são apresentadas na seção III.8.

Aos leitores ainda não fe4(m)7.4(i)1.7(l)-9.5(i)1.7ar(iz)1045(ado)21.7as com0.7(e4(m)7.4(a)10.4((f)12(e)-0.
I I el(e)10.7s, ssãodolgounster(m)7.6(o)10.7(s18(es(en)1075cb)-0.5(i)-924sa)1075(i)-924(s18(àeico)1075em)7.7(p)
pítul(o)10.6((I)23.4(I)23.4,.)-3.4s(um)7.6(a)10.6(

até que um número par seja observado, tem-se o método de Las Vegas com probabilidade de sucesso igual a 1! Neste caso, o número de repetições do procedimento será representado por uma variável aleatória com distribuição geométrica e valor esperado *1/p* (vide Capítulo II). Como *p=1/2,* teremos aproximadamente 2 repetições do algoritmo.

De maneira geral, considere um problema π com instância *I* de tamanho *n*. Além disso, considere um algoritmo de Monte Carlo *A* com tempo esperado de processamento $T_1(n)$ e probabilidade de acerto igual a *p(n).* Suponha que, gerada uma solução para π, a corretude ou não desta solução possa ser constatada em tempo $T_2(n)$. O algoritmo de Las Vegas, neste caso, consiste na simples repetição de *A* até que uma resposta correta seja obtida. Em outras palavras, uma solução correta (sucesso) é gerada satisfazendo uma distribuição geométrica com valor esperado *1/p(n)* (vide Capítulo II). Como o tempo de processamento associado a cada repetição é

Algoritmo III.1: Las Vegas - Coloração de conjuntos
Início
leia (S_1,S_2,...,S_k) ;
viavel ← F; {inicializa coloração viável com falso}
Enquanto não viavel **faça**
para i=1 **até** n **faça**
x_i ← escolha aleatoriamente *branco* ou *preto*;
viavel ← checa-viabilidade (x,S_1,S_2,...,S_k) ; {retorna V ou F}
fim;
fim.

Figura III.1: Las Vegas - coloração de conjuntos

A figura III.2 ilustra uma coloração válida com 13 elementos onde $k=5$ e $r=5$.

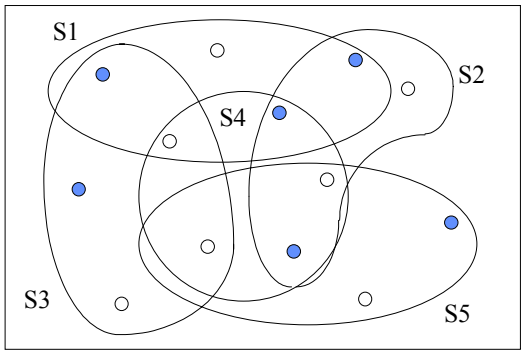


Figura III.2: Coloração válida p/ o problema de coloração de conjuntos

Para mostrar que o algoritmo III.1 sempre encontra uma solução desejada (em uma dada iteração do algoritmo) deve-se mostrar inicialmente que uma coloração válida é obtida com probabilidade de sucesso p estritamente positiva. Para comprovar este fato, considere Y uma variável aleatória indicando o número total de iterações até que uma configuração viável seja obtida. Observe que Y define uma distribuição geométrica com parâmetro $p>0$ e valor esperado $E(Y)$ igual a $1/p$ (vide Capítulo II). Isto garante um número finito de passos mesmo que a probabilidade de sucesso em uma dada iteração seja bastante pequena.

Suponha que um evento A ocorra sempre que uma coloração viável for obtida. Caso contrário, a ocorrência do evento complementar A^c irá indicar que pelo menos um dos subconjuntos S_i ($p/\ i \in \{1,..,k\}$) foi colorido com mesma cor (*branco* ou *preto*). Analogamente, sejam B_i ($p/\ i=1,..,k$), eventos indicando que os elementos de S_i foram pintados com cores distintas. Logo, os eventos complementares B_i^c irão indicar que apenas uma cor foi utilizada na coloração de S_i.

Como $|S_i|=r$ para $i=1,..,k$, tem-se que $Pr(B_i^c)=2/2^r$. Note que cada S_i pode ser colorido só com *branco* ou só com *preto*. A probabilidade de fracasso $Pr(A^c)$ (vide Capítulo II) será dada por:

$$\Pr(A^c) = \Pr\left(\bigcup_{i=1}^{k} B_i^c\right) \le \sum_{i=1}^{k} \Pr(B_i^c) = 2k / 2^r \cdot$$

Como $k \le 2^{r-2}$, conclui-se diretamente que $Pr(A^c) \le 1/2$. Isto garante que a probabilidade de sucesso em uma iteração do algoritmo randômico será superior ou igual a 50%, ou seja, $Pr(A) \ge 1/2$.

Repete-se o processo sempre que alguma falha tenha sido detectada pelo procedimento *Checa-viabilidade*.

Observe neste exemplo que, fazendo a probabilidade de sucesso igual a *1/2,* o número esperado de iterações *E(Y)* para obtenção de uma coloração válida será apenas 2! Informalmente falando, é provável que, em aproximadamente 2 repetições do algoritmo, já se tenha uma coloração desejada!

Se a probabilidade de sucesso for muito pequena, ter-se-á provavelmente, um longo período de processamento. Como discutido anteriormente, o método de Las Vegas será eficiente se o valor esperado para o tempo de processamento for polinomial no tamanho da entrada. No problema da coloração de conjuntos serão necessárias *O(n)* passos para coloração completa de *S* e *O(kr)* passos, no pior caso, para checar se a coloração é ou não viável (procedimento checa-viabilidade). Note que, por hipótese, os *k* subconjuntos de S (de tamanho *r*) são distintos entre si, logo $k \leq n!/(r!(n-r)!)$. Como *E(Y)=2,* o tempo esperado de processamento será polinomial e igual a *O(2(n+kr)).* O procedimento de Las Vegas será polinomial no tamanho do problema ? Verifique !

III.3.2 - Quicksort randômico

O procedimento *Quicksort* necessitará, no pior caso, de $O(n^2)$ iterações para a ordenação completa de uma lista *A* de *n* elementos (vide Capítulo I). Isto ocorrerá sempre que a lista *A* já estiver ordenada (ordem crescente ou decrescente) e *chave* for o primeiro (ou último) elemento selecionado a cada chamada recursiva. Mesmo escolhendo-se elementos *chave* não necessariamente na primeira posição, pode-se ainda construir instâncias (ordenadas ou não) cujo tempo de processamento no pior caso seja da ordem de $O(n^2)$ iterações. Em certo sentido, isto pode ser visto como um jogo entre dois oponentes onde um deles desenvolve um algoritmo e o outro propõe a pior instância para aquele algoritmo.

No *Quicksort* randômico, o pivô ou elemento chave é escolhido aleatoriamente e trocado com a primeira posição de A. As demais etapas são idênticas ao algoritmo determinístico (Capítulo I). Pode-se provar neste caso que o valor esperado para o total de comparações será igual a *O(nlogn)* independentemente da instância considerada.

Suponha que a lista A, de *n* elementos, deva ser colocada em ordem crescente. Considere ainda *A(i)* o *i-ésimo* menor elemento de *A* e X_{ij} uma variável aleatória valendo 1, se *A(i)* e *A(j)* são comparados entre si em uma iteração qualquer do algoritmo. Caso contrário $X_{ij}=0$. O número total de comparações executadas será dado por:

$$\sum_{i=1}^{n} \sum_{j>i} X_{ij}$$

Para se calcular o tempo esperado de processamento, é suficiente computar o valor esperado do total de comparações executadas (supondo que cada comparação consuma uma unidade de tempo). Assim, da linearidade do valor esperado (Capítulo II), tem-se que o total esperado de comparações será dado por:

$$E\left(\sum_{i=1}^{n} \sum_{j>i} X_{ij} \right) = \sum_{i=1}^{n} \sum_{j>i} E(X_{ij})$$

Seja p_{ij}, a probabilidade de comparação entre *A(i)* e *A(j)* em alguma iteração do algoritmo. Como X_{ij} assume valores 0 ou 1 tem-se que:

$$E(X_{ij}) = 1.p_{ij} + 0.(1 - p_{ij}) = p_{ij}$$

Para determinar de uma expressão de probabilidade p_{ij}, considere inicialmente o caso onde $i=1$ e $j=n$. Uma comparação direta entre $A(1)$ e $A(n)$ será possível se e somente se $A(1)$ ou $A(n)$ forem escolhidos como elemento *chave* antes que qualquer outro elemento $A(k)$ (entre $A(1)$ e $A(n)$) seja escolhido! Se $A(k)$ para $1<k<n$ for selecionado primeiramente, tem-se a divisão da lista $A\{A(k)\}$ em duas sub-listas distintas, uma contendo $A(1)$ e outra contendo $A(n)$. Desta forma, não será mais possível uma comparação entre $A(1)$ e $A(n)$ nas iterações posteriores do algoritmo. Conclui-se então que $p_{1n}=2/n$.

Generalizando o processo no cálculo de p_{ij} tem-se que qualquer um dos elementos $A(i), A(i+1),...,A(j)$ pode ser escolhido como pivô com igual probabilidade. Entretanto, uma comparação entre $A(i)$ e $A(j)$ só será possível se nenhum elemento $A(k)$ entre $A(i)$ e $A(j)$ for escolhido antes de $A(i)$ ou $A(j)$ respectivamente! Desta forma tem-se que: $p_{ij} = 2/(j-i+1)$. Note por exemplo que, se $j=i+1$ então $A(i)$ e $A(j)$ serão elementos consecutivos na lista ordenada. Como não existe nenhum elemento entre eles pode-se afirmar que $A(i)$ e $A(j)$ serão comparados com probabilidade 1!

Substituindo p_{ij} na expressão do valor esperado chega-se a:

$$\sum_{i=1}^{n}\sum_{j>i} E(X_{ij}) = \sum_{i=1}^{n}\sum_{j>i} \frac{2}{j-i+1} \leq n\sum_{j=2}^{n}\frac{2}{j} \cong 2n\ln n$$

Este resultado é obtido majorando-se convenientemente as parcelas presentes no primeiro membro da desigualdade. Mudando da base *e* para a base *2* conclui-se diretamente que o tempo esperado para o total de comparações será igual a $O(nlogn)$. Uma maneira interessante de se incrementar o desempenho do *Quicksort* randômico é escolher um elemento médio entre *3* (ou mais) elementos selecionados aleatoriamente. Neste caso, a probabilidade de seleção do primeiro ou último elemento como *pivô* será igual a *zero*.

É importante enfatizar que o cálculo do tempo esperado de processamento nos algoritmos randômicos difere do cálculo da complexidade de caso médio nos algoritmos puramente determinísticos (vide Capítulo I). O cálculo de complexidade média é obtido analisando-se o conjunto de todas as instâncias do problema satisfazendo uma dada distribuição de probabilidade. Nos algoritmos randômicos, o tempo esperado de processamento irá depender apenas de escolhas aleatórias feitas durante o processamento e não de distribuições impostas sobre a entrada de dados.

III.3.3 – Determinação de Orientações Acíclicas em Grafos

Seja $G=(V,E)$ um grafo qualquer não-orientado. No problema da determinação de orientações acíclicas em grafos, deseja-se definir uma orientação qualquer das arestas de E de maneira que nenhum ciclo direcionado seja obtido. Este problema possui inúmeras aplicações em sistemas distribuídos onde processos que compartilham um mesmo recurso não podem operar simultaneamente. Neste caso, um par de processos i, j estará conectado se, e somente se, existir um canal de comunicação entre i e j (para maiores detalhes sobre este problema vide Arantes et. al [2002]).

III.3.3.1 – Algoritmo de Calabrese&França

Calabrese&França[1994] apresentam *dois* algoritmos de Las Vegas para este problema. Em ambos, uma moeda é jogada para cada um dos vértices de V. No primeiro, apenas moedas equilibradas são consideradas, enquanto que, no segundo, as moedas são viciadas ou polarizadas. Sempre que um nó i de V tirar *1 (p. ex.: cara)* e os demais vizinhos de i tirarem *0* (*p. ex.: coroa*) a orientação ocorrerá no sentido dos vizinhos, para o nó i. O nó i será chamado *sumidouro*. Caso ocorra algum empate, o processo será repetido novamente até que algum *sumidouro* tenha sido determinado. Os nós que contiverem alguma aresta ainda não-orientada serão chamados

probabilísticos. Caso contrário serão chamados, *determinísticos*. A velocidade de convergência (complexidade esperada) indicará o número de passos a serem executados pelo algoritmo até que todas as arestas tenham sido orientadas.

Pode-se provar facilmente que, para grafos completos o algoritmo de *Calabrese/ França*[1994] para moedas equilibradas tem complexidade esperada igual a $O(2^n)$. Para provar esse fato, considere S_i um evento indicando que o nó *i* pertencente a *V* obteve *1* e seus nós vizinhos obtiveram *0*. Observe que, no máximo, um evento S_i ocorrerá de cada vez (já que *G* é completo). Dessa forma, os eventos S_i serão disjuntos entre si. Verifique!

Seja S um evento indicando a união de *n* eventos S_i *(p/ 1=1,..,n).* Como os eventos S_i são disjuntos tem-se que:

$$\Pr(S) = \Pr\left(\bigcup_{i=1}^{n} S_i\right) = \sum_{i=1}^{n} \Pr(S_i) = \sum_{i=1}^{n} \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)^{n-1} = \frac{n}{2^n}$$

Assim, $O(2^n.n^{-1})$ repetições serão necessárias (valor esperado) até que algum dos eventos S_i ocorra. Observe que o número de repetições pode ser modelado por uma variável aleatória com distribuição geométrica e valor esperado $2^n.n^{-1}$ (vide Capítulo II). Logo, a orientação completa para cada um dos *n* vértices de V irá consumir aproximadamente $n.2^n.n^{-1} = 2^n$ passos.

A complexidade do algoritmo de *Calabrese&França[1994]* cai drasticamente quando se troca as moedas equilibradas por moedas viciadas ou polarizadas. Novamente, para grafos completos, tem-se que $Pr\{moeda_i=1\}=1/n$. Assim:

$$\Pr(S) = \Pr\left(\bigcup_{i=1}^{n} S_i\right) = \sum_{i=1}^{n} \Pr(S_i) = \sum_{i=1}^{n} \left(\frac{1}{n}\right)\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{n}{n.e} = \frac{1}{e}$$

Aproximadamente *e* repetições serão necessárias até que algum dos eventos S_i ($p/\ i \in \{1,..,n\}$) ocorra. Como $|V|=n$, a orientação completa de todas as arestas consumirá aproximadamente igual a $n.e$ passos, sendo portanto $O(n)$. Para maiores detalhes consulte Calabrese&França[1994] e Calabrese[1997].

A análise do tempo de processamento utilizando grafos completos é interessante já que, além de mais simples, um grafo completo representa na verdade, a pior instância para o problema.

III.3.3.2 - Procedimento Alg-Viz

Em Arantes et. al. [2002], resolve-se o mesmo problema em um grafo *G* qualquer com o auxílio de dados (equilibrados ou não) ao invés de moedas. Neste caso, uma moeda equilibrada (ou viciada) com *f* faces será jogada por cada um dos nós de V. Analogamente ao algoritmo de Calabrese&França[1994], a orientação só ocorrerá em direção a um nó *i* de *V*, quando *i* tirar o maior valor entre todos os seus vizinhos (nós probabilísticos). Qual será o tempo esperado de convergência neste caso?

Seja $d_i = \propto$ para $\propto \in \{0,1,2,..,f-1\}$, o resultado associado a um nó $i \in V$, obtido após jogar-se um dado equilibrado com *f* faces. É fácil ver que neste caso que $Pr(d_i = \propto) = 1/f$. Assim, o evento S_i, associado ao nó $i \in$ V ocorre, se e somente se, $d_j < d_i$, $\forall\ j \in N(i)$, onde $N(i)$ representa o conjunto dos vizinhos probabilísticos de *i*. Tem-se então o seguinte resultado preliminar:

Proposição III.1. Considere $G(V,E)$ um grafo conexo qualquer onde $|V| = n \geq 2$. Considere ainda um dado equilibrado com $f \geq 2$ faces. Se $\Delta = max\{|N(i)|: i \in V\}$ então:

$$\Pr(S_i) \geq h(\Delta, f) \quad onde \quad h(\Delta, f) = \left(\frac{1}{\Delta+1}\right)\left(1-\frac{1}{f}\right)^{\Delta+1} + \frac{1}{2f}\left(1-\frac{1}{f}\right)^{\Delta} \quad \text{e} \quad i \in V.$$

Prova: (Arantes et. al. [2002]) •

O resultado seguinte (conseqüência imediata da Proposição III.1), mostra que, se *G* é completo e o número de faces tende a infinito, a probabilidade de orientação das arestas incidentes a algum $i \in V$, tende a 1. Em outras palavras, isto significa que o empate entre dados adjacentes tende a 0.

Proposição III.2: Se *G* é completo e $f \to \infty$ então $Pr(S) \to 1$.

Prova: Como G é completo então Δ=n-1. Fazendo $f \to \infty$ na expressão de $h(\Delta,f)$ (Proposição III.1), tem-se $Pr(S_i) \to 1/n$ e portanto $Pr(S) \to 1$. •

Seja *S* um evento indicando a união de *n* eventos distintos S_i, ou seja, *S* ocorre se S_i ocorre para algum $i \in \{1,..,n\}$. Cabe notar que, se G é um grafo qualquer, os eventos S_i, não são necessariamente disjuntos. Esta situação é bem esquematizada na Figura III.3 seguinte. Para simplificar o exemplo, foi utilizado um dado com apenas *f=2* faces (moeda). Os valores obtidos em cada face estão representados ao lado de cada nó:

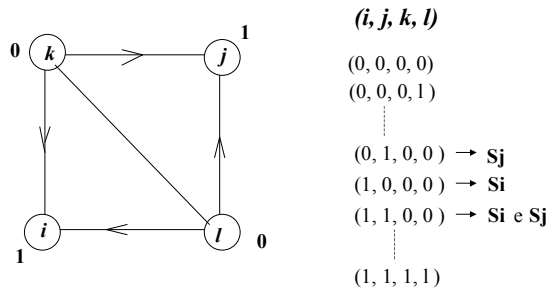


Figura III.3: Procedimento Alg-Viz.

Observe que o evento elementar (1,1,0,0) pertence aos eventos S_i e S_j respectivamente, ou seja, $S_i \cap S_j = \varnothing$. Como constatado mais adiante na Proposição III.3 este tipo de situação cria dificuldades na análise de complexidade do Alg-Viz, já que, no caso geral, a propriedade de aditividade finita não pode ser utilizada (vide definição II.3, Capítulo II).

Tem-se então o seguinte resultado:

Proposição III.3: (Arantes, França, Martinhon [2002])

Considere *G(V,E)* um grafo conexo qualquer, onde $|V| = n \geq 2$. Além disso, considere Δ o grau máximo de G, e um dado equilibrado com $f = \Delta+1$ faces. O algoritmo *Alg-Viz* terá tempo esperado de processamento igual a $O((\Delta+1).e)$.

Prova: Temos que $f=\Delta+1$(hipótese). Segue então da Proposição III.1 que:

$$\Pr(S_i) = \Pr(A_i) \geq \left(\frac{1}{\Delta+1}\right)\left(1-\frac{1}{\Delta+1}\right)^{\Delta}\left(1-\frac{1}{\Delta+1}\right) + \frac{1}{2(\Delta+1)}\left(1-\frac{1}{\Delta+1}\right)^{\Delta}$$

onde o evento S_i (associado a $n_i \in V$) ocorre, se e somente se, $d_j < d_i$, $\forall\ j \in N(i)$, e $A_i \in S_i$ é evento elementar onde $Pr(d_k = 0)=1$, $\forall\ k \in R_i = V-(\{i\} \cup N(i))$, ou seja, $Pr(A_i|S_i)=1$. Observe que o evento elementar A_i pode ser definido sem perda de generalidade. Assim, substituindo $(1-1/(\Delta+1))^{\Delta} \geq 1/e$, na desigualdade acima obtêm-se:

$$\Pr(A_i) \geq \left(\frac{1}{\Delta+1}\right)\left(\frac{3}{2}-\frac{1}{\Delta+1}\right)\left(\frac{1}{e}\right)$$

Como G é um grafo conexo e $n \geq 2$ (hipótese), conclui-se diretamente que $\Delta \geq 1$. Além disso, como consideramos $Pr(A_i|S_i)=1$, o evento S irá representar (sem perda de generalidade), a união dos eventos disjuntos S_i (p/ i=1,2,..,n) e será dado pela seguinte expressão:

$$\Pr(S) = \Pr\left(\bigcup_{i=1}^{n} S_i\right) = \sum_{i=1}^{n} \Pr(A_i) \geq \frac{n}{(\Delta+1).e}$$

Note agora que $(\Delta+1).e.\ n^{-1}$ repetições (valor esperado) serão necessárias até que pelo menos um evento S_i ocorra. Como temos $|V| = n$, segue que a complexidade esperada será igual a $O((\Delta+1)e)$. •

Este resultado é bastante interessante já que expressa a complexidade de tempo independentemente do número de vértices e arestas.

Como conseqüência imediata da proposição anterior tem-se o seguinte resultado:

Proposição III.4: Se G é completo então *Alg-Viz* terá complexidade esperada igual a $O(n.e)$. •

Observe que, se G é completo, o algoritmo de Calabrese&França[1994] para moedas polarizadas e o algoritmo *Alg-Viz* para dados com n faces não-polarizados possuem complexidade esperada $O(n.e)$. Obviamente, ao contrário do que ocorre com Calabrese&França[1994], o desempenho do *Alg-Viz* pode ser incrementado aumentando-se o número de faces do dado. Para maiores detalhes sobre o *Alg-Viz* com dados polarizados consulte Arantes et. al [2002].

III.3.3.3 - Procedimento Alg-Arestas

No procedimento *Alg-Arestas* (Arantes et. al. [2002]), analogamente ao *Alg-Viz*, um dado equilibrado é jogado por cada um dos vértices de V. Neste caso, para cada $[i,j] \in E$, se $d_i > d_j$, orienta-se a aresta $[i,j]$ de j para i. No caso de empate entre os dados as arestas não são orientadas e o processo é repetido até que todas as arestas de G tenham sido orientadas.

Como provado em Arantes et. al.[2002], o procedimento *Alg-Arestas* sempre gera uma orientação acíclica em G com probabilidade 1 (método de Las Vegas).

É bastante fácil perceber que *Alg-Arestas* tem convergência muito mais rápida que *Alg-Viz*. Neste caso, o tempo esperado de convergência será inferior a $O(n)$ mesmo considerando-se grafos completos!

A idéia da prova de complexidade consiste no seguinte: em um sorteio, a probabilidade de empate entre dois vizinhos é de $1/f$. Assim, enquanto o número de arestas é grande, esta é a proporção de empates ocorridos a cada passo do algoritmo. Assim, em cada passo, uma proporção de $1/f$ das arestas não orientadas até aquele passo permanecem sem serem orientadas. Portanto, a convergência é dada por $O(\lceil \log_f m \rceil)$, onde m é o número de arestas do grafo original $G=(V,E)$.

O resultado seguinte tem uma prova ligeiramente diferente daquela apresentada em Arantes et. al.[2002]. Em ambos os casos, entretanto, a complexidade será igual a $O(\lceil \log_f m \rceil)$.

Proposição III.5: O procedimento *Alg-Arestas* para dados equilibrados com f faces tem complexidade esperada $O(\lceil \log_f m \rceil + 1)$.

Prova: O processo de orientação das arestas no *Alg-Arestas* pode ser descrito por uma relação de recorrência do tipo $T(m)=a(m)+T(h(m))$, onde $m = f^k$ e $k \in Z^+$ (m é potência de f), $a(m)$ é função inteira não-decrescente e não-negativa de m, e $h(m)$ é variável aleatória assumindo valores inteiros no intervalo $[0,m]$. A função $a(.)$ representa, na verdade, o esforço gasto na orientação de m arestas. Observe que, se todos os dados são jogados simultaneamente, então $a(m)=1$ (função constante). O total de arestas não orientadas nesta iteração será representado por $h(m)$.

Cada aresta é orientada independentemente das outras. Portanto, pode-se definir uma variável aleatória de Bernoulli (vide Capítulo II) que modela cada tentativa de orientação de uma aresta. A probabilidade de fracasso, ou seja, de a aresta não se orientar, será igual à de dar empate na disputa entre dois dados equilibrados de número de faces iguais a f, ou seja, $1/f$. Assim, $1/f$ representa a probabilidade de fracasso e $1-1/f$ a probabilidade de sucesso.

Note que número total de arestas orientadas a cada iteração (somatório dos ensaios de Bernoulli) pode ser representado por uma variável aleatória com distribuição binomial e valor esperado $m(1-1/f)$. Logo, $E(h(m)) = m/f$, irá representar o número esperado de arestas não orientadas após a primeira iteração. Desta forma, será bem razoável substituir a variável aleatória $h(m)$ por $E(h(m))$ na expressão de recorrência probabilística. Desenvolvendo a nova expressão de recorrência tem-se que:

$$T(m) = 1+T(m/f) = 2+ T(m/f^2) = 3+ T(m/f^3), \quad etc.$$

Após k passos obtém-se $T(m)=k+ T(m/f^k)$. O processo é repetido até que $m/f^k=1$, ou ainda $k= \log_f m$ (pois m é potência de f). Como $T(1)=1$ (base da recursão) é fácil ver que $T(m) = \log_f m+1$.

Para m qquer, basta notar que $f^{k-1} \leq m \leq f^k$ (m estará entre *duas* potências consecutivas de f). Da segunda parte da desigualdade tem-se que:

$$T(m) \leq T(f^k) = k+1. \qquad (1)$$

Da primeira parte da desigualdade tem-se que $k \leq \log_f m+1$. Como k é inteiro positivo, é fácil ver que, $k \leq \lceil \log_f m \rceil$. Substituindo esta última desigualdade em (1) conclui-se finalmente que $T(m) \leq \lceil \log_f m \rceil + 1$. •

Para maiores detalhes sobre os procedimentos de Las Vegas aqui apresentados vide Arantes et. al. [2002]. Nele, a demonstração da Proposição III.5 é realizada de maneira um pouco diferente e se baseia nos conceitos de recorrência probabilística apresentados inicialmente por Karp[1994].

III.4 - MÉTODO DE MONTE CARLO

Como relatado em Kalos e Whitlock [1986], o nome Monte Carlo foi utilizado, inicialmente, na simulação de sistemas físicos por cientistas que trabalhavam no desenvolvimento de armas nucleares em Los Alamos/Estados Unidos. Por volta de 1940, pesquisadores como Von Neuman, Fermi, Ulam, Metropolis juntamente com o surgimento dos modernos computadores digitais deram grande impulso a trabalhos que utilizavam o método de Monte Carlo em aplicações como a mecânica estatística, transporte de material radioativo, economia e outras áreas (vide Donsker e Kac [1949], Metropolis e Ulam[1951], Householder et al.[1951]). Atualmente, o termo Monte Carlo possui uma acepção mais ampla, sendo difícil defini-lo exatamente. Em nosso trabalho, o significado de Monte Carlo será diferente daquele utilizado na simulação de sistemas físicos.

A utilização do método de Monte Carlo, como discutido aqui, não garante solução exata na resolução de um determinado problema. Pode-se dizer apenas que uma solução correta é obtida

segundo uma probabilidade de acerto que poderá ser definida a *priori*. Em contrapartida, consegue-se, normalmente, um tempo de processamento inferior àquele obtido no método de Las Vegas.

Para problemas de decisão (vide Capítulo I), existem dois tipos de algoritmo de Monte Carlo: algoritmos com *erro unilateral* (*one-sided error*) e algoritmos com e*rro bilateral* (*two-sided error*). Um algoritmo de Monte Carlo tem erro unilateral, se a probabilidade de fracasso é *zero* para, pelo menos, uma das saídas *Sim* ou *Não.* Se a probabilidade de falha é não-nula para as saídas *Sim* e *Não*, o algoritmo possui erro bilateral. Observe por exemplo que o método de Las Vegas (aplicado a problemas de decisão) possui erro *zero* para *Sim* e *Não* respectivamente.

A seguir, são apresentados alguns exemplos ilustrando algumas aplicações do método de Monte Carlo.

III.4.1 - Seleção de um elemento entre os 50% maiores de uma lista

Este exemplo bastante simples, sintetiza alguns dos principais conceitos presentes no método de Monte Carlo. Considere $A = \{x_1, x_2, .., x_n\}$ uma lista desordenada com *n* valores inteiros. Deseja-se selecionar um elemento que esteja entre os $\lceil n/2 \rceil$ maiores de *A*. Este problema pode ser resolvido simplesmente com a seleção do maior elemento da lista. Neste caso, *n-1* comparações serão necessárias. Uma outra opção, mais interessante, é executar apenas $\lceil n/2 \rceil - 1$ comparações salvando-se sempre o maior elemento. Ao final do processo obtêm-se, certamente, um elemento entre os 50% maiores de *A*. Nos dois casos, pode-se garantir uma solução exata para o problema com probabilidade de erro igual a *zero*. Entretanto, se a probabilidade de falha for arbitrariamente pequena, mas não nula, talvez se consiga uma redução ainda maior do número total de comparações realizadas.

Sem perda de generalidade considere *n* par, e $B \subset A$, o conjunto dos n/2 maiores elementos de *A*. Suponha que 2 valores x_i , $x_j \in A$ sejam obtidos aleatoriamente com distribuição uniforme e que $x_i \geq x_j$. A probabilidade de cada um desses valores pertencer a *B* será igual a *1/2*. Logo, a probabilidade de ambos não pertencerem será exatamente *1/4* (probabilidade de fracasso). Como $x_i \geq x_j$, pode-se dizer, equivalentemente, que *1/4* é a probabilidade de x_i não pertencer a *B*, ou ainda, que x_i pertence a *B* com probabilidade *3/4*. Portanto, selecionando-se o maior valor entre x_i e x_j, pode-se garantir uma solução com 75% de chance de sucesso. Observe que esta margem de acerto foi assegurada para apenas 2 valores gerados aleatoriamente!

O procedimento acima pode ser estendido para *k* valores aleatórios gerados uniformemente. Assim, se *x** for o maior entre *k* valores selecionados, a probabilidade de falha será igual a $1/2^k$! Equivalentemente, *x** pertencerá a *B* com probabilidade $1-1/2^k$*!* Note por exemplo que, para *k=10* tem-se uma probabilidade de sucesso igual a *0,999.* Para *k=20* esta probabilidade sobe para *0,999999*! Se A contém *1000* elementos, o algoritmo determinístico discutido acima irá executar *500* comparações para resolução do problema, ao passo que, no algoritmo randômico, *20* elementos gerados aleatoriamente já garantem uma solução correta com probabilidade de falha igual a *0,000001!* A probabilidade de um erro em um programa ou uma máquina por exemplo superam, em muito, estes valores.

Assim, se *k* tentativas forem realizadas, a probabilidade de falha diminui exponencialmente à medida que *k* aumenta linearmente! Apesar de possível, a determinação de uma solução exata utilizando o método de Las Vegas não é interessante neste caso. Para verificar se a solução gerada está ou não correta, serão necessários mais *O(n)* comparações! Se a solução estiver incorreta, todo o processo deverá ser repetido novamente. O método de Monte Carlo irá retornar uma solução muito mais atrativa para o problema.

Outro aspecto interessante a ser observado no exemplo acima é que, se a probabilidade de erro $\varepsilon > 0$ no método de Monte Carlo for definida a *priori*, então $k = \lceil \log(1/\varepsilon) \rceil$ elementos deverão ser gerados aleatoriamente com distribuição uniforme. A complexidade neste caso, será igual a $O(\log(1/\varepsilon))$ passos e, portanto, polinomial em $1/\varepsilon$.

III.4.2 - O problema do Corte-Mínimo em Multigrafos

Considere $G=(V,E)$ um grafo conexo não-orientado com n vértices e m arestas. Em algumas situações será interessante ampliar a definição de grafos para multigrafos. Um grafo G define um *multigrafo* se não contiver laços[1] e se existirem arestas paralelas entre, pelo menos, um par de vértices $i,j \in V$. No caso de grafos sem a presença de laços e arestas paralelas (também conhecidos por *grafos simples*), valores inteiros positivos associados a cada aresta $[i,j]$ poderão representar o número de arestas entre os vértices i e j no multigrafo associado. Desta forma, o problema do corte mínimo em multigrafos pode ser visto como um problema de corte mínimo em grafos (grafos simples) com pesos positivos nas arestas. Como observado por Motwani e Raghavan[1995], se G admite arestas com pesos negativos então o problema se torna NP-Completo.

Um *corte* em um multigrafo G é um conjunto de arestas cuja remoção implica na divisão de G em duas ou mais componentes conexas[2]. No problema do corte-mínimo em multigrafos deseja-se determinar o corte de menor cardinalidade. Mais formalmente, espera-se encontrar uma partição[3] (V_1,V_2) de V de maneira que a cardinalidade do corte $c(V_1,V_2) = \{e=[u,v]$ tal que $u \in V_1$, $v \in V_2$ ou $v \in V_1$, $u \in V_2\}$ induzido pela partição seja minimizada.

Uma versão mais simples deste problema, aqui representada por *corte-mínimo(s,t)*, pode ser obtida fixando-se dois vértices $s, t \in V$. Neste caso, o objetivo será encontrar um corte mínimo $c(V_1,V_2)$ tal *que* $s \in V_1$ e $t \in V_2$. O problema do corte mínimo original pode então ser resolvido através de $O(n-1)$ chamadas ao procedimento corte-mínimo(s,t) (verifique). Uma das maneiras de se resolver o corte-mínimo(s,t), é computando-se o fluxo máximo entre s e t respectivamente (vide Ahuja et. al[1993]). Neste caso, os pesos associados a cada aresta representam capacidades. Ford& Fulkerson[1956] provaram que o valor do fluxo máximo é igual à capacidade do corte-mínimo entre s e t. Uma vez encontrado o fluxo máximo, o corte mínimo pode ser obtido diretamente.

Atualmente, *dois* dos melhores algoritmos determinísticos para o problema do fluxo máximo são devidos a King et. al.[1993] e Philips e Westbrook[1991], ambos de complexidade $O(nmlog(n^2/m))$. Felizmente, as $n-1$ repetições do fluxo máximo entre s e t, necessárias para a resolução do problema do corte mínimo original são desnecessárias, e podem ser implementadas atráves de uma única chamada ao procedimento. Tem-se portanto, um algoritmo determinístico para o problema do corte mínimo em multigrafos (ou grafos simples com pesos nas arestas) de complexidade é igual a $O(nmlog(n^2/m))$ (Motwani&Raghavan[1995]).

Nesta seção será apresentado um algoritmo randômico de complexidade $O(n^4logn)$ que utiliza o conceito de contração de arestas (Karger[1993]). Karger&Stein[1993] se baseiam neste procedimento para construção de um novo algoritmo randômico de complexidade $O(n^2$

O algoritmo seguinte (desenvolvido por Karger[1993]) aplica sucessivas contrações de arestas visando a determinação do corte-mínimo. O processo de contração é repetido até que apenas 2 vértices *i* e *j* sejam obtidos. Portanto, *n-2* contrações serão necessárias até o final do processo. O número total de arestas entre *i* e *j* será uma solução candidata para o corte mínimo em *G*. As chances de obtenção de um corte mínimo por este método são incrementadas repetindo-se o processo um número pré-determinado de vezes (representado no algoritmo pela constante *T*).

Seja *Gi* o multigrafo obtido após *i* contrações de *G*. O conjunto *CM* (vide Algoritmo III.2),s rm(.)7.3(a)-8.6(z)1
nd(e)14.(to)14(n3(co)27.5(

$$\Pr\left(\bigcap_{i=1}^{n-2} \mathrm{E}_i\right) = \Pr(\mathrm{E}_1).\Pr(\mathrm{E}_2|\mathrm{E}_1).\Pr(\mathrm{E}_3|(\mathrm{E}_1 \cap \mathrm{E}_2)).....\Pr\left(\mathrm{E}_{n-2}|\bigcap_{i=1}^{n-3} \mathrm{E}_i\right)$$

Note que a expressão anterior (representando a probabilidade de sucesso), indica que nenhuma das arestas pertencentes ao corte mínimo foi selecionada ao final das *n-2* contrações.

O grau do vértice $v \in V$ (representado por *d(v))* é igual ao número de arestas incidentes a *v*. Se *c* representa o tamanho do corte mínimo em *G* então:

$$m = \frac{\sum_{v \in V} d(v)}{2} \geq \frac{cn}{2}$$

Para constatar esse fato, basta notar que cada aresta do grafo contribui com 2 vértices distintos, cada um deles de grau 1. Como cada um dos *n* vértices tem grau maior ou igual a *c*, segue que *cn/2* define um limite inferior para o número de arestas em *G*.

Sem perda de generalidade suponha que *G* contenha um único corte mínimo de cardinalidade *c*. Como $m \geq cn/2$, pode-se afirmar que, se E_1^c representa o evento c0 5.4 2/TT14443emít m

por exemplo o seguinte resultado (proveniente do cálculo) onde se tem $1+x<e^x, \forall x \in \Re$. Fazendo-se $x = -2/n^2$ chega-se a $n^2/2 = -1/x > 0$. Assim, se $T=O((n^2/2).\delta)$ repetições do Algoritmo III.2 forem executadas tem-se:

$$(1+x)^{-\delta/x} < (e^x)^{-\delta/x} = e^{-\delta}$$

Agora, fazendo $\delta = logn$ obtêm-se um novo algoritmo com complexidade igual a $O(n^4 logn)$ e probabilidade de sucesso maior ou igual a $1-1/e^{logn}$! Por exemplo, para $n=8$ a probabilidade de sucesso será superior a *95%,* para $n = 64$ será superior a 99.7% (verifique)!

Karger&Stein [1993], apresentam uma versão mais sofisticada deste algoritmo onde o tempo esperado de processamento cai para $O(n^2 logn)$ passos! Maiores detalhes sobre este procedimento podem ser encontrados em Karger[1995] e Motwani& Raghavan[1995].

III.4.3 - Amostra Randômica (Random Sampling)

Para uma grande quantidade de aplicações, é possível gerar um pequeno e representativo subproblema do problema original através da técnica de *Amostra Randômica.* Nela, essencialmente, o objetivo será tratar a amostra selecionada a um baixo custo computacional, e estender resultados e conclusões obtidas sobre a amostra para o problema original. Em um problema de otimização combinatória, por exemplo, pode ocorrer que uma solução ótima de um determinado subproblema (amostra) esteja suficientemente próxima de uma solução ótima do problema original. Em muitas situações, pode-se ainda refinar (quando possível) a solução obtida até que uma solução exata seja obtida.

O trabalho de D.Karger[1995][4] é ótima referência sobre a utilização de Amostra Randômica na solução de problemas de otimização combinatória. O exemplo seguinte, embora não se enquadre neste grupo de problemas, ilustra bem o potencial desta técnica.

III.4.3.1 - Seleção Randômica

Considere S uma lista com n elementos inteiros e distintos *dois-a-dois*. Neste problema, deseja-se identificar o k-ésimo menor elemento de S, para algum $k \in \{1,..n\}$. De maneira geral, pode-se considerar S formada por elementos quaisquer não necessariamente distintos entre si. Isto pode ocorrer sempre que os elementos de S pertencerem a um conjunto universo Γ satisfazendo uma relação de ordem total. Um algoritmo determinístico de complexidade $O(nlogn)$ bastante natural para este problema consiste simplesmente na ordenação de todos os n elementos de S e na seleção direta do k-ésimo menor elemento. O melhor algoritmo determinístico conhecido para este problema executa $O(3n)$ comparações. Os trabalhos de M. Blum et al.[1973] e Schönhage et al.[1976] são duas referências clássicas sobre este assunto.

O algoritmo randômico aqui apresentado (conhecido na literatura como *LazySelect*), foi baseado no procedimento recursivo de Floyd&Rivest[1975] e tem tempo esperado de processamento igual a $2n+o(n)$.

No procedimento *LazySelect* (Figura III.6), considere $S(i)$ o *i-ésimo* menor elemento de S e $r_S(j)$ a posição ocupada por um elemento j na lista S. Assim, $S(r_S(j))=j$ e $r_S(S(i))=i$ (verifique!). Esta notação será também estendida às listas R e P respectivamente. Note que, no algoritmo da Figura III.6 deseja-se calcular $S(k)$. Entretanto, qual será a probabilidade de sucesso e o tempo de execução em uma única iteração do algoritmo? Ainda, caso a probabilidade de sucesso seja 1, qual o tempo

[4] Por sua tese intitulada "*Random Sampling in Graph Optimization Problems*", D. Karger recebeu o ACM Doctoral Dissertation award em 1994 e o Tucker Prize em 1997.

esperado de processamento? Questões como esta serão discutidas mais adiante. A atribuição de valores dados às variáveis *x,l* e *h* também será analisada posteriormente.

Procedimento III.3: *LazySelect(k,S)*
Início
- Constrói uma nova lista R selecionando $O(n^{3/4})$ elementos de *S* aleatoriamente e com reposição;
- **Repita**
  - Ordena *R* em tempo $O(n^{3/4}\log n)$;
  - Calcule:
    $x \leftarrow kn^{-1/4}$,
    $l \leftarrow max\{\lfloor x-n^{1/2} \rfloor, 1\}$;
    $h \leftarrow min\{\lceil x+n^{1/2} \rceil, n^{3/4}\}$;
  - $a \leftarrow R(l)$ e $b \leftarrow R(h)$;
  - Compare *a* e *b* com cada elemento de S retornando $r_S(a)$ e $r_S(b)$ respectivamente;
  - **Se** $k < n^{1/4}$ **então** $P \leftarrow \{y \in S / y \le b\}$
    senão
    - **Se** $k > n-n^{1/4}$ **então** $P \leftarrow \{y \in S / y \ge a\}$
      senão
      Se $k \in [n^{1/4}, n- n^{1/4}]$ **então** $P \leftarrow \{y \in S / a \le y \le b\}$;

    fim;
- **Até que** $(S(k) \in P)$ e $(|P| \le 4n^{3/4}+2)$;
- Ordena *P* em $O(|P|\log|P|)$ passos;
- Retorne $S(k) \leftarrow P(k - r_S(a) + 1)$;

fim.

Figura III.6: Procedimento *LazySelect* - Las Vegas

Este procedimento, como discutido mais adiante, sempre retorna uma solução correta, sendo portanto uma aplicação do método de Las Vegas. Entretanto, se apenas uma iteração for realizada tem-se o método de Monte Carlo com probabilidade de sucesso superior a $1-O(n^{-1/4})$ e tempo esperado de processamento igual a $2n+o(n)$ (Proposição III.6).

Os elementos de *R* são selecionados aleatoriamente e com reposição, ou seja, um elemento de *S* pode ser selecionado mais de uma vez na composição da lista *R*. Apesar de tornar implementação e análise mais complicadas, uma seleção sem reposição pode também ser considerada (Motwani&Raghavan[1995]).

A idéia básica do algoritmo consiste na determinação de elementos $a,b \in S$ tais que, com alta probabilidade, $S(k)$ pertença à lista *P* e *P* possa ser ordenado a um baixo custo computacional (em tempo $o(n)$). Deve-se buscar, portanto, uma minimização da probabilidade de falha de uma dada iteração. Isto será feito utilizando-se a desigualdade de Tchebyshev (Capítulo II).

Note que os procedimentos: ordena *R* e ordena *P* tem complexidade $o(n)$ (verifique!). Para determinação de $r_S(a)$ e $r_S(b)$ serão necessárias $O(2n)$ comparações. Segue então que o tempo de processamento da primeira iteração será igual a $T(n)=2n+o(n)$.

Para analisar a probabilidade de falha ao final da primeira iteração, considere o caso onde $k \in [n^{-1/4}, n- n^{1/4}]$. As situações onde $k < n^{-1/4}$ e $k > n- n^{-1/4}$ são análogas e deixadas como exercício. A Figura III.7 ilustra as principais etapas presentes no algoritmo.

Note que o procedimento *LazySelect* falha na 1ª primeira iteração quando:

1) $S(k)<a$ ou $S(k)>b$;
2) $|P| > 4n^{3/4} + 2$ onde $P = \{y \in S / a \le y \le b\}$.

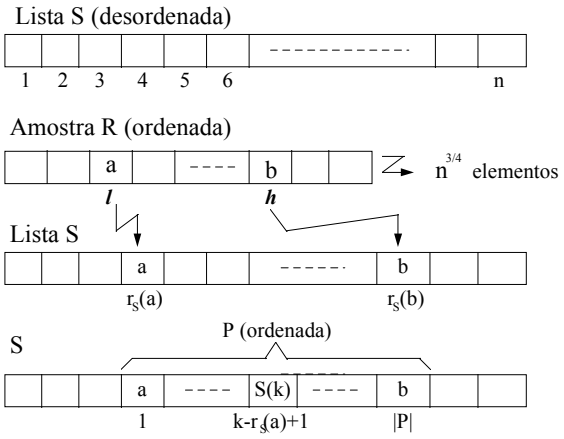


Figura III.7 Procedimento *LasySelect* para $k \in [n^{1/4}, n - n^{1/4}]$

Sejam A e B dois eventos associados ao caso (1). Mais precisamente, A ocorre se $S(k)<a$ e B ocorre se $S(k)>b$. Se $S(k)<a$, menos do que l elementos (em R) serão menores ou iguais a $S(k)$. Analogamente, se $S(k)>b$ então mais do que h elementos em R serão menores ou iguais a $S(k)$. Note que, no caso de falha, o elemento $S(k)$ poderá não pertencer a lista R.

Seja X_i uma variável aleatória de Bernoulli onde:

$$X_i = \begin{cases} 1, & se\ o\ i-esimo\ elemento\ de\ R\ e'\ no\ maximo\ S(k) \\ 0, & caso\ contrario \end{cases}$$

Segue então que $Pr(X_i=1) = k/n$ e $Pr(X_i=0) = 1- k/n$, já que o i-ésimo elemento da amostra é um elemento arbitrário em S. Conseqüentemente, $X_1, X_2, ..., X_m$ são variáveis aleatórias independentes onde $\mu_{Xi} = E(X_i) = k/n$ e $\sigma_{Xi}^2 = (k/n).(1-k/n)$ para $i=1,.., n^{3/4}$ (vide Capítulo II).

Seja X outra variável aleatória igual a:

$$X = \sum_{i=1}^{n^{3/4}} X_i \cdot$$

O valor de X representa o número total de elementos que são no máximo iguais a $S(k)$, ou seja, $Pr(S(k) < a) = Pr(X < l)$. Sem perda de generalidade considere $Pr(S(k) < a) = Pr(X \leq l)$. Da mesma forma pode-se concluir que: $Pr(S(k) > b) = Pr(X \geq h)$.

A variável aleatória X define uma distribuição binomial com parâmetros $p=k/n$ e $n^{3/4}$ respectivamente (vide Capítulo II). Assim, valor esperado μ_X, variância σ_X^2 e desvio-padrão σ_X serão iguais a:

$$\mu_X = \frac{kn^{3/4}}{n} = kn^{-1/4}, \qquad \sigma_X^2 = n^{3/4}\left(\frac{k}{n}\right)\left(1-\frac{k}{n}\right) \leq \frac{n^{3/4}}{4}, \qquad \sigma_X \leq \frac{n^{3/8}}{2} \cdot$$

Observe que $E(X) = \mu_X$ foi obtido da linearidade do valor esperado e σ_X^2 (variância) foi calculada utilizando-se a propriedade P20 do Capítulo II. Ainda, na expressão de σ_X^2 tem-se que $(k/n)(1-k/n) \leq 1/4$ (verifique).

Da desigualdade de Tchebyshev tem-se:

$$\Pr\left(|X - \mu_X| \geq t\sigma_X\right) \leq \frac{1}{t^2}, \quad \forall\, t \in \Re^+ .$$

Se $X - \mu_X < 0$ então $Pr(X \leq \mu_X - t\sigma_X) \leq 1/t^2, \ \forall t \in \Re^+$. Fazendo $l = \mu_X - t\sigma_X$ e $t = 2n^{1/8}$ tem-se, através da substituição de μ_X e σ_X que:

$$\Pr(X \leq l) = \Pr\left(X \leq kn^{-1/4} - \sqrt{n}\right) \leq \left(4n^{1/4}\right)^{-1}.$$

Analogamente, se $X - \mu_X \geq 0$, tem-se $Pr(X \leq \mu_X + t\sigma_X) \leq 1/t^2, \ \forall t \in \Re^+$. Fazendo $h = \mu_X + t\sigma_X$ e $t = 2n^{1/8}$ chega-se a:

$$\Pr(X \geq h) = \Pr\left(X \geq kn^{-1/4} + \sqrt{n}\right) \leq \left(4n^{1/4}\right)^{-1}.$$

Como $Pr(A) = Pr(X \leq l)$ e $Pr(B) = Pr(X \geq h)$ são mutuamente excludentes segue então que $Pr(A \cup B) = Pr(A) + Pr(B) = O(n^{-1/4})$.

Como $t = 2n^{1/8}$ então $t\sigma_X = n^{1/2}$. Desta forma, tem-se uma sub-lista $[l,h]$ de tamanho $2n^{1/2}$ (lista R) em torno do valor esperado $kn^{-1/4}$ como provável localização de $S(k)$. A Figura III.8 ilustra esta situação. Obviamente, outros valores podem ser atribuídos a t. Entretanto, como $r_S(a)$ e $r_S(b)$ são funções de l e h respectivamente, é importante observar o que ocorre com o tamanho da sub-lista $P = [a,b] \subseteq S$. Quanto maior o tamanho de $[a,b]$, maior a complexidade envolvida na ordenação de P.

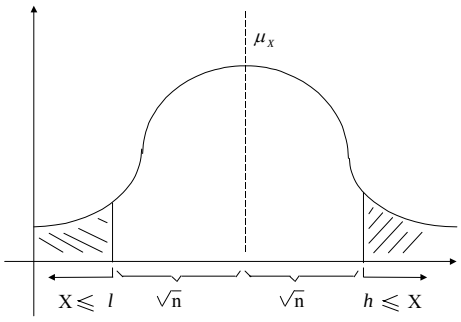


Figura III.8: Desigualdade de Tchebyshev

Outra possibilidade de falha ocorre quando $|P| > 4n^{3/4}+2$.

A estratégia neste caso será considerar u e v (onde $1 \leq u \leq v \leq n$) de maneira que*ura-11.2(qs)1er21.0449*

$$Y_i = \begin{cases} 1, & se\ o\ \ i-esimo\ elemento\ de\ R\ e'\ menor\ que\ S(u) \\ 0, & caso\ contrario \end{cases}$$

Assim $Pr(a < S(u)) = Pr(Y \geq l)$ onde $Y = \sum_{i=1}^{n^{3/4}} Y_i$.

Para aplicação da desigualdade de Tchebyshev, calcula-se valor esperado e desvio-padrão respectivamente. Logo:

$$\mu_Y = E(Y) = \sum_{i=1}^{n^{3/4}} E(Y_i), \quad \text{onde } E(Y_i) = 1.p(Y_i = 1) + 0.p(Y_i = 0)$$

Como $Pr(Y_i=1) = u/n$ e $Pr(Y_i=0) = 1- u/n$ tem-se que:

$$\mu_Y = \frac{un^{3/4}}{n} = un^{-1/4}, \qquad \sigma_Y^2 = n^{3/4}\left(\frac{u}{n}\right)\left(1-\frac{u}{n}\right) \leq \frac{n^{3/4}}{4}, \qquad \sigma_Y \leq \frac{n^{3/8}}{2}.$$

Assim, $Pr(Y \geq l) = Pr(Y \geq kn^{-1/4} - n^{1/2}) = Pr(Y - kn^{-1/4} \geq -n^{1/2}) = Pr(Y-kn^{-1/4}+2n^{1/2} \geq n^{1/2})$. Fazendo $\mu_r = kn^{-1/4} - 2n^{1/2} = u.n^{-1/4}$, conclui-se que: $u = k - 2n^{3/4}$. Para evitar que u seja menor que 1 basta fazer $u = max\{1, k - 2n^{3/4}\}$.

Analogamente, seja W_i uma variável aleatória associada ao *i-ésimo* elemento da amostra R onde:

$$W_i = \begin{cases} 1, & se\ o\ \ i-esimo\ elemento\ de\ R\ e'\ menor\ que\ S(v) \\ 0, & caso\ contrario \end{cases} \qquad e \qquad W = \sum_{i=1}^{n^{3/4}} W_i.$$

Da mesma forma, se $S(v)<b$ então $W \leq v$ representa outra possibilidade de falha. Logo, $Pr(W \leq v) = Pr(W \leq kn^{-1/4} + n^{1/2}) \leq (4n^{1/4})^{-1} \Rightarrow Pr(kn^{-1/4} - W \geq -n^{1/2}) = Pr(kn^{-1/4} + n^{1/2} - W \geq n^{1/2}) \leq (4n^{1/4})^{-1}$ (desigualdade de Tchebyshev). Fazendo $\mu_W = kn^{-1/4}+2n^{1/2} = v.n^{-1/4}$ conclui-se *que* $v= k+2n^{3/4}$. Para evitar v maior do que n faz-se $v = min\{k+2n^{3/4},n\}$.

Após a substituição de u e v tem-se que: $|P| = (k + 2n^{3/4}+1) - (k - 2n^{3/4} - 1) + 1 = 4n^{3/4} + 3$. Logo, o algoritmo falha sempre que $|P|>4n^{3/4}+2$. Portanto, $Pr(|P|>4n^{3/4}+2) = Pr(a<S(u)) + Pr(b>S(v)) \leq O(n^{-1/4})$. Provou-se então o seguinte resultado:

Proposição III.6: O algoritmo *LazySelect* encontra $S(k)$ na primeira iteração do procedimento e tem probabilidade de sucesso superior a $1- O(n^{-1/4})$ e tempo esperado igual a $2n+o(n)$. •

Como conseqüência direta desse resultado, pode-se mostrar que o tempo de processamento associado ao método de Las Vegas é igual a $2n + o(n)$ (verifique!).

Versões mais sofisticadas deste procedimento podem ser implementadas em um tempo esperado de processamento igual a $1.5n+o(n)$ iterações (Motwani&Raghavan[1995]).

III.4.4 – Impressão Digital (Fingerprinting)

Pode-se encontrar facilmente, uma grande quantidade de situações onde a comparação entre objetos x, y de um dado conjunto U é necessária. Na técnica da impressão digital (*fingerprinting*), utiliza-se normalmente, um rótulo (impressão digital) associado a cada objeto de U, de maneira que, se uma comparação entre rótulos for realizada, e se os rótulos associados forem iguais, então é bem provável que os objetos originais x , y sejam iguais.

Os rótulos são gerados, normalmente, utilizando-se uma função $f : U \rightarrow V$, onde V representa o conjunto de possíveis rótulos. Caso $f(.)$ seja definida a *priori*, ou seja, se cada elemento do

domínio é associado unicamente a um determinado elemento da imagem, pode ocorrer que um "adversário" exiba uma situação indesejada onde se tenha $f(x) = f(y)$, mas $x \neq y$. Este problema pode ser minimizado gerando-se $f(.)$ randômicamente, ou seja, cada repetição do algoritmo produzirá novas imagens associadas a um mesmo ponto do domínio. Desta forma, se $f(x) \neq f(y)$ pode-se concluir diretamente que $x \neq y$. Entretanto, se $f(x) = f(y)$, deseja-se que $Pr(x=y)$ seja suficientemente grande (próxima de *um*).

Outra característica importante é que o conjunto de rótulos V seja menor que o conjunto original U. Assim, se existir uma relação de ordem sobre os elementos de U, saber se x é igual a y irá consumir tempo $O(log|U|)$. Por outro lado, saber se $f(x)$ é igual a $f(y)$ terá complexidade $O(log|V|)$ (vide Figura III.10).

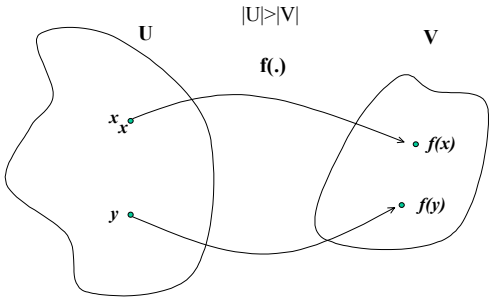


Figura III.10 : Impressão Digital (*Fingerprinting*)

Um exemplo de aplicação bastante simples que ilustra a utilização da técnica *Fingerprint* é o chamado problema da Determinação de Padrões (*Pattern Matching*). Este problema é bastante importante em processamento de texto. Nele, são dadas uma *string* de n caracteres (chamada padrão) e uma outra *string* com m caracteres, normalmente muito maior, representando um *texto* qualquer. O problema consiste em determinar se a *string* ocorre ou não em alguma parte do texto. A utilização de força-bruta implicaria em um algoritmo de complexidade $O(nm)$ passos no pior caso. Uma abordagem de Monte Carlo, utilizando *Fingerprint* (como discutido acima) reduz essa complexidade para $O(n+m)$ com probabilidade de erro igual a $O(1/n)$ (Karp&Rabin[1987]).

III.4.4.1 – *O problema da Multiplicação de Matrizes (Decisão):*

Suponha que A, B e C sejam 3 matrizes de ordem nxn. No seguinte problema de decisão associado à multiplicação de matrizes, deseja-se responder simplesmente se o produto de A por B é igual a C. O melhor algoritmo determinístico para esse problema, multiplica A por B utilizando o algoritmo de Coppersmith e Winograd [1987] (de complexidade $O(n^{2,37})$) e compara o resultado obtido com C.

O algoritmo randômico apresentado a seguir, é devido a Freivalds[1979] e tem um tempo de processamento da ordem de $O(log(1/\varepsilon)n^2))$ iterações, onde $\varepsilon > 0$, representa a probabilidade de falha definida a *priori* (vide algoritmo da Figura III.11). Como será visto mais adiante (Proposição III.7), a constante k presente no algoritmo, representa o número total de repetições até que se tenha probabilidade de falha inferior a ε. A notação $x \in \{0,1\}^n$ indica que cada uma das n coordenadas de x assume valores 0 ou 1 respectivamente.

Procedimento III.4: Multiplicação de matrizes (A, B, C, ε)
Início
- continua ← *true*;
- $k \leftarrow 1$;

- Repita

- escolhe $x \in \{0,1\}^n$ aleatoriamente;

-Se $ABx \neq Cx$ **então**

continua ← false;

- $k \leftarrow k+1$;

Até que ($k = \lceil \log(1/\varepsilon) \rceil$) ou (continua = *false*);

Se (continua) **então**

retorna ($AB = C$);

senão

retorna ($AB \neq C$);

fim;

fim.

Figura III.11: Verifica se *AB=C* através do método de Monte Carlo

Observe que a comparação de $ABx = Cx$ pode ser implementada em tempo $O(n^2)$ computando-se inicialmente Bx seguido de $A(Bx)$ e Cx respectivamente.

Uma vez definidos A, B e C na entrada de dados note que, se $AB=C$ então $ABx = Cx$, $\forall\ x \in \{0,1\}^n$ ou seja, $Pr(ABx=Cx)=1$. Nesta situação o algoritmo de Freivalds nunca retorna uma resposta incorreta. O que ocorre, entretanto, se $AB \neq C$? Neste caso, se para algum $x \in \{0,1\}^n$ a igualdade $ABx = Cx$ se verifica, isto não implicará necessariamente que $AB = C$! Para ilustrar essa situação considere o exemplo seguinte onde são dadas 3 matrizes A,B e C de ordem $2x2$ e o vetor $x = e$, (onde e é um vetor com todas as coordenadas iguais a 1):

$$A = \begin{pmatrix} 2 & 4 \\ 1 & 3 \end{pmatrix}; \qquad B = \begin{pmatrix} 3 & 4 \\ 1 & 1 \end{pmatrix}; \qquad C = \begin{pmatrix} 11 & 11 \\ 7 & 6 \end{pmatrix}; \quad x = e = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Calculando $A(Be)$ e Ce separadamente conclui-se diretamente que $ABe = Ce$. Entretanto, $AB \neq C$ (Verifique!).

O resultado seguinte garante que, se $AB \neq C$ então $Pr(ABx=Cx) \leq 1/2$ para x selecionado aleatoriamente. Isto é equivalente a afirmar que a probabilidade de uma resposta incorreta em uma única repetição do algoritmo é menor ou igual a 50%! A probabilidade de erro é minimizada aumentando-se o número de repetições.

Proposição III.7: Se $x \in \{0,1\}^n$ é escolhido aleatoriamente com distribuição uniforme e $AB \neq C$ então $Pr(ABx=Cx) \leq 1/2$.

Prova: Seja $D=AB-C$. Como $AB \neq C$ (hipótese) então $D \neq 0$. Assim, existirão índices $i,j \in \{1,..,n\}$ tais que $d_{ij} \neq 0$ (onde $d_{ij} \in D$). Além disso, $Pr(ABx=Cx) = Pr(Dx=0)$. Se $d_i=(d_{i1},...,d_{ij},...,d_{in})$ representa a *i-ésima* linha de D então:

$$\Pr(Dx = 0) \leq \Pr\left(\sum_{j=1}^{n} d_{ij} x_j = 0\right) \text{ , para algum } i \in \{1,..,n\} \text{ e } d_{ij} \neq 0.$$

Isolando-se x_j no segundo membro da desigualdade (já que $d_{ij} \neq 0$) conclui-se diretamente que:

$$\Pr\left(\sum_{j=1}^{n} d_{ij} x_j = 0\right) = \Pr(x_j = v), \quad onde \quad v = \frac{-1}{d_{ik}} \sum_{k \neq j} d_{ik} x_k$$

Assumindo-se que todos os demais elementos x_k $(p/\ k \neq j)$ sejam escolhidos anteriormente a x_k, tem-se que a probabilidade de x_j ser igual a um valor fixo v será menor ou igual a $1/2$. Lembre-se

que $x_j \in \{0,1\}$ é escolhido aleatoriamente satisfazendo uma distribuição uniforme. Portanto, $Pr(Dx = 0) \leq 1/2$. •

Como conseqüência deste resultado, se $AB \neq C$ então $Pr(ABx \neq Cx) \geq 1/2$. Entretanto, se $ABx \neq Cx$ para algum $x \in \{0,1\}^n$ escolhido arbitrariamente, então $(AB-C)x \neq 0$ e portanto $AB \neq C$ (pois $x \neq 0$). Logo, a cada iteração, o algoritmo retorna uma resposta correta com probabilidade de acerto superior a *50%*. A probabilidade de fracasso pode ser minimizada repetindo-se o processo um número pré-determinado de vezes. Se k repetições forem realizadas então a probabilidade de falha ε no procedimento será igual a $1/2^k$. Assim, $k \geq \lceil log(1/\varepsilon) \rceil$. Se, em algumas destas iterações ocorrer $ABx \neq Cx$ para algum $x \in \{0,1\}^n$, o processo é interrompido imediatamente retornando $AB \neq C$.

Como discutido anteriormente, o algoritmo de Freivalds terá complexidade polinomial e igual a $O(log(1/\varepsilon)\ n^2)$ para $\varepsilon > 0$, selecionado a *priori*. Note que, uma vez escolhido ε , o valor $log(1/\varepsilon)$ se torna constante. De maneira geral, pode-se afirmar que, se x é um elemento de $S \subseteq F$ (onde F é um corpo finito ou não), então $Pr(ABx=Cx) \leq 1/|S|$ (Motwani&Raghavan[1995]). É fácil ver então que a probabilidade de falha tende a 0 à medida que $|S|$ aumenta. A determinação de um algoritmo determinístico $O(n^2)$ para este problema continua um problema em aberto.

VII.4.4.2 – Verificando igualdade entre polinômios

Considere $P(x)$ e $Q(x)$ dois polinômios pertencentes a $F[x]$, onde $F[x]$ representa o conjunto de todos os polinômios em x sobre um corpo F (finito ou não). Deseja-se responder simplesmente, se $P(x) = Q(x)$. Como discutido mais adiante, a verificação deste tipo de identidade poderá ser útil em uma grande quantidade de aplicações. Obviamente, o problema é trivial se os coeficientes forem dados explicitamente. Neste caso, a igualdade será verificada diretamente em tempo $O(n)$.

Considere então o seguinte problema: dados 3 polinômios $P_1(x)$, $P_2(x)$ e $P_3(x)$ de grau no máximo n e pertencentes a $F[x]$, deseja-se responder se $P_1(x).P_2(x) = P_3(x)$? Através de transformadas de Fourieur, pode-se efetuar o produto $P_1(x).P_2(x)$ em tempo $O(nlogn)$ comparando, em seguida, o resultado com $P_3(x)$ (Motwani&Raghavan[1995]). Como o grau de $P_3(x)$ é no máximo igual a $2n$, a complexidade total deste procedimento será $O(nlogn)$.

Considere S um subconjunto qualquer de F onde $|S| \geq 2n+1$. Considere ainda o seguinte algoritmo randômico para o problema:

Algoritmo III.5: Verifica igualdade entre polinômios;

Dados: $P_1(x), P_2(x), P_3(x) \in F[x]$, onde $x \in S \subseteq F$.
Início
- $r \leftarrow$ Escolha(S);
- Calcula $P_1(r), P_2(r)$ e $P_3(r)$;
- **Se** $P_1(r).P_2(r) = P_3(r)$ **então**
  $P_1(x).P_2(x) = P_3(x)$
  senão
  $P_1(x).P_2(x) \neq P_3(x)$;

fim.

Figura III.12: Verifica se $P_1(x).P_2(x) = P_3(x)$

O cálculo de $P_i(r)$ p/ $i=1,2,3$, pode ser realizado em tempo $O(n)$ utilizando-se, por exemplo, o método de Horner (vide seção I.5). É fácil ver, portanto, que o procedimento terá complexidade total igual a $O(n)$. Observe equivalentemente que, checar se $P_1(x).P_2(x) = P_3(x)$, é o mesmo que checar se $Q(x) = P_1(x).P_2(x) - P_3(x) \equiv 0$. Note que o Algoritmo III.5 falha quando $Q(x) \neq 0$ e r é raiz de $Q(x)$. Por outro lado, se $Q(x) \equiv 0$ então $Q(r)=P_1(r).P_2(r) - P_3(r)=0, \forall r \in F$. Assim, caso se

tenha $Q(x) \bullet 0$, espera-se que $Pr(Q(r)=0)$ seja próxima de 0. Como $Q(x) \bullet 0$, possui no máximo $2n$ raízes e $|S| \geq 2n+1$, é fácil ver neste caso que: $Pr(P_1(r).P_2(r) = P_3(r)) \leq 2n/|S|$.

A Figura III.13 ilustra bem essa situação:

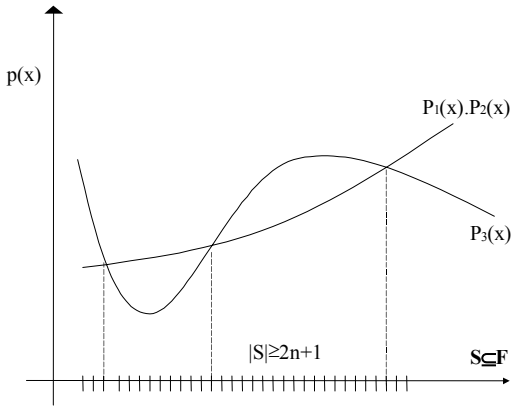


Figura III.13 Raízes de Q(x) $\bullet$ 0.

A probabilidade de falha é reduzida aumentando-se $|S|$ ou aumentando-se o número de repetições do algoritmo. Se F for um corpo infinito e r for escolhido uniformemente em F a probabilidade de erro será nula! Neste caso, entretanto, serão necessários infinitos *bits* aleatórios para geração de r!

Uma versão determinística deste algoritmo (de complexidade $O(n^2)$) pode ser obtida computando-se $2n+1$ valores distintos em S. Se para algum k entre estes valores ocorrer $Q(k) \neq 0$, pode-se concluir que $Q(x) \bullet 0$. Como discutido em Motwani&Raghavan[1995], este algoritmo pode ser implementado em tempo um pouco menor e igual a $O(n\log^2 n)$, o que é pior do que a multiplicação direta entre $P_1(x)$ e $P_2(x)$ de complexidade $O(n\log n)$ (via transformadas de Fourier).

Considere agora a seguinte extensão do problema anterior onde polinômios com múltiplas variáveis são considerados.

Seja $Q(x_1, x_2,..., x_n) \in F[x_1, x_2,..., x_n]$ um polinômio com múltiplas variáveis e grau total d[5] e S um subconjunto qualquer de F. Espera-se determinar se $Q(x_1, x_2,..., x_n) \equiv 0$, ou seja, deseja-se responder se $Q(x_1, x_2,..., x_n)=0$, $\forall x_1, x_2,..., x_n \in S \subseteq F$.

Antes de apresentar um exemplo ilustrando a utilização de polinômios com múltiplas variáveis, considere a seguinte definição:

Definição III.2: (Determinante)

Seja M de ordem nxn, uma matriz qualquer sobre F. O *determinante* de M é definido por:

$$\det(M) = \sum_{\sigma \in S_n} sinal(\sigma).\prod_{i=1}^{n} M_{i\sigma(i)}$$

onde S_n representa o conjunto de todas as permutações σ de n elementos e $sinal(\sigma)=(-1)^t$, o expoente t representa o número de inversões para se obter σ a partir da permutação identidade σ_I = (1,2,3). Note por exemplo que, se $\sigma = (2,3,1)$ então $sinal(\sigma) = 2$ pois:

$$\sigma = (2,3,1) \rightarrow (3,2,1) \rightarrow \sigma_I = (1,2,3) \qquad \bullet$$

[5] O grau de um termo qualquer em Q é a soma dos expoentes de suas variáveis. O *grau total d* será o maior grau entre todos os seus termos.

Considere agora o seguinte exemplo:

Exemplo III.1: (Matriz de Vandermonde)

Uma matriz de Vandermonde $M(x_1, x_2,..., x_n)$ com n variáveis é definida de maneira que $M_{ij}=x_i^{j-1}$, para $i,j=1,2,...,n$. Ou seja:

$$M_{ij} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix}$$

Pode-se provar neste caso que:

$$\det(M) = \prod_{j<i} (x_i - x_j)$$

Entretanto, suponha que esta igualdade não esteja provada e que se deseje verificar a validade ou não desta afirmação. Em outras palavras, espera-se responder se:

$$Q(x_1,...x_n) = \det(M) - \prod_{j<i} (x_i - x_j) \equiv 0$$

Note que, calcular diretamente o determinante da matriz simbólica $M(x_1, x_2,..., x_n)$ seria proibitivo já que $det(M(x_1, x_2,..., x_n))$ possui $n!$ parcelas (vide Definição III.2). Por outro lado, atribuindo-se valores numéricos $a_1, a_2,..., a_n$ respectivamente a $x_1, x_2,..., x_n$ tem-se que $det(M)$ pode ser avaliado em tempo $O(n^3)$ (vide Golub&Loan[1996]). •

A proposição seguinte retorna a probabilidade de falha, se $Q(x) \neq 0$:

Proposição III.8: (Schwartz-Zipel)

Seja $Q(x_1, x_2,..., x_n) \in F[x_1, x_2,..., x_n]$ uma função com múltiplas variáveis e grau máximo d. Se $S \subseteq F$ é um conjunto finito e $a_1, a_2,..., a_n \in S$ são escolhidos independentemente, de maneira aleatória e com distribuição uniforme então:

$$Pr(Q(a_1, a_2,..., a_n))=0 \mid Q(x_1, x_2,..., x_n) \neq 0) = d/|S|$$

Prova: (vide Motwani&Raghavan [1995]). •

A Proposição III.8 pode ser utilizada na construção de um novo procedimento análogo ao Algoritmo III.5 para polinômios com uma única variável. Repetições adicionais do algoritmo tornam a probabilidade de falha arbitrariamente pequena. Se ε (probabilidade de falha) é definida a *priori* então $k = \lceil log_{|S|/d}(1/\varepsilon) \rceil$ repetições serão necessárias. Verifique!

O problema da determinação das raízes de um polinômio $Q(x_1, x_2,..., x_n)$ pode ser aplicado, por exemplo, ao problema de emparelhamentos em grafos, como descrito a seguir.

III.4.4.3 – O problema do Emparelhamento Perfeito em Grafos

Esta seção ilustra a importância da utilização de polinômios com múltiplas variáveis (através de técnicas algébricas) na determinação, exibição e enumeração de Emparelhamentos Perfeitos em Grafos.

O problema da determinação das raízes do polinômio $Q(x_1, x_2,..., x_n)$ pode ser utilizado na determinação de emparelhamentos em grafos. Considere inicialmente, a seguinte situação onde G é um grafo bipartido. Um grafo $G=(U,V,E)$ é *bipartido* se, e somente se, as seguintes condições forem satisfeitas: *(a)* $U \cap V = \varnothing$, e *(b)* se $(i,j) \in E$ implicar em $i \in U, j \in V$ ou $i \in V, j \in U$. Um *emparelhamento*

$M \subseteq E$ será qualquer coleção de arestas de E sem vértices em comum. O emparelhamento M será *perfeito* se todos os vértices de $U \cup V$ forem extremidade de alguma aresta de M.

Deseja-se responder então às seguintes questões:

a) G admite algum emparelhamento perfeito M (decisão)?
b) Em caso afirmativo, encontre M ou mostre que M não existe (localização).
c) Quantos emparelhamentos perfeitos existem em G (enumeração) ?

Tratemos inicialmente da questão *(a)*. O grafo G pode ser representado por uma matriz de adjacências A de maneira que: $a_{ij}=1$, se $(i,j) \in E$ e $a_{ij}=0$, caso contrário. O exemplo seguinte exibe um grafo bipartido juntamente com sua matriz A associada:

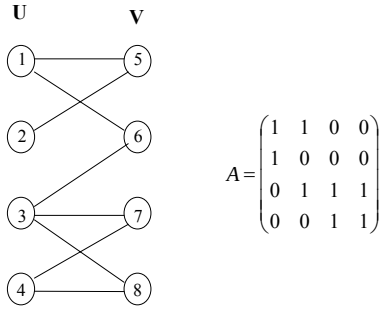


Figura III.14: Matriz Aassociada ao grafo bipartido G=(U,V,E)

Um emparelhamento perfeito em um grafo $G=(U,V,E)$ qualquer pode ser visto como uma permutação de U em V. Neste caso, cada linha e cada coluna, deverão conter um único elemento não-nulo igual a *1*. A Figura III.15, apresenta algumas permutações com 4 vértices associadas ao grafo G da Figura III.14. Note que nem todas as permutações correspondem a emparelhamentos perfeitos (apenas P_2 e P_3). Ao contrário, se G é grafo é completo e bipartido, qualquer permutação define um emparelhamento perfeito.

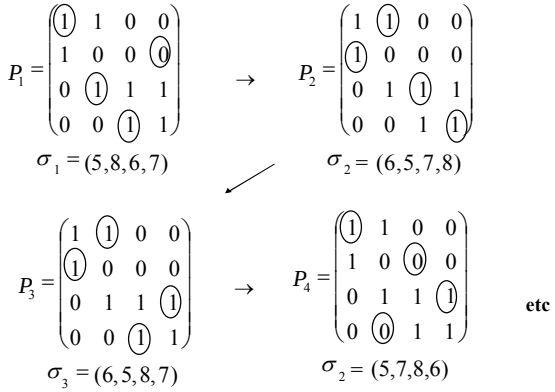


Figura III.15: Emparelhamentos Perfeitos

A proposição seguinte estabelece condições necessárias e suficientes para a existência de emparelhamentos perfeitos em grafos bipartidos.

Proposição III.9: (Edmonds[1967])

Seja A uma matriz quadrada *nxn* associada ao grafo *G=(U,V,E)* obtida da seguinte forma:

$$a_{ij} = \begin{cases} x_{ij}\,, & (u_i, v_j) \in E \\ 0\,, & (u_i, v_j) \notin E \end{cases}$$

Considere um polinômio $Q(x_{11}, x_{12},..., x_{nn})$ representando o determinante da matriz simbólica *A*. Então, *G* tem emparelhamento perfeito se, e somente se, $det(A) \neq 0$.

Prova: Motwani&Raghavan[1995] •

Para responder se $det(A) \neq 0$, deve-se determinar um conjunto *S* e um corpo *F* sendo $S \subset F$. O resultado seguinte, conseqüência imediata da Proposição III.8, responde esta questão:

Proposição III.10: Seja *G=(U,V,E)* um grafo bipartido qualquer onde $|U|=|V|=n$. Suponha que a matriz *A* seja definida como acima e que os valores $x_{11}, x_{12},..., x_{nn}$ sejam escolhidos de maneira independente e uniforme sobre o conjunto $S=\{1,2,...,2n\} \subset F$ (sendo *F* é um corpo qualquer).

Então:

$$Pr(det(A)=0 \mid det(A) \neq 0) \leq 1/2$$ •

O algoritmo seguinte utiliza as Proposições *III.9* e *III.10*, e verifica se um grafo *G* admite ou não um emparelhamento perfeito com probabilidade de erro inferior a $\varepsilon>0$.

Algoritmo III.6: Verifica se existe emparelhamento

Dados: *G=(U,V,E)* onde $|U|=|V|=n$ *e* $\varepsilon>0$.
Início
- continua $\leftarrow$ *true;*
- $k \leftarrow 1$;
- A partir de *G*, constrói matriz simbólica *A;*
- **Repita**
  - **Para** *i:=1* **até** *n* **faça**
    $a_i \leftarrow$ Escolhe*{1,2,3,..,2n};*
  - **Se** $Q(a_1,...,a_n) = det(A) \neq 0$ **então**
    continua $\leftarrow$ *false;*
  - $k \leftarrow k+1$;
- **Até que** ($k = \lceil \log(1/\varepsilon) \rceil$) ou (continua = *false*);

Se (continua) **então**
retorna (*Não existe* emparelhamento perfeito);
senão
retorna (*Existe* emparelhamento perfeito);

fim;
fim.

Figura III.16: Verifica se G contém ou não emparelhamento perfeito

Note que o Algoritmo III.6 tem complexidade $O(n^3)$ devido ao cálculo do determinante (Golub&Loan[1996]). A determinação de um emparelhamento máximo, (que é um problema de otimização), pode ser obtida em tempo determinístico $O(mn^{1/2})$ onde $m=|E|$ ou $O(n^{2,5})$ para grafos

completos (para maiores detalhes veja Micali&Vazirani[1980], Vazirani[1994] e Blum[1990])! Apesar desse resultado aparentemente insatisfatório, pode-se calcular o determinante, (etapa mais custosa do Procedimento III.6), em tempo $O(log^2 n)$ utilizando-se $O(n^{3,5})$ processadores (Chistov[1985]).

Tratemos agora da questão (b), ou seja, como determinar um emparelhamento $M \subseteq E$, caso exista? Devido a grande quantidade de assunto sobre o tema omitiremos as provas (para os leitores interessados vide Motwani&Raghavan[1995]).

Antes de tratar desta questão considere a seguinte generalização do teorema de Edmonds (apresentado na Proposição III.9). Analogamente ao teorema de Edmonds, este resultado define condições necessárias e suficientes para saber se um grafo *G* qualquer (não necessariamente bipartido) possui ou não emparelhamento perfeito:

Proposição III.11: (Tutte [1947])

Seja *A* uma matriz quadrada *nxn* anti-simétrica[6]. Considere um grafo $G=(V,E)$ qualquer e uma matriz simbólica $A=[a_{ij}]$ obtida da seguinte forma:

$$a_{ij} = \begin{cases} x_{ij}, & (u_i, v_j) \in E \ \ e \ \ i < j \\ -x_{ji}, & (u_i, v_j) \notin E \ \ e \ \ i > j \\ 0, & (u_i, v_j) \in E \end{cases}$$

Então, *G* admite emparelhamento perfeito se, e somente se, $det(A) \neq 0$. A matriz *A* como definida acima é também conhecida como *matriz de Tutte*. •

Observe agora que, com o auxílio da Proposição III.11, será possível construir um procedimento análogo ao Algoritmo III.6. Neste caso, entretanto, o resultado poderá ser aplicado a um grafo *G* qualquer, não necessariamente bipartido (vide Karp[1991]).

Considere agora os seguintes conceitos e definições da álgebra linear:

Definição III.3: (Matriz dos cofatores e matriz adjunta)

Considere *B* uma matriz quadrada *nxn.* O *cofator* Δ_{ij} associado ao elemento b_{ij} da matriz *B* é definido por $(-1)^{i+j}.det(B_{ij})$, onde B_{ij} é a submatriz de *B* obtida após a remoção da *i-ésima* linha e *j-ésima* coluna. Com estes cofatores, pode-se construir uma outra matriz $cof(B)=[\Delta_{ij}]$, denominada *matriz de cofatores.* A *matriz adjunta* (representada por *adj(B)*) de uma matriz quadrada *B* será a transposta da matriz dos cofatores *cof(B).* •

Para ilustrar os conceitos apresentados acima considere o seguinte exemplo apresentado em Boldrini et. al.[1984]:

Exemplo III.2: (Matriz dos cofatores e matriz adjunta)

Considere a seguinte matriz *B* de ordem 3 e os respectivos cofatores:

$$B = \begin{pmatrix} 2 & 1 & 0 \\ -3 & 1 & 4 \\ 1 & 6 & 5 \end{pmatrix} \quad e \quad \Delta_{11} = (-1)^{1+1}\begin{vmatrix} 1 & 4 \\ 6 & 5 \end{vmatrix} = -19, \quad \Delta_{12} = (-1)^{1+2}\begin{vmatrix} -3 & 4 \\ 1 & 5 \end{vmatrix} = 19, \ etc.$$

Logo:

[6] Lembre-se que uma matriz $A=[a_{ij}]$ de ordem *nxn* é *anti-simétrica* se, se e somente se, $a_{ij}=-a_{ji}$, $\forall i,j \in \{1,..,n\}$. Observe ainda que $a_{ii}=0$, $p/\ i=1,..,n$.

$$cof(B)=\begin{pmatrix}\Delta_{11} & \Delta_{12} & \Delta_{13}\\ \Delta_{21} & \Delta_{22} & \Delta_{23}\\ \Delta_{31} & \Delta_{23} & \Delta_{33}\end{pmatrix}=\begin{pmatrix}-19 & 19 & -19\\ -5 & 10 & -11\\ 4 & -8 & 5\end{pmatrix} \quad e \quad adj(B)=\begin{pmatrix}\Delta_{11} & \Delta_{21} & \Delta_{31}\\ \Delta_{12} & \Delta_{22} & \Delta_{32}\\ \Delta_{13} & \Delta_{32} & \Delta_{33}\end{pmatrix}=\begin{pmatrix}-19 & -5 & 4\\ 19 & 10 & -8\\ -19 & -11 & 5\end{pmatrix}.$$

•

É importante lembrar ainda da Álgebra Linear que $adj(B)=det(B).B^{-1}$ (vide Boldrini et al.[1984]). Esta propriedade será utilizada mais adiante.

Novamente, seja $G=(V,E)$ um grafo qualquer. A idéia central na determinação de um emparelhamento perfeito $M \subseteq E$ é atribuir, convenientemente, pesos às arestas de G de maneira que, se um emparelhamento perfeito M^* de custo mínimo for único, será possível exibir as arestas de um emparelhamento perfeito M qualquer, computando-se determinantes em paralelo. Antes de apresentar o procedimento para determinação de M, considere ainda os seguintes resultados auxiliares:

Proposição III.12: (*Isolation Lemma*)

Seja (X,Φ) um par representando um conjunto qualquer $X=\{x_1,...,x_m\}$ e uma família $\Phi=\{S_1,S_2,...,S_k\}$ de subconjuntos de X. Seja $w:X\rightarrow\{1,...,2m\}$ uma função inteira positiva obtida após a atribuição de valores $1,2,...,2m$ (escolhidos de maneira uniforme e independente) aos elementos de X. Além disso, considere $w(S)=\sum_{xj \in S} w(x_j)$ o custo associado ao subconjunto $S \subseteq X$. Então, com probabilidade maior ou igual a $1/2$, existirá um único subconjunto S de Φ de custo mínimo. •

Observe que a Proposição III.12 pode estar associada ao problema do Emparelhamento desde que X, represente o conjunto de arestas de G, e Φ, o conjunto de todos os emparelhamentos perfeitos em G. Assim, atribuindo-se pesos inteiros variando de 1 a $2m$ às arestas de G, garante-se a existência de um único emparelhamento perfeito de custo mínimo com probabilidade de sucesso maior ou igual a 50%. Como observado mais adiante, este resultado será fundamental na determinação de um emparelhamento perfeito M em G.

Considere agora a matriz de Tutte A associada a um grafo G qualquer. Além disso, considere uma matriz B (associada a A) onde $x_{ij}=2^{w(i,j)}$ e os elementos $w(i,j)$ pertencentes ao conjunto $\{1,2,...,2m\}$ são escolhidos aleatoriamente de maneira uniforme e independente. Tem-se então a seguinte proposição:

Proposição III.13: Suponha que $G=(V,E)$ contenha um único emparelhamento perfeito M^* de custo mínimo W. Então, pode-se provar que $det(B) \neq 0$, e a maior potência de 2 que divide $det(B)$ é igual a 2^{2W}. •

Observe que esta proposição auxilia na determinação de W desde que se conheça $det(B)$ (verifique). Finalmente, a proposição seguinte é a base do procedimento para determinação de um emparelhamento perfeito. Lembre-se que B_{ij} é obtida de B após a eliminação da linha i e coluna j respectivamente.

Proposição III.14: Seja M^* o único emparelhamento perfeito de custo mínimo W em G. A aresta (i,j) de E pertence a M^* se, e somente se o valor,

$$\frac{\det(B_{ij}).2^{w(i,j)}}{2^{2W}}$$

for ímpar. •

Com base nestes resultados, tem-se o seguinte algoritmo (executado em paralelo) para determinação de um emparelhamento perfeito $M \subseteq E$:

Algoritmo III.7: (Determinação de um Emparelhamento Perfeito)
Início

1. Leia $(G=(V,E))$;
2. **Para** todas as arestas (i,j) de E **faça** (em pararelo)
   compute: $w(i,j) \in \{1,..,2m\}$ aleatóriamente e uniformemente;
3. Calcule a matriz de Tutte B fazendo $x_{ij}=2^{w(i,j)}$, p/ $i,j=1,...,n$;
4. Calcule $det(B)$;
5. Calcule W de maneira que 2^{2W} represente a maior potência de 2 dividindo $det(B)$;
6. Calcule $adj(B)=det(B)xB^{-1}$
7. **Para** todas as arestas (i,j) de E **faça** (em paralelo)
   compute: $r_{ij}=det(B_{ij}).2^{w(i,u)}/2^{2W}$;
8. **Para** todas as arestas (i,j) de E **faça** (em paralelo)
   Se r_{ij} é ímpar **então** adiciona (i,j) a M;
9. **Se** M é emparelhamento perfeito **então** retorna (M)
   senão volta para passo 2.

Fim.

Figura III.17: Emparelhamento perfeito em paralelo – Las Vegas

Note no procedimento acima (etapas 6 e 7) que a posição (j,i) da matriz $adj(B)$ contém o elemento $\Delta_{ij}=(-1)^{i+j}det(B_{ij})$, logo os determinantes $det(B_{ij})$, para $i,j=1,...,n$ são determinados diretamente.

A etapa mais cara computacionalmente neste procedimento é o cálculo de B^{-1}, $det(B)$ e $det(B_{ij})$ respectivamente. Entretanto, como observado em Motwani&Raghavan[1995] este trabalho poderá ser implementado em tempo $O(log^2 n)$ utilizando-se $O(n^{3,5}m)$ processadores. Note ainda, que a probabilidade de falha pode ser reduzida a *zero* repetindo-se o processo até que um emparelhamento perfeito tenha sido obtido (método de Las Vegas).

Finalmente, na questão (c), quantos emparelhamentos perfeitos existem em um grafo bipartido G? Trata-se, na verdade, de um problema de contagem ou enumeração.

De maneira geral, se I representa uma instância de um problema de decisão $\pi \in NP$ (vide Capítulo I), pode-se associar um valor $\#(I) \in Z^+$ representando o número de provas (ou soluções) justificadas polinomialmente à resposta "*sim*" de π. Representa-se por $\#P$, a classe dos problemas de enumeração associados a problemas de decisão em NP[7]. Ainda, um problema de decisão π será *#P-Completo* se, e somente se, um problema π' qualquer em $\#P$ puder ser reduzido a π em tempo polinomial.

Como discutido no Capítulo I, um problema de otimização será tão difícil quanto seu problema de decisão associado e vice-versa. O mesmo não ocorre entre problemas de decisão e enumeração. Claramente, um problema de enumeração será tão difícil quanto sua versão de decisão associada. A recíproca, entretanto, não é verdadeira. Como observado no item (a), a determinação da existência ou não de emparelhamentos perfeitos em grafos bipartidos é polinomial. Entretanto, a menos que $P=NP$, o problema de enumeração correspondente não pode ser resolvido polinomialmente.

Seja grafo $G=(V \cup U,E)$ (onde $|V|=|U|=n$), um grafo bipartido representado por uma matriz de adjacências A de maneira que: $a_{ij}=1$, se $(i,j) \in E$ e $a_{ij}=0$, caso contrário. Como observado a seguir, o número de emparelhamentos perfeitos em G pode ser representado pelo permanente da matriz A, definida logo a seguir:

[7] A classe #P foi introduzida inicialmente por Valiant[1979],[1982].

Definição III.4: (Permanente)

Seja *A* de ordem *nxn*, uma matriz qualquer sobre *F*. O *permanente* de *A* é definido por:

$$perm(A) = \sum_{\sigma \in S_n} \left(\prod_{i=1}^{n} A_{i\sigma(i)} \right)$$

onde S_n representa o conjunto de todas as permutações σ de *n* elementos. •

Pode-se mostrar facilmente que, se *#(G)* representa o número de emparelhamentos perfeitos em *G* então *perm(A(G))=#(G).* Para mostrar esse resultado basta notar que, em um emparelhamento perfeito, o número de 1's em cada linha e cada coluna, definido pela permutação σ é igual a *um.* Isto significa que, na expressão do permanente, a parcela correspondente à permutação σ será igual a *um* (verifique). Observe, por inspeção, que o exemplo das Figuras III.14 e III.15, contém exatamente 2 permutações correspondendo aos 2 únicos emparelhamentos perfeitos de *G*.

Como discutido anteriormente, embora a definição formal de determinante contenha um número fatorial de termos, pode-se calcular o valor do determinante em tempo $O(n^3)$ utilizando técnicas do tipo decomposição *LU* (Golub&Loan[1996]). Surpreendentemente, apesar da grande semelhança existente nas definições do permanente e determinante, não se conhece nenhum procedimento polinomial para o cálculo do permanente. Na verdade, o melhor algoritmo para este problema foi obtido por Ryser[1963] e tem complexidade exponencial igual a $O(n2^n)$!

Devido a grande semelhança existente nas definições do permanente e determinante, alguns autores sugerem que o cálculo do determinante, possa talvez ser utilizado convenientemente na determinação do permanente. Godsil e Guttman propõem a seguinte relação: se *A* é uma matriz com coeficientes *0-1,* e *B* é uma matriz obtida atribuindo-se aos coeficientes não-nulos de *A* sinal "+" ou "-", de maneira aleatória e independente. Então $E(det^2(B))=perm(A)$.

Como observado em Karp[1991], isto sugere a criação de um procedimento de Monte Carlo baseado na geração de *n* amostras independentes (matrizes *B*). Apesar de interessante, esta abordagem não funciona bem em certas situações. Por exemplo, se *A* possui *n/2* blocos de tamanho *2x2* na diagonal, formados apenas por coeficientes *unitários* (sendo os demais elementos iguais a *zero)*, então, cada um dos *n/2* blocos terão determinante não-nulo com probabilidade 1/2. Note que uma submatriz B_k de 1's tem determinante nulo sempre que o número de sinais positivos (ou negativos) for par e igual a 0 ou 2 (verifique). Da álgebra linear tem-se que o determinante de *B* é não-nulo apenas se os *n/2* blocos da diagonal tiverem determinante não-nulo. É fácil ver então que $Pr(det^2(B)\neq 0)=2^{-n/2}$. Isto implica que aproximadamente $2^{n/2}$ repetições deverão ser realizadas até que uma das amostras B (geradas aleatoriamente) possua determinante não-nulo.

Uma abordagem mais interessante é a utilização algoritmos randômicos polinomiais que geram soluções aproximadas. Mais detalhes sobre este assunto podem ser obtidos em Motwani&Raghavan[1995].

III.8 - CLASSES DE COMPLEXIDADE

O enunciado de um problema deve conter uma descrição formalizada de todos os seus parâmetros e objetos. Contudo, para que um programa de computador resolva um problema é necessário que qualquer entrada I associada seja representada convenientemente pela máquina. Usualmente são utilizados símbolos ou dígitos binários 0,1. De maneira geral, uma *codificação* mapeia um conjunto de objetos abstrados I para uma coleção de elementos ou palavras formadas por um conjunto finito de símbolos Σ também chamado *alfabeto*. Por exemplo, *a1$$aal* ou *aa1a1* são palavras formadas por elementos de $\Sigma=\{1,a,\$\}$.

Representa-se por Σ^*, o conjunto de todas as palavras, incluindo a palavra vazia *e*, construídas a partir dos elementos de Σ. Logo, uma *linguagem* $L \subseteq \Sigma^*$ é qualquer conjunto de palavras formadas por símbolos de um alfabeto Σ. Em um problema de *reconhecimento da linguagem L* deve-se responder se uma palavra qualquer *x* de Σ^* pertence ou não a *L*.

Usualmente, deseja-se representar linguagens cujos elementos (ou palavras) satisfaçam certas propriedades definidas a *priori*. Mais formalmente:

$$L=\{x \in \Sigma^*: x \text{ satisfaz a propriedade } \Pi\}$$

Os problemas de decisão por exemplo (discutidos no Capítulo I), podem ser associados a linguagens desde que, em sua representação, esteja também associado um procedimento sistemático para decidir se *x* satisfaz ou não a propriedade Π, ou seja, deseja-se responder simplesmente se $x \in L$. Esse procedimento sistemático, ou algoritmo, deverá ser formado por uma seqüência finita de instruções e terminar cedo ou tarde sempre que $x \in L$. Contudo, se $x \notin L$ o algoritmo poderá não terminar e entrar em *loop* indefinidamente. Caso isto ocorra, a linguagem *L* estará representando um problema *não-decidível*. Caso contrário, se o algoritmo sempre termina $\forall x \in \Sigma^*$, a linguagem *L* representa um problema *decidível*.

Um algoritmo *A aceita* $x \in \{0,1\}^*$ se $A(x)=1$. Em outras palavras, pode-se dizer que o elemento $x \in \{0,1\}^*$, representando uma instância do problema de decisão, satisfaz a propriedade Π. Logo, a linguagem *L* é *aceita* por *A* se $L=\{x \in \{0,1\}^*: A(x)=1\}$. O algoritmo A *rejeita* uma palavra $x \in \{0,1\}^*$ se $A(x) = 0$. Caso isso ocorra, pode-se afirmar então que $x \notin L$ ou que *x* não satisfaz a propriedade Π. Se o algoritmo *A* aceita ou rejeita $x \in \{0,1\}^*$, então *A decide x* em $\{0,1\}^*$.

Definição III.5: Uma linguagem *L* é *aceita* por um algoritmo *A* em tempo polinomial se, $\forall x \in L$, *A* aceita *x* em tempo $O(|x|^k)$, para *k* constante. •

Em teoria de complexidade, para se classificar problemas de otimização quanto a seu grau de dificuldade, observa-se, apenas, sua versão de decisão associada. Qualquer problema de decisão pode ser tratado, equivalentemente, como um problema de reconhecimento de linguagem. Uma classe de complexidade pode então ser vista como uma coleção de linguagens cujo problema de reconhecimento seja resolvido dentro de limites de tempo ou espaço pré-estabelecidos. Se *A* decide *L* em tempo $O(2^{p(x)})$, $\forall x \in \{0,1\}^*$ tem-se a classe *EXP* (exponencial) dos problemas de decisão ($p(.)$ representa um polinômio em $|x|$). Mais formalmente:

$$EXP = \{L \subseteq \{0,1\}^*: \exists A \text{ que decide } L \text{ em tempo exponencial}\}$$

Definição III.6: Um algoritmo *A decide* uma linguagem *L* em tempo polinomial se, e somente se, $\forall x \in \{0,1\}^*$, *A* decide *x* em tempo $O(|x|^k)$, sendo *k* constante. •

A classe *P* dos problemas de decisão, pode ser definida como:

$$P = \{L \subseteq \{0,1\}^*: \exists A \text{ que decide } L \text{ em tempo polinomial}\}$$

Em outras palavras, uma linguagem *L* pertence a *P* se existir um algoritmo *A* polinomial tal que $A(x)=1$ ou $A(x)=0$, $\forall x \in \{0,1\}^*$. Note, a partir das definições acima que $P \subseteq EXP$.

Um algoritmo A *verifica* $x \in \{0,1\}^*$ se existir $y \in \{0,1\}^*$ tal que $A(x,y)=1$. A palavra *y* é chamada *certificado* de *x*. O algoritmo A *verifica* uma linguagem *L* contida em $\{0,1\}^*$ se: $L = \{x \in \{0,1\}^*: \exists\, y \in \{0,1\}^*$ tal que $A(x,y)=1\}$. Se esta verificação ocorre em tempo polinomial então:

$$L = \{x \in \{0,1\}^*: \exists y \text{ onde } |y| = O(|x|^k) \text{ tal que } A(x,y)=1 \text{ e } k \in N\}$$

A classe *NP* é o conjunto de todas as linguagens verificadas por *A* em tempo polinomial, ou seja:

$$NP = \{L \subseteq \{0,1\}^*: \exists A \text{ que verifica } L \text{ em tempo polinomial}\}$$

Observe que se $x \notin L$ então $\forall y \in \{0,1\}^*$ tem-se $A(x,y) = 0$. Associando a linguagem *L* a um problema de decisão π em *NP*, pode-se afirmar equivalentemente que, se $\pi \in NP$ então o reconhecimento ou certificado à resposta *sim* de π será realizado em tempo polinomial (vide Capítulo I). Em outras palavras, a classe *P* consiste de todos os problemas de decisão resolvidos em tempo polinomial, e *NP,* consiste de todos os problemas de decisão verificados polinomialmente.

Ao associar um problema de decisão π a uma linguagem *L*, a propriedade Π pode ser encarada como o enunciado deste problema, e $x \in \{0,1\}^*$, uma instância deste problema. O problema do Caixeiro Viajante (*CV)* por exemplo, na sua versão decisão, pode ser representado da seguinte forma:

$$CV = \{x = \langle G,k\rangle : x \text{ admite } \textit{ciclo hamiltoniano} \text{ de tamanho} \leq k\}$$

A palavra $x \in \{0,1\}^*$, representa a codificação em binário de um grafo valorado qualquer *G* juntamente com $k \in Z^+$.

De maneira geral, para qualquer classe de complexidade Ψ, pode-se definir a classe *co-*Ψ como sendo *co-*$\Psi = \{L: L^c \in \Psi\}$ (onde L^c é complemento de *L*). Note por exemplo que: *co-P, co-NP* e *co-EXP* são exemplos de classes complementares de *P, NP* e *EXP* respectivamente. Pode-se mostrar facilmente que $P = co\text{-}P$ e $P \subseteq NP \cap co\text{-}NP$. Entretanto, não se sabe se $P = NP \cap co\text{-}NP$ ou $NP = co\text{-}NP$.

Se a linguagem *L* estiver associada a um algoritmo randômico, outras classes de complexidade podem ser definidas:

Definição III.7: (Classe *RP - Randomized Polinomial time*)

A classe *RP* consiste de todas as linguagens *L* com um algoritmo randômico *A* com pior caso polinomial e tal que $\forall x \in \{0,1\}^*$, têm-se:

$$\text{i) } x \in L \Rightarrow Pr(A(x) = 1) \geq 1/2$$
$$\text{ii) } x \notin L \Rightarrow Pr(A(x) = 0) = 1 \quad \bullet$$

O quociente *1/2* foi definido arbitrariamente. Observe que, se $x \notin L$, a probabilidade do algoritmo finalizar com uma resposta incorreta, ou seja, $A(x) = 1$ é nula. Contudo, se $x \in L$ o algoritmo retorna uma resposta correta (neste caso, $A(x) = 1$) com probabilidade de acerto maior ou igual a *1/2*.

Considere, por exemplo, o problema da multiplicação de matrizes (decisão) estudado na Seção III.4.4.1 representado por $MULT=\{x = \langle A,B,C\rangle : AB \neq C\}$ onde $x = \langle A,B,C\rangle \in \{0,1\}^*$ representa a codificação (em binário) das matrizes *A, B* e *C* respectivamente. Neste caso, a propriedade Π é válida apenas se $AB \neq C$. Assim, após a primeira iteração do procedimento III.4 tem-se que: se $AB \neq C$ (ou seja, se $x \in L$) então $Pr(ABz \neq Cz) \geq 1/2$ para algum $z \in \{0,1\}^n$ (ou seja, $Pr(A(x)=1) \geq 1/2$). Analogamente, se $AB=C$ (ou seja, se $x \notin L$) então $Pr(ABz = Cz) = 1, \forall z \in \{0,1\}^n$ (ou seja, $Pr(A(x)=0)=1$). Logo, da definição III.7 conclui-se que $MULT \in RP$.

Considere agora a seguinte definição de co-RP:

Definição III.8: (Classe co-*RP* - *Randomized Polinomial time*)

A classe *RP* consiste de todas as linguagens *L* com um algoritmo randômico *A* com pior caso polinomial e tal que $\forall x \in \{0,1\}^*$, têm-se:

$$\text{i) } x \in L \Rightarrow Pr(A(x) = 1) = 1$$
$$\text{ii) } x \notin L \Rightarrow Pr(A(x) = 0) \geq 1/2.$$ •

Analogamente, seja $MULT^c$ o complemento do problema *MULT* (discutido acima), pode-se mostrar então que $MULT^c = \{x = \langle A,B,C\rangle : AB = C\} \in co\text{-}RP$ (verifique). Problemas de decisão em RP ou co-RP são conhecidos, equivalentemente, como problemas com *erro unilateral* (*one-sided error*).

Como discutido anteriormente, se *A* é um algoritmo randômico para um problema de decisão pertencente a RP ou co-RP, repetições adicionais de *A* reduzem a probabilidade de falha exponencialmente. Se $\varepsilon > 0$ é a probabilidade de erro (definida na entrada de dados), e $1/2^k$ representa a probabilidade de falha de *A* após *k* passos, espera-se então que $1/2^k < \varepsilon$. Assim, se $k = \lceil \log(1/\varepsilon) \rceil$ repetições forem realizadas a probabilidade de erro será inferior a ε.

Definição III.9: (Classe *ZPP* - *Zero-error Probabilistic Polynomial time*)

A classe *ZPP* representa a classe de todas as linguagens cujo algoritmo de Las Vegas associado tem tempo esperado de processamento polinomial. •

Equivalentemente, um problema de decisão π pertence a *ZPP* se, e somente se, existir um algoritmo randômico *A* com pior caso polinomial tal que $Pr(A(x) = 1) = 1$ se $x \in L$ e $Pr(A(x) = 0) = 1$, se $x \notin L$, $\forall x \in \{0,1\}^*$. Os problemas pertencentes a essa classe tem *erro-nulo* (*zero-sided error*).

Os problemas da Coloração de Conjuntos, Ordenação e Geração de Orientações Acíclicas em Grafos (vistos na seção III.3), embora não sejam problemas de decisão, são exemplos de problemas pertencentes a *ZPP* já que possuem tempo esperado polinomial. A proposição seguinte estabelece a relação existente entre as classes *ZPP*,*RP* e *co-RP:*

Proposição III.15: $ZPP = RP \cap co\text{-}RP$.

Prova: É fácil ver que, se uma linguagem *L* qualquer pertence à classe *ZPP* então *L* pertence a *RP* e *co-RP* simultaneamente. Logo, $ZPP \subseteq RP \cap co\text{-}RP$.

Considere agora uma linguagem L qualquer pertencente a $RP \cap co\text{-}RP$. Sejam A_1 e A_2 dois algoritmos com pior caso polinomial tais que: se $x \in L$ então $Pr(A_1(x) = 1) \geq 1/2$ e $Pr(A_2(x) = 1) = 1$. Além disso, se $x \notin L$ então $Pr(A_1(x) = 0) = 1$ *e* $Pr(A_2(x) = 0) \geq 1/2$. Note que isto é sempre possível já que *L* pertence a *RP* e *co-RP* simultaneamente. Um novo algoritmo *A* pode ser obtido da seguinte forma:

Algoritmo A: (Mostra que $L \in ZPP$)
Início
  Leia $x \in \{0,1\}^n$;
  Repita
    Se $A_1(x) = 1$ **então** retorna "*sim*"
    senão
      Se $A_2(x) = 0$ **então** retorna "*não*"
  Até retornar "*sim*" ou "*não*";
fim.

A idéia da prova consiste em mostrar simplesmente que o algoritmo *A* acima decide *L* em tempo esperado polinomial. Note inicialmente que, se $A_1(x)=1$ então $x \in L$. Caso contrário, se $x \notin L$ tem-se $A_1(x)=0$, pois $Pr(A_1(x)=0)=1$ sempre que *x* não pertencer a *L* (lembre-se que $L \in RP$). Por outro lado, se $A_1(x)=0$ a *string x* pode ou não pertencer a *L*.

Novamente, se $A_2(x)=0$ então $x \notin L$. Caso contrário, se $x \in L$ tem-se $A_2(x)=1$, pois $Pr(A_2(x)=1)=1$ sempre que *x* pertencer a *L* (lembre-se que $L \in$ a *co-RP*). Se $A_2(x)=1$ então a *string x* pode ou não pertencer a *L*.

Observe que o processo é repetido até que *A* retorne "*sim*" ou "*não*". Assim, se $x \in L$ é fácil ver que $Pr(A(x)=1)=Pr(A_1(x)=1) \geq 1/2$ e $Pr(A(x)=0)=(1- Pr(A_1(x)=1)).Pr(A_2(x)=0)=0$ (como $x \in L$ então $Pr(A_2(x)=0)=0$). Resumindo, se $x \in L$ então $Pr(A(x)=1) \geq 1/2$.

Se $x \notin L$ então $Pr(A(x)=0)=Pr(A_2(x)=0) \geq 1/2$. Note neste caso que, como $x \notin L$ então $Pr(A_1(x)=0)=1$.

Finalmente, em uma dada iteração do algoritmo *A* tem-se "*sim*" ou "*não*" com probabilidade maior ou igual a 1/2 (probabilidade de sucesso). O número de repetições pode ser representado convenientemente por uma variável aleatória com distribuição geométrica e probabilidade de sucesso 1/2, e, consequentemente, com valor esperado 2. Como A_1 e A_2 são polinomiais segue que o algoritmo randômico *A* tem tempo esperado polinomial. Mostrou-se portanto que $ZPP \subseteq RP \cap co\text{-}RP$ e $RP \cap co\text{-}RP \subseteq ZPP$, ou seja, $ZPP = RP \cap co\text{-}RP$. •

Considere agora a seguinte definição:

Definição III.10: (Classe *BPP - Bounded-error Probabilistic Polynomial time*)

A classe *BPP* consiste de todas as linguagens *L* com um algoritmo randômico *A* cujo pior caso seja polinomial . Além disso, $\forall x \in \{0,1\}^*$, obtêm-se:

$$\text{i) } x \in L \Rightarrow Pr(A(x) = 1) \geq 3/4$$
$$\text{ii) } x \notin L \Rightarrow Pr(A(x) = 0) \geq 3/4$$ •

Para exemplificar esta definição, considere novamente o problema do Corte Mínimo em Grafos (estudado na seção III.4.2).

Exemplo III.3: (Corte Mínimo – BPP)

Seja *G* um grafo qualquer com peso positivo nas arestas, e *K*, um inteiro positivo. Deseja-se responder se o Corte Mínimo em *G* tem valor CM exatamente igual a *K*. Sempre que isto ocorrer diz-se que $x=<G, K> \in L$, ou seja, que *x* satisfaz a propriedade Π (neste caso, representada por $CM=K$). Seja *k*, uma solução do problema do Corte Mínimo gerada após sucessivas repetições do método de Monte Carlo com probabilidade de sucesso maior ou igual a 3/4 (vide seção III.4.2). Considere agora o seguinte algoritmo:

Algoritmo III.8.: Corte-Mínimo-BPP
Dados: Grafo *G* c/ peso positivo nas arestas e $K \in Z^+$
Início
- $k \leftarrow$ Valor do Corte Mínimo(Monte Carlo) c/ probab. de sucesso $\geq 3/4$;
- **Se** $K=k$ **então** retorna ($CM=K$)
  senão retorna ($CM \neq K$).

fim.

Figura III.18: Corte Mínimo-BPP

Observe neste caso que, se $x \in L$, ou seja, se o corte mínimo *CM* é de fato igual a *K* então o Algoritmo III.8 aceita *x* com probabilidade maior ou igual a 3/4 ($Pr(K=k) \geq 3/4$). Por outro lado, se $x \notin L$, ou seja, se $CM < K$ então o Algoritmo III.8 rejeita x com probabilidade maior ou igual a 3/4. Assim, $Pr(K \neq k) \geq 3/4$. Se CM>K, o algoritmo III.8 sempre rejeita *x*, já que a solução *k* nunca será inferior a *K*. Logo, da Definição III.10 conclui-se que o problema do Corte-Mínimo em Grafos pertence a *BPP*. •

Considere agora a seguinte questão: dados $\varepsilon > 0$, e um algoritmo randômico polinomial *A* para uma linguagem *L* pertencente a *BPP*, quantas repetições de *A* deverão ser executadas até que se tenha uma probabilidade de falha inferior a ε? Da definição de *BPP*, tem-se que, se $x \in L$ então $Pr(A(x)=1) \geq 3/4$. Caso contrário, se $x \notin L$ então $Pr(A(x)=0) \geq 3/4$. As *duas* situações representam a probabilidade de sucesso (sempre maior ou igual a 3/4). Observe agora que repetições adicionais de *A* incrementam a probabilidade de sucesso. Assim, espera-se que *k* repetições sejam realizadas de maneira que $Pr(A(x)=1) \geq 1-\varepsilon$ (se $x \in L$) ou $Pr(A(x)=0) \geq 1-\varepsilon$ (se $x \notin L$). Considere então o Algoritmo III.9 descrito logo a seguir. As variáveis *N0* e *N1* contam, respectivamente, o número de 0's (rejeições) e 1's (aceitações) de $x \in \{0,1\}^*$. A atribuição dada a *k* (número de repetições) será discutida mais adiante.

Algoritmo III.9: Reconhece *L* em *BPP*.

Dados: $x \in \{0,1\}^*$, $L \in BPP$ e $\varepsilon > 0$.
Início
- Leia(*x*);
- $N0 \leftarrow 0$; $N1 \leftarrow 0$;
- **Repita**
  Se $A(x)=0$ **então** $N0 \leftarrow N0+1$;
  senão $N1 \leftarrow N1+1$;
- **Até que** $k \geq \lceil 8|\ln 2\varepsilon| \rceil$,
Se $N0 \geq N1$ **então** retorna ('0')
senão retorna ('1');

Fim.

Figura III.19: Incremento da probabilidade de sucesso.

Observe que, se $x \notin L$, o Algoritmo III.9 retorna '0' apenas quando o número de sucessos (rejeições) for maior ou igual a 50% (ou seja, se $N0 \geq N1$). Por outro lado, se $x \in L$ o algoritmo retorna '1' sempre que $N1 > N0$. Para evitar conflito na escolha de "0" ou "1" (que ocorre quando $N1=N0$) basta executar o procedimento um número ímpar de vezes.

O exemplo seguinte ilustra que, quanto maior o número de repetições do algoritmo, maior a probabilidade de sucesso.

Exemplo III.4: (Número de repetições no Algoritmo III.9)

Considere L uma linguagem qualquer pertencente a BPP. Suponha que *x* pertença a *L* e que 3 repetições do Algoritmo III.9 sejam realizadas. Neste caso, o algoritmo retorna "1", apenas se, em 3 repetições, o número de aceitações ($A(x)=1$) for maior que o de rejeições ($A(x)=0$). O número de sucessos (aceitações) pode ser representado convenientemente por uma variável aleatória com distribuição binomial (vide Capítulo II). Assim:

$$\Pr(\text{Alg. } III.9 \text{ retorna "1"}) = \binom{3}{2}\cdot\left(\frac{3}{4}\right)^2\left(\frac{1}{4}\right) + \binom{3}{3}\cdot\left(\frac{3}{4}\right)^3 = \frac{27}{64} + \frac{27}{64} = \frac{54}{64} \cong 0{,}8437$$

Por outro lado, se 5 repetições forem realizadas então:

$$\Pr(A\text{lg}.III.9\ retorna\ "1") = \binom{5}{3}\left(\frac{3}{4}\right)^3\left(\frac{1}{4}\right)^2 + \binom{5}{4}\left(\frac{3}{4}\right)^4\left(\frac{1}{4}\right) + \binom{5}{5}\left(\frac{3}{4}\right)^5 = \frac{270}{1024} + \frac{405}{1024} + \frac{243}{1024} = \frac{963}{1024} \cong 0{,}9404$$

Observe que, à medida que o número *k* de repetições aumenta, então *Pr(Alg. III.9 retorna "1")* se aproxima de *1* sempre que *x* pertencer a *L*. O mesmo raciocínio pode ser aplicado quando $x \notin L$. •

Para determinar o valor *k* tal que a probabilidade de sucesso seja superior a $1-\varepsilon$, considere, sem perda de generalidade, que *x* pertença a *L*. Neste caso, obtêm-se sucesso sempre que *A(x)=1*. Seja *r* o número de eventos *A(x)=1* (sucessos) obtidos em *k* repetições do algoritmo. O total de sucessos pode ser modelado convenientemente por uma variável aleatória com distribuição binomial como descrito a seguir. Se $r \le k/2$ então:

$$\Pr\big(A(x)=1,\ r\ vezes\ em\ k\ \ tentativas\big) = \binom{k}{r}(3/4)^r(1/4)^{k-r} \le \binom{k}{r}(3/4)^r(1/4)^{k-r}\frac{(3/4)^{k/2-r}}{(1/4)^{k/2-r}} = \binom{k}{r}(3/16)^{k/2}$$

Seja *Y* um evento que ocorre sempre que *A* aceitar *x* mais do que $\lfloor k/2 \rfloor$ vezes . Assim:

$$\Pr(Y) = 1 - \sum_{r=0}^{\lfloor k/2 \rfloor} \Pr(A(x)=1,\ r\ vezes\ em\ k\ tentativas) \ge 1 - \left(\frac{3}{16}\right)^{k/2} \sum_{r=0}^{\lfloor k/2 \rfloor}\binom{k}{r}$$

Do binômio de Newton tem-se que, para *a* e *b* inteiros positivos:

$$(a+b)^k = \binom{k}{0}.a^k.b^0 + \binom{k}{1}.a^{k-1}b + \ldots + \binom{k}{k}.a^0b^k$$

Fazendo-se *a=b=1,* e dividindo ambos os lados da igualdade por 2 conclui-se que:

$$2^{k-1} = \sum_{r=0}^{k}\binom{k}{r}\Big/2 = \sum_{r=0}^{\lfloor k/2 \rfloor}\binom{k}{r}$$

Logo, substituindo esta última igualdade em *Pr(Y)*:

$$\Pr(Y) = 1 - \sum_{r=0}^{\lfloor k/2 \rfloor}\Pr(A(x)=1,\ r\ vezes\ em\ k\ tentativas) \ge 1 - (3/16)^{k/2}2^{k-1}$$

A constante *k* deve ser escolhida de maneira que: $1-(3/16)^{k/2}2^{k-1} \ge 1-\varepsilon$. Após as devidas simplificações conclui-se que:

$$(3/16)^{k/2}2^{k-1} \le \varepsilon \quad \Rightarrow \quad (3/4.2^2)^{k/2}2^{k-1} \le \varepsilon \quad \Rightarrow \quad (3/4)^{k/2}2^{-1} \le \varepsilon \quad \Rightarrow \quad (3/4)^{k/2} \le 2\varepsilon$$

Ou ainda:

$$k \ge \left|\frac{2.\ln(2\varepsilon)}{\ln(3/4)}\right| \quad \Rightarrow \quad k \ge \frac{2\,|\ln(2\varepsilon)|}{1/4} \quad \Rightarrow \quad k \ge 8\,|\ln(2\varepsilon)|$$

Portanto, $k = \lceil 8|ln\ 2\varepsilon| \rceil$ repetições do Algoritmo III.9 serão suficientes para se garantir uma probabilidade de sucesso superior a $1-\varepsilon$.

A classe PP, definida logo a seguir engloba a classe *BPP*:

Definição III.11: (Classe *PP - Probabilistic Polynomial time*)

A classe *PP* consiste de todas as linguagens L com um algoritmo randômico A cujo pior caso seja polinomial . Além disso, $\forall x \in \{0,1\}^*$, obtêm-se:

$$\text{i) } x \in L \Rightarrow Pr(A(x) = 1) > 1/2$$
$$\text{ii) } x \notin L \Rightarrow Pr(A(x) = 0) > 1/2 \quad \bullet$$

Uma linguagem L tem *erro bilateral* se pertencer a PP ou BPP.

Considere L uma linguagem qualquer pertencente a PP. Suponha, sem perda de generalidade, que o Algoritmo III.9 seja utilizado novamente no reconhecimento de $x \in \{0,1\}^*$. Quantas repetições k do procedimento deverão ser realizadas para se garantir uma probabilidade de sucesso superior a $1-\varepsilon$? De maneira geral, se $L \in PP$ e $\beta > 0$ então:

$$\text{i) } x \in L \Rightarrow Pr(A(x) = 1) \geq 1/2 + \beta$$
$$\text{ii) } x \notin L \Rightarrow Pr(A(x) = 0) \geq 1/2 - \beta$$

Note que se $L \in BPP$ então $\beta = 1/4$.

Repetindo-se os mesmos passos utilizados acima pode-se mostrar que o número total de repetições k (no Algoritmo III.9) será:

$$k \geq \frac{2\,|\ln(2\varepsilon)\,|}{4\beta^2}$$

Observe que o número de repetições no procedimento será polinomial se $\beta = O(1/p|x|)$, onde $p(|x|)$ representa um polinomio qualquer no tamanho de da *string* x. Entretanto, se L é uma linguagem pertencente a PP, pode-se escolher $\beta = O(1/2^{p(|x|)})$, o que torna o número de repetições k exponencial (verifique)!!

As seguites proposições são verdadeiras: $P \subseteq ZPP$, $NP \subseteq PP$, $RP \cup co\text{-}RP \subseteq BPP$, RP $\subseteq$ *NP*, co-RP $\subseteq$ *co-NP*, BPP $\subseteq$ *PP* e $NP \cup co\text{-}NP \subseteq PP$ (veja Figura III.20). Para maiores detalhes sobre o assunto consulte Welsh [1983], Papadimitriou[1994], E. Waisbard, G. Weiss [1998].

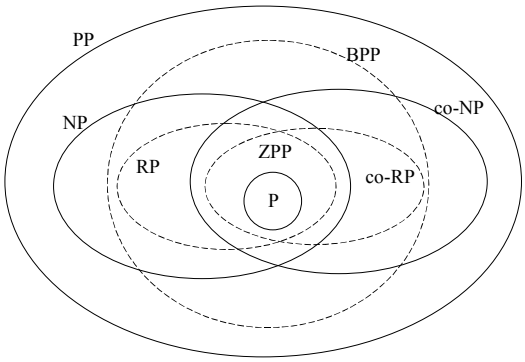


Figura III.20: Classes de Complexidade

Alguns problemas em aberto são listados a seguir: $P=NP$, $NP=co\text{-}NP$, $RP = co\text{-}RP$, $BPP \subseteq NP$, $RP \subseteq NP \cap co\text{-}NP$.

Capítulo IV

Algoritmos Aproximativos: Determinísticos e Randômicos

"Embora isso possa parecer um paradoxo,
toda a ciência é dominada pela idéia de aproximação"
Bertrand Russel

IV.1 - *INTRODUÇÃO:*

O crescente avanço dos algoritmos aproximativos pode ser atribuído, basicamente, à dificuldade de resolução de uma grande variedade de importantes problemas combinatórios. Na verdade, pode-se afirmar que grande parte dos problemas combinatórios de interesse prático são NP-Árduos! Isto significa que a possibilidade de resolvê-los em tempo polinomial não está totalmente descartada, mas é pouco provável que se consiga faze-lo. Esta situação pode ser bem sintetizada nas palavras de Garey e Johnson[1979 - pag3]: "*I can't find an efficient algorithm, but neither can all these famous people*".

A constatação que um determinado problema seja NP-árduo (ou NP-completo) nos coloca imediatamente diante de outra questão: qual a melhor estratégia de resolução a ser adotada? Embora garantam solução ótima, métodos exatos como a programação dinâmica, *backtracking, branch and bound* entre outros, demandam, geralmente, um elevado tempo de processamento (Nemhauser &Wolsey[1988], Wolsey[1998], Cook et. al.[1998]). Esta dificuldade se agrava drasticamente à medida que grandes instâncias são consideradas. A utilização de métodos exatos se justifica, especialmente, sob determinadas circunstâncias, onde as instâncias consideradas são suficientemente pequenas ou o tempo de processamento que se tem disponível é adequado o suficiente para a aplicação considerada. Além disso, em muitos problemas práticos, os dados de entrada são conhecidos apenas parcialmente. Isto significa que uma solução ótima dispendiosa em termos de tempo não se justifica em relação às soluções suficientemente "próximas" desta solução obtidas a um baixo custo computacional.

Os algoritmos aproximativos (especialmente os determinísticos aproximativos), foram introduzidos por Johnson[1974], e são algoritmos polinomiais que buscam sacrificar o mínimo possível da qualidade, obtida nos métodos exatos, ganhando, simultaneamente, o máximo possível em eficiência (tempo polinomial). Como discutido em Hochbaum[1997], a busca do equilíbrio entre estas situações conflitantes é o grande paradigma dos algoritmos aproximativos.

Antes do advento dos algoritmos aproximativos a análise de desempenho dos métodos heurísticos se baseava, simplesmente, em sua execução para um conjunto de finito de instâncias (*benchmark*). A performance da heurística era então comparada com a de outras heurísticas para o mesmo conjunto de instâncias considerado. Este tipo de comparação, ainda hoje bastante utilizado, retorna apenas uma medida parcial de desempenho já que o conjunto de instâncias é normalmente pequeno, além de não representar, satisfatoriamente, o conjunto de todas as instâncias associadas ao problema. Em outras palavras, uma heurística com bom desempenho para este conjunto finito não mantém, necessariamente, a mesma performance quando aplicada a outro conjunto de instâncias com características diferenciadas.

Neste capítulo serão discutidas algumas aplicações bem como aspectos teóricos associados aos algoritmos aproximativos. As seções IV.2 e IV.3 tratam dos algoritmos aproximativos determinísticos e randômicos respectivamente. A diferença central nestes dois tipos de abordagem reside no fato de que, no caso determinístico, execuções adicionais do procedimento aplicadas a uma mesma instância produzem sempre uma mesma saída com tempo de processamento sempre idêntico. Por outro lado, no caso probabilístico, solução gerada e tempo de processamento se modificam a cada nova repetição do procedimento, sendo, por isso, representadas convenientemente por variáveis aleatórias.

Muitos dos conceitos e definições apresentados na abordagem determinística (Seção IV.2.1), são extensíveis ao caso probabilístico. No caso, determinístico, estuda-se os algoritmos de aproximação relativa aplicados ao problema da Programação de Tarefas Independentes e Caixeiro Viajante (Seções IV.2.2 e IV.2.3). Para uma grande quantidade de problemas, algoritmos de aproximação relativa (ou absoluta) só serão possíveis se $P = NP$. Esta situação é exemplificada com a apresentação de resultados negativos para o Caixeiro Viajante (seção IV.2.4).

A seção IV.3, trata dos algoritmos randômicos aproximativos. As Seções IV.3.1, IV.3.2 e IV.3.3 tratam, respectivamente, do problema do Corte-Máximo em Grafos (*MAX-CUT*), *MAX-SAT* e o Problema Geral de Recobrimento (*General Covering Problem*). Uma atenção especial é dada ao problema de Recobrimento de Conjuntos (*Set Covering Problem*). Discute-se a técnica de Arredondamento Randômico (*Randomized Rounding*). Nela, formula-se inicialmente uma relaxação linear do problema. A solução relaxada define probabilidades para um procedimento randômico utilizado logo a seguir. Após esta etapa, uma versão determinística (*Derandomization*) pode ser obtida utilizando-se, por exemplo, o método das expectâncias condicionais (Seções IV.3.4 e IV.3.5 respectivamente).

IV.2 – ALGORITMOS DETERMINÍSTICOS APROXIMATIVOS

IV.2.1 – Conceitos Básicos e Definições:

Em um problema de otimização combinatória deseja-se minimizar (ou maximizar) uma função objetivo $f(.)$ sujeita a um conjunto discreto X de soluções viáveis. Seja Π um problema de otimização combinatória, e I, uma instância qualquer de Π[1]. Se A é um algoritmo aproximativo (determinístico ou randômico) para Π, então $x_A(I)$ é o valor da função objetivo gerado por A, $\forall I \in \Pi$. O valor da solução ótima associado será representado por $x^*(I)$.

Idealmente, nos algoritmos aproximativos, deseja-se obter uma solução que difira da solução ótima apenas por uma pequena constante. Medidas desse tipo serão denominadas *medidas de aproximação absoluta*. Algoritmos aproximativos que se encaixam nesse conceito para algum k positivo serão chamados *algoritmos de aproximação absoluta*. Mais formalmente, tem-se a seguinte definição:

Definição IV.1: (Algoritmos de aproximação absoluta)

Um algoritmo aproximativo A será de *aproximação absoluta* para um problema Π se, e somente se, para qualquer inteiro positivo k: $|x_A(I) - x^*(I)| \leq k, \forall I \in \Pi$. •

Note que a definição acima se aplica indistintamente para problemas de minimização e maximização. Eliminando-se o módulo da desigualdade, conclui-se diretamente que: $x_A(I) \leq x^*(I)+k$ para problemas de minimização, e $x_A(I) \geq x^*(I)-k$ para problemas de maximização (veja Figura IV.1(a)).

Claramente, conseguir um algoritmo de aproximação absoluta é o melhor que se espera obter para problemas NP-Árduos. Infelizmente, para uma grande quantidade de problemas, algoritmos de aproximação absoluta só existirão se $P=NP$! Em outras palavras, encontrar um algoritmo de

[1] Para facilitar a notação adotaremos um abuso de linguagem dizendo simplesmente que $I \in \Pi$.

aproximação absoluta para um problema otimização Π (NP-Árduo) poderá ser tão difícil quanto encontrar um algoritmo de complexidade polinomial para o problema de decisão associado! Em função disso, criou-se uma outra medida de desempenho denominada *medida de performance relativa*.

Garey, Graham e Ullman [1972] e posteriormente Johnson [1974] formalizaram o conceito de algoritmos aproximativos. Como discutido anteriormente, um algoritmo aproximativo deverá necessariamente ser polinomial no tamanho de qualquer instância para o problema.

Considere agora a seguinte definição:

Definição IV.2: (Algoritmo *f(n)*-aproximado)

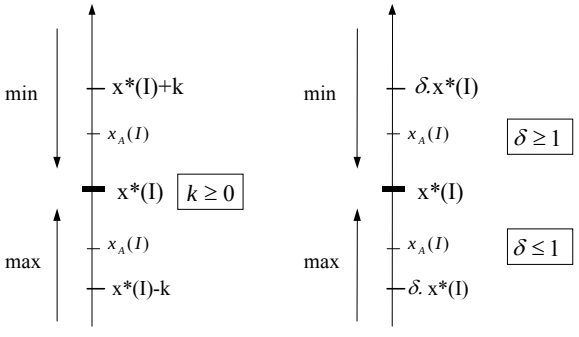
Um algoritmo A com solução $x_A(I)$ é *f(n)-aproximado* para um problema de minimização (de maximização) Π se, e somente se, qualquer que seja a instância I de tamanho *n*, a solução obtida é no máximo (no mínimo) *f(n)* vezes o valor da solução ótima $x^*(I)$. •

Observe através da definição acima que, se A é *f(n)*-aproximado, então $x_A(I) \leq f(n).x^*(I)$ para problemas de minimização, e $x_A(I) \geq f(n).x^*(I)$ para problemas de maximização. Normalmente, em problemas de maximização assume-se que $x^*(I) > 0$.

Definição IV.3: (Algoritmo δ-aproximado)

Um algoritmo *A f(n)*-aproximado é *δ-aproximado* para um problema de minimização (de maximização) Π se, e somente se, $f(n) \leq \delta$ $(f(n) \geq \delta)$ para algum $\delta > 0$. •

Naturalmente, deve-se ter $\delta \geq 1$ em problemas de minimização, e $0 < \delta \leq 1$ (ou $1/\delta \geq 1$) em problemas de maximização. Observe ainda que, quanto mais δ se aproxima de 1, melhor a qualidade da solução obtida pela heurística. A Figura IV.1.(b) ilustra bem estas duas situações. O parâmetro δ é também conhecido como *razão de performance absoluta* ou *fator de aproximação* do algoritmo *A*. Em problemas de maximização é também comum representar o fator de aproximação por $1/\delta$.



(a) Aproximação absoluta **(b) Algoritmos δ -aproximados**

Figura IV.1: Função objetivo p/ prob. de minimização e maximizaçã

De maneira geral, se um algoritmo aproximativo *A* é *(1+ε)*-aproximado, pode-se afirmar equivalentemente que:

$$\frac{|x_A(I) - x^*(I)|}{x^*(I)} \leq \varepsilon, \quad \forall I \in \Pi \ e \ \varepsilon > 0.$$

Neste caso, basta fazer $\delta = 1 - \varepsilon$ em problemas de maximização, e $\delta = \varepsilon + 1$ em problemas de minimização (verifique!).

Definição IV.4: (Esquema de aproximação polinomial)

Uma família[2] de algoritmos aproximativos para um problema Π, $\{A_\varepsilon\}_\varepsilon$, é chamada um *esquema de aproximação polinomial* se, e somente se, o algoritmo A_ε for $(1+\varepsilon)$-aproximado e seu tempo de processamento for polinomial no tamanho da entrada para ε fixo. •

Em outras palavras, o algoritmo polinomial A $(1+\varepsilon)$-aproximado pode ser visto como uma família de algoritmos $\{A_\varepsilon \mid \varepsilon > 0\}$. Vale ressaltar ainda que, se A_ε é algoritmo $(1+\varepsilon)$-aproximado para um problema de maximização então $1/\delta = \varepsilon + 1$ é fator de aproximação. Desta forma, o fator de aproximação de problemas de maximização e minimização são sempre maiores que 1, e a definição IV.4 se aplica perfeitamente a ambos os casos.

Para exemplificar esta definição, suponha que um algoritmo A_ε para um problema Π qualquer tenha complexidade igual a $n^{1/\varepsilon}$, onde $\varepsilon > 0$ e n é o tamanho do problema. Note que, embora $n^{1/\varepsilon}$ represente uma função polinomial em n, o tempo de processamento cresce bastante quando $\varepsilon \to 0$. Idealmente, deseja-se que o tempo de processamento cresça mais lentamente quando ε decresce. Esta situação pode ser mais bem formalizada através da definição de esquemas de aproximação totalmente polinomiais:

Definição IV.5: (Esquema de aproximação totalmente polinomial)

Uma família de algoritmos aproximativos para um problema Π, $\{A_\varepsilon\}_\varepsilon$, é chamada um *esquema de aproximação totalmente polinomial* se, e somente se, o algoritmo A_ε for $(1+\varepsilon)$-aproximado e seu tempo de processamento for polinomial no tamanho da entrada e $1/\varepsilon$. •

Note agora que, se A_ε tem complexidade igual a $(1/\varepsilon)^2 n^4$ para um problema Π qualquer, então A_ε define um esquema de aproximação totalmente polinomial para Π.

O exemplo seguinte, apresentado em Hochbaum[1997] ilustra alguns dos conceitos e definições apresentados acima:

Exemplo IV.1: (*O Problema da Programação de Tarefas Independentes*)

2.

O problema da Programação de Tarefas Independentes também conhecido na literatura como *Scheduling Independent Tasks* foi o primeiro problema resolvido por algoritmos aproximativos. Nele,4(a 0 10.68 341.1.9(i)2.ic13 0 TD0íbd(ks)]TJ-1.146s44-0.8J/TT16 1 T 8901106 0 TD0.0056 Tc0 Twn(.)Tj/TT2

conclua a última tarefa, no pior caso, apenas às 22:00h! Graham[1966] provou que esta estratégia garante uma solução com erro relativo não maior que 100%. Em outras palavras, enquanto o tempo de processamento das 147 tarefas consume um total de 12 horas, a obtenção da solução ótima pode durar no mínimo 6 horas de processamento mas não menos. Pode-se garantir, portanto, que as tarefas não estarão concluídas antes das 16:00h. De acordo com a Definição IV.3, tem-se um algoritmo 2-aproximado (ou 2-aproximativo). Posteriormente Graham observou que outra heurística poderia trazer melhores resultados. Ele provou que, atribuindo-se repetidamente a maior tarefa à primeira máquina disponível, uma solução 4/3-aproximada poderia ser obtida. Agora, se a última tarefa termina às 20:00h então 10 horas de processamento serão necessárias no pior caso (limite superior). Como a nova solução é 4/3-aproximada então uma solução ótima não consumirá menos do que 7, 5 horas de processamento, ou seja, o funcionário certamente não concluirá todas as tarefas antes das 17:30h.

Hochbaum e Shmoys[1987] desenvolveram posteriormente uma família de algoritmos $(1+\varepsilon)$-aproximativos para o mesmo problema. Assim, com a diminuição de ε tem-se, necessariamente, um aumento no tempo de processamento $t(\varepsilon)$ associado. A busca do equilíbrio entre estas situações conflitantes é ponto fundamental a ser observado. Por exemplo, suponha que para $\varepsilon=1/5$ (solução 6/5-aproximada), o tempo máximo de processamento $t(\varepsilon)$ seja igual a 9 horas. É fácil ver então que , embora o funcionário termine a última tarefa no máximo às 19:00h, um escalonamento ótimo não poderá ser obtido antes das 17:30h (verifique!). Como $t(1/5)=9$, pode-se tentar uma nova solução com maior grau de aproximação do escalonamento ótimo diminuindo-se ε gradativamente. Neste caso, como a última tarefa deverá estar concluída no máximo às 21:00h, então $t(\varepsilon)$ não poderá ser superior a 11 horas. •

A seguir são definidos alguns conceitos bastante utilizados na literatura. Define-se razão de performance em uma instância *I* particular e razão de performance absoluta:

Definição IV.6: (Razão de performance em uma instância *I*)

Seja A um algoritmo aproximativo para um problema de minimização Π. A *razão de performance na instância I*, representada por $R_A(I)$ é definida como:

$$R_A(I) = \frac{x_A(I)}{x^*(I)} \geq 1$$

Se Π é um problema de maximização então:

$$R_A(I) = \frac{x^*(I)}{x_A(I)} \geq 1$$ •

Note que, independentemente do problema Π ser de maximização ou minimização, a performance do algoritmo é melhor quando $R_A(I)$ se aproxima de 1.

Definição IV.7: (Razão de performance absoluta)

A *razão de performance absoluta* R_A de um algoritmo aproximativo *A* para um problema de otimização Π é definida por: $R_A = inf\{r \mid R_A(I) < r\ , \forall I \in \Pi\}$[3]. •

A Figura IV.2 ilustra bem este conceito. Note que R_A representa o ínfimo do intervalo (R_A, ∞). Ainda, se um algoritmo *A* é δ-aproximado para um problema de otimização Π então ele tem razão de performance absoluta igual a R_A.

Estudos experimentais indicam que a razão de performance $R_A(I)$ observada para uma instância *I* particular é, normalmente, bem inferior ao da razão de performance absoluta R_A Hochbaum[1997]. Isto se deve à presença de instâncias cujo desempenho do algoritmo não é

[3] Se $X \subset \Re$ então $a \in \Re$ é *ínfimo* (ou inf) de X se e somente se *a* for a maior das cotas inferiores do conjunto X. Lembre-se que $a \in \Re$ é *cota inferior* de X se e somente se $a \leq x, \quad \forall x \in X$.

satisfatório (pior caso). Uma maneira de contornar esta dificuldade é através de uma análise de comportamento médio. Neste caso, é importante que se conheça uma distribuição dos dados associados ao problema. Coffman, Garey e Johnson ilustram a utilização do caso médio aplicada ao *Bin Packing problem* (vide Hochbaum[1997] - Capítulo 2).

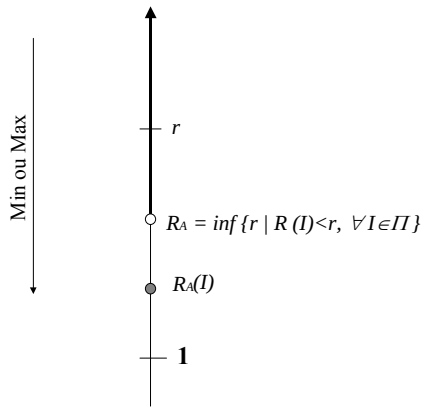


Figura IV.2: Razão de Performance Absoluta R_A

A menos que *P=NP,* a adoção algoritmos de aproximação relativa (ou absoluta), esquemas de aproximação polinomiais, totalmente polinomiais ou mesmo algoritmos δ-aproximados não pode ser aplicada indistintamente a todos os problemas combinatórios. Nas Seções IV.2.2 e IV.2.3, serão apresentados *dois* problemas clássicos da literatura resolvidos por algoritmos aproximativos e alguns resultados negativos associados (Seção IV.2.4).

VI.2.2 – O problema da Programação de Tarefas Independentes - PTI

No problema da Programação de Tarefas Independentes (instância particular do *makespan problem*) deseja-se atribuir *n* tarefas a *m* máquinas idênticas. Cada uma das *n* tarefas possuem um tempo t_i associado. O objetivo é minimizar o tempo de duração da última tarefa processada. As *m* máquinas devem funcionar em paralelo e estão inicialmente vazias. Como discutido anteriormente, este problema é NP-Árduo para $m \geq 2$.

Seja x_{ij} uma variável binária que indica se a tarefa i foi ou atribuída ao processador j. Se z representa o término de processamento da última tarefa tem-se então a seguinte formulação matemática para o problema:

$$\text{minimizar} \quad z$$

$$\text{sujeito a:} \begin{cases} \sum_{i=1}^{n} t_i x_{ij} \leq z, & j = 1,2,\ldots,m \\ \sum_{j=1}^{m} x_{ij} = 1, & i = 1,..,n \end{cases}$$

$$x_{ij} = 0 \text{ ou } 1 \quad \text{onde} \quad i, j = 1,..,n$$

O conjunto de desigualdades garante um limite superior para o tempo de duração de cada uma das *m* máquinas enquanto que o conjunto de igualdades obriga que cada tarefa seja atribuída a exatamente uma máquina.

Um algoritmo bastante simples para este problema, conhecido na literatura como algoritmo de escalonamento de listas (*List Scheduling Algorithm - LSA*) consiste simplesmente na atribuição de uma tarefa qualquer ao primeiro processador que se tornar ocioso. O processo é repetido até que todas as tarefas tenham sido concluídas. Dada uma instância qualquer, Graham [1966] provou que

esta heurística tem razão de performance na instância I menor ou igual a *2-1/m*. Note que, quando $m \to \infty$ então $R_A(I) \to 2$. Pode-se dizer portanto que o algoritmo de escalonamento de listas tem razão de performance absoluta igual a 2 (algoritmo 2-aproximado). Se *m=1,* tem-se uma instância particular onde o algoritmo de escalonamento de listas se torna exato ($R_A(I)=1$).

Proposição IV.1: Seja *I* uma entrada qualquer e $x_A(I)$, o tempo total de processamento do algoritmo de escalonamento de listas (representado por *A*) das *n* tarefas nas *m* máquinas. Então,

$$R_A(I) \leq \left(2 - \frac{1}{m}\right).$$

Além disso, tem-se garantida a existência de uma instância $I^* \in \Pi$ onde:

$$R_A(I^*) = \left(2 - \frac{1}{m}\right).$$

Prova: Sem perda de generalidade, considere *n* a última tarefa processada e t_n seu tempo de processamento associado. Além disso, considere que a tarefa *n* seja processada na máquina M_1. Note que nenhuma máquina deverá estar inativa ao final de $x_A(I)-t_n$ unidades de tempo. Caso contrário, existiria uma máquina M_k (para algum $k \neq 1$) cujo término seria anterior a $x_A(I)-t_n$. Tem-se portanto um absurdo pois, do algoritmo de escalonamento de listas, a inserção da tarefa *n* seria em M_k e não M_1.

Note agora que:

$$\sum_{i=1}^{n} t_i - t_n \geq m.\left(x_A(I) - t_n\right), \quad \forall I \in \Pi \qquad (01)$$

A desigualdade acima pode ser melhor compreendida com o auxílio das Figuras IV.3.(a) e IV.3.(b). A região *A* representa a área total definida por $m(x_A(I)-t_n)$ enquanto que a região *B* representa a área definida por:

$$\sum_{i=1}^{n-1} t_i - t_n .$$

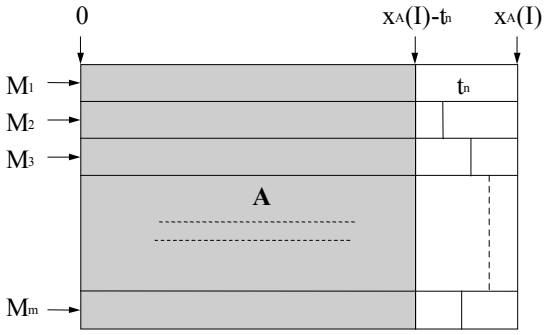


Figura VI.3.(a): Região A

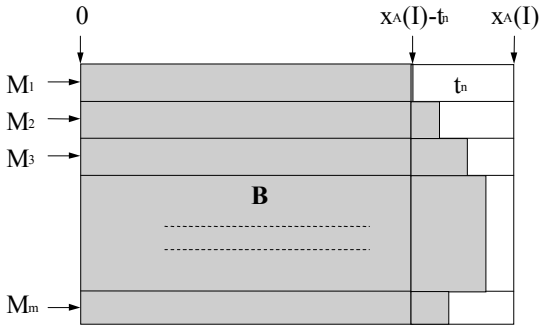


Figura VI.3.(b): Região B

É fácil ver que:

$$\frac{1}{m}\sum_{i=1}^{n} t_i \leq x^*(I), \quad \forall I \in \Pi \qquad (02)$$

define um limite inferior para o valor da solução ótima. Substituindo (02) em (01) obtém-se a seguinte expressão:

$$m.x^*(I) - t_n \geq m.\left(x_A(I) - t_n\right), \quad \forall I \in \Pi$$
$$m.x^*(I) \geq m.x_A(I) + t_n(1-m) \quad \forall I \in \Pi \qquad (03)$$

Como $t_n \leq x^*(I)$ e $m \geq 1$ então:

$$t_n(1-m) \geq x^*(I).(1-m) \qquad (04)$$

Substituindo (04) em (03) obtém-se:

$$m.x^*(I) \geq m.x_A(I) + x^*(I).(1-m) \quad \forall I \in \Pi$$

Após as devidas simplificações chega-se a:

$$\frac{x_A(I)}{x^*(I)} \leq \left(2 - \frac{1}{m}\right), \quad \forall I \in \Pi \cdot$$

Para mostrar a segunda parte da proposição considere uma instância *I** onde *n=m(m-1)+1.* Suponha ainda que o tempo de execução das *n* tarefas seja dado por $t_i=1$, para *i=1,..,n-1* e $t_n=m$. É fácil ver neste caso que *x*(I*) = m* (tempo de duração das *n* tarefas nas *m* máquinas) enquanto que $x_A(I^*) = 2m-1$ é o valor obtido pelo algoritmo de escalonamento de listas (vide Figura IV.4).

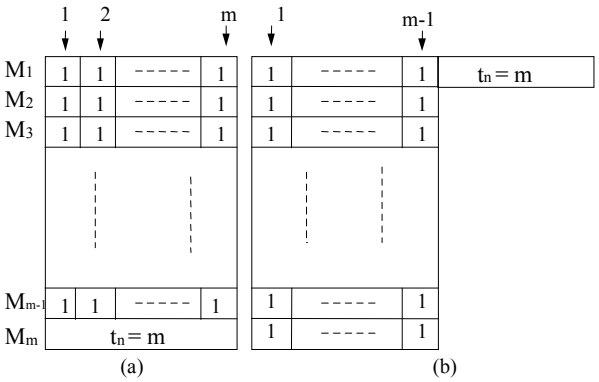


Figura IV.4: (a) Solução ótima $x^*(I^*)$ (b) Solução heurística $x_A(I^*)$

Na heurística *LPT* (*Longest Processing Time*) , os processos são ordenados inicialmente por ordem decrescente de tamanho. Em seguida, eles são colocados em operação nas máquinas ociosas sempre respeitando-se esta ordem. A complexidade total será de *O(nlogn)* iterações (Verifique!). O seguinte exemplo ilustra bem esta situação:

Exemplo IV.2: (Heurística LPT)

Considere que *n=12* tarefas devam ser alocadas a *m=3* máquinas distintas. O tamanho de cada tarefa é dado por:

$$t_1 = 6,\ \ t_2 = 6,\ \ t_3 = 5,\ \ t_4 = 5,\ t_5 = 4,\ t_6 = 4,\ t_7 = 3,\ t_8 = 3,\ t_9 = 3,\ t_{10} = 2,\ t_{11} = 1,\ t_{12} = 1$$

A solução esta representada na Figura IV.5. Note que, como os tempos de processamento de cada tarefa são inteiros segue que:

$$\left\lfloor \frac{1}{3}\left(\sum_{i=1}^{12} t_i\right) \right\rfloor + 1 = \left\lfloor \frac{43}{3} \right\rfloor + 1 = 15$$

é um limite inferior para o valor da solução ótima *x*(I).* Como $x_A(I)=15$, a solução obtida pela heurística também é solução ótima. Note que, $15 \leq x^*(I) \leq x_A(I) \leq 15$.

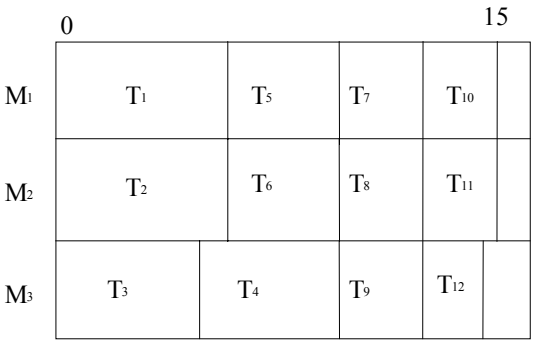


Figura IV.5: Heurística LPT

O resultado seguinte demonstrado por Graham[1966] (Proposição IV.3) garante um desempenho superior da heurística *LPT* em relação à heurística de escalonamento de listas. Considere entretanto o seguinte resultado auxiliar (Proposição IV.2). Sua demonstração será deixada como exercício.

Proposição IV.2: Suponha que, em uma solução ótima, não mais que 2 tarefas sejam associadas a cada máquina. Então, o escalonamento gerado pela heurística *LPT* também é ótimo.

Proposição IV.3: (Heurística LPT)

Seja $x_A(I)$ o tempo total de processamento da heurística *LPT* (representado por *A*) das *n* tarefas nas *m* máquinas. Então,

$$\frac{|x_A(I) - x^*(I)|}{x^*(I)} \leq \frac{1}{3} - \frac{1}{3m}.$$

Prova: A proposição se verifica imediatamente para $m=1$ (caso trivial). Considere então $m \geq 2$. Suponha, por absurdo, que exista um conjunto $S=\{t_1,t_2,...,t_n\}$ de tarefas onde a desigualdade não se verifica, ou seja:

$$\frac{|\theta_k - \theta_k^*|}{\theta_k^*} > \frac{|x_A(I) - x^*(I)|}{x^*(I)} > \frac{1}{3} - \frac{1}{3m} \qquad \text{(I)}$$

Além disso, considere S o conjunto com menor número de elementos satisfazendo essa propriedade. Sem perda de generalidade seja, $t_1 \geq t_2 \geq ... \geq t_n$ e $1,2,..,.n$ a ordem de escolha das *n* tarefas alocadas às *m* máquinas.

Seja *k* um índice associado a última tarefa processada. Pode-se mostrar neste caso que k=n. Se $k<n$ então o tempo de término das tarefas $1,2,...,k$ será dado por $x_A(I)=\theta_k$. Observando-se ainda as *k* primeiras tarefas conclui-se que $\theta_k^* \leq x^*(I)$ onde θ_k^* representa o tempo ótimo de processamento associado a estas *k* tarefas. Logo,

$$\frac{|\theta_k - \theta_k^*|}{\theta_k^*} > \frac{|x_A(I) - x^*(I)|}{x^*(I)} > \frac{1}{3} - \frac{1}{3m}$$

Observe agora que as tarefas $S'=\{t_1,t_2,...,t_k\}$ (onde $k<n$) podem ser alocadas aos *m* processadores de maneira que o resultado da proposição não se verifique. Isto é absurdo pois, considerava-se, *S* o menor conjunto satisfazendo (I). Logo $k=n$.

Como t_n é a última tarefa processada tem-se, de acordo com a heurística *LPT* que, no instante $x_A(I)-t_n$, nenhum outro processador estava ocioso naquele momento. Logo, seguindo raciocínio

análogo ao utilizado na Proposição IV.1 conclui-se que:

$$m.(x_A(I) - t_n) \leq \sum_{i=1}^{n-1} t_i$$

Então:

$$m.x_A(I) \leq \sum_{i=1}^{n-1} t_i + m.t_n = \sum_{i=1}^{n} t_i + (m-1).t_n$$

Dividindo esta desigualdade por $m \geq 2$ e lembrando que:

$$x^*(I) \geq \frac{1}{m}\sum_{i=1}^{n} t_i$$

Chega-se a:

$$x_A(I) - x^*(I) \leq \frac{m-1}{m} t_n$$

Como $x^*(I) > 0$ segue:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} \leq \frac{m-1}{m} \cdot \frac{t_n}{x^*(I)}$$

Da hipótese de absurdo tem-se:

$$\frac{1}{3} - \frac{1}{3m} < \frac{m-1}{m} \cdot \frac{t_n}{x^*(I)}$$

Multiplicando ambos os lados dessa desigualdade por $3m$ conclui-se que:

$$m - 1 < 3(m-1)\frac{t_n}{x^*(I)} \qquad \Rightarrow \qquad x^*(I) < 3t_n$$

Como t_n é a menor de todas as tarefas segue então que, em um escalonamento ótimo, não mais que 2 tarefas devem ser adicionadas a uma mesma máquina. Da Proposição IV.2, se a heurística *LPT* é aplicada a uma instância I, e um escalonamento ótimo possui no máximo 2 tarefas associadas a cada máquina então $x_A(I) = x^*(I)$. Logo:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} = 0$$

Observe que a situação acima é absurda pois considerava-se:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} > \frac{1}{3} - \frac{1}{3m} > 0, \quad \forall\ m \geq 2.$$

Conclui-se então que:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} \leq \frac{1}{3} - \frac{1}{3m}, \quad onde \quad I \in \Pi. \qquad \bullet$$

Observe que, para toda instância I com m é arbitrariamente grande, a razão de performance $R_A(I)$ se aproxima de *4/3* (algoritmo *4/3*-aproximado).

IV.2.3 – O problema do Caixeiro Viajante:

Considere $G=(V,E)$ um grafo completo não-orientado com pesos $c_{ij} \geq 0$ associados a cada aresta $[i,j] \in E$. No problema do caixeiro viajante, deseja-se determinar um ciclo de custo mínimo percorrendo todos os vértice de G uma única vez.

De maneira geral, se $A \subseteq E$ tem-se que:

$$c(A) = \sum_{[i,j]\in A} c_{ij}$$

representa o custo associado ao subconjunto *A* de arestas. Assim, no problema do caixeiro viajante deseja-se determinar um ciclo hamiltoniano *H** tal que $c(H^*) \leq c(H)$, qualquer que seja $H \subseteq E$.

Suponha que os pesos que definem a função de custo *c(.)* satisfaçam a desigualdade triangular, ou seja: $c_{ik} \leq c_{ij} + c_{jk}$, $\forall\, i, j, k \in V$. Este tipo de restrição é bastante comum e ocorre com freqüência em uma grande quantidade de situações práticas como, por exemplo, em problemas de transporte, energia elétrica, telecomunicações entre outros.

Considere *ΔPCV*, a notação utilizada para representação do problema do caixeiro viajante satisfazendo a desigualdade triangular. Observe por exemplo que, se os custos forem euclideanos a desigualdade triangular é satisfeita diretamente.

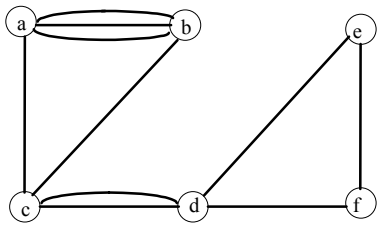
Rosenkrantz et al. [1977], descrevem várias heurísticas 2-aproximadas para o *ΔPCV*. A maior parte destas heurísticas se baseiam na geração inicial de ciclos eurelianos[4] seguida de uma seqüência de "atalhos" (*short-cutting*) visando a determinação de um ciclo hamiltoniano de custo mínimo.

A heurística seguinte, aqui representada por *AGM-CV*, constrói inicialmente uma árvore geradora mínima. Esta árvore é então utilizada na construção de um multigrafo contendo um ciclo eureliano. Em um multigrafo, a cada par de vértices podem estar associados uma ou mais arestas distintas.

O conceito de ciclo eureliano pode ser estendido facilmente para multigrafos, como na definição seguinte:

Definição IV.8: Seja *G* um multigrafo. Um ciclo eureliano em *G* é um caminho que percorre cada vértice de *G* pelo menos uma vez e cada aresta exatamente uma vez. •

A Figura IV.6 ilustra a presença de um ciclo eureliano em um multigrafo com 6 vértices.



Seqüência ---> a, b, a, b, c, d, e, f, d, c, a

Figura IV.6: Geração de um ciclo eureliano

A proposição seguinte caracteriza melhor a classe de multigrafos que contém ciclos eurelianos. A construção de um ciclo eureliano em tempo *O(m)* é conseqüência direta da prova deste teorema. Para maiores detalhes sobre sua demonstração veja Gibbons[1985] (pág. 155-157).

Proposição IV.4: Um multigrafo *G* contém um ciclo eureliano se e somente se *G* for conexo e todos os vértices tiverem grau par[5]. •

Após os conceitos e definições apresentados acima tem-se então o seguinte algoritmo

[4] Um ciclo em um grafo *G* é eureliano se todas as arestas de *G* forem visitadas exatamente uma vez.
[5] O grau de v pertencente ao multigrafo *G* é igual a soma de todas as arestas com extremidade no vértice v.

(desenvolvido por Rosenkrantz et al.[1977]) para "resolução" do *ΔPCV*. A seqüência de vértices H, inicialmente vazia, irá representar o ciclo hamiltoniano gerado pelo algoritmo.

Heurística IV.1: AGM-CV.
Início

1. Leia o grafo original *G* e faça $H \leftarrow \varnothing$.
2. Encontre uma árvore geradora mínima *T** em *G*.
3. Construa um multigrafo com arestas *T'* através da duplicação de cada uma das arestas de *T**.
4. Encontre um ciclo eureliano $\overline{E}$ em *T'* gerando uma seqüência de vértices: $v_1,v_2,...,v_k,v_1$. O vértice inicial v_1 é escolhido arbitrariamente.
5. Encontre um ciclo hamiltoniano H a partir de $\overline{E}$ da seguinte forma: Percorra cada um dos vértices da seqüência $v_1,v_2,...,v_k,v_1$. Se v_i (para $i \neq 1$) ocorre pela primeira vez, então adicione a aresta $[v_i\text{-}1, v_i]$ a ciclo *H*. Caso contrário, v_i é descartado e o próximo vértice da seqüência é pesquisado.
6. Retorna o ciclo hamiltoniano *H*.

fim.

Figura IV.7: Algoritmo AGM -CV

A execução da heurística *AGM-CV* pode ser ilustrada no exemplo da Figura IV.8. Neste caso, cada um dos vértices em *G* estão dispostos em uma "grade" com unidades de área iguais a 1. Logo, a distância euclideana pode ser calculada diretamente[6] (vide Figura IV.8(a)). Uma árvore geradora mínima está representada na Figura IV.8.(b). As arestas duplicadas e a orientação do ciclo eureliano $\overline{E}$, com origem no vértice *a*, estão representados na Figura IV.8(c). Note que a existência de $\overline{E}$ é garantida já que, no novo grafo resultante, todos os vértices possuem grau par (vide Proposição IV.4). A determinação do ciclo hamiltoniano (Figura IV.8(d)) é feita a partir da seqüência *a,b,e,g,e,b,f,d,c,h,c,d,f,b,a* após a eliminação de vértices repetidos em $\overline{E}$, excetuando-se obviamente o primeiro e o último nós (vértice *a*). É fácil ver que a solução gerada pela heurística não é ótima já que as arestas *[e,h]* e *[f,g]* se cruzam mutuamente (Verifique).

Como a etapa 4 do algoritmo tem complexidade *O(m)* (vide Gibbons[1985]), tem-se que o "maior esforço computacional" é obtido no cálculo da Árvore Geradora Mínima (etapa 3). Karger et. al [1995] por exemplo, apresentam um algoritmo randômico de complexidade *O(m)*.

A proposição seguinte garante que a solução gerada pela heurística *AGM-CV* é 2-aproximada.

Proposição IV.5: A heurística *AGM-CV*, quando aplicada ao *ΔPCV*, sempre retorna uma solução 2-aproximada.

Prova: Seja *H** uma solução ótima do problema do caixeiro viajante, e *c(H*)*, o valor da solução ótima correspondente. Se *T** é uma árvore geradora mínima, é fácil ver que $c(T^*) \leq c(T) \leq c(H^*)$, onde *T* é a árvore geradora resultante após a remoção de uma aresta qualquer de *H**. Logo, se cada uma das arestas de T* são duplicadas para a construção do ciclo eureliano $\overline{E}$, tem-se que $c(\overline{E}) = 2.c(T^*) \leq 2.c(H^*)$. Se H é a solução gerada pelo algoritmo *AGM-CV*, da utilização de "atalhos" e da desigualdade triangular conclui-se que $c(H) \leq c(\overline{E})$. Então $c(H) \leq 2.c(H^*)$. •

No exemplo da Figura IV.8(d) o custo da solução heurística *c(H)* é aproximadamente 22,997. Como esta solução é 2-aproximada pode-se afirmar que o valor ótimo *c(H*)* deverá ser superior a 11,498 (aproximadamente) e inferior a 22,997.

[6] Note que outras medidas de distância (ou métricas) podem ser utilizadas. Por definição, qualquer medida de distância deverá satisfazer a desigualdade triangular.

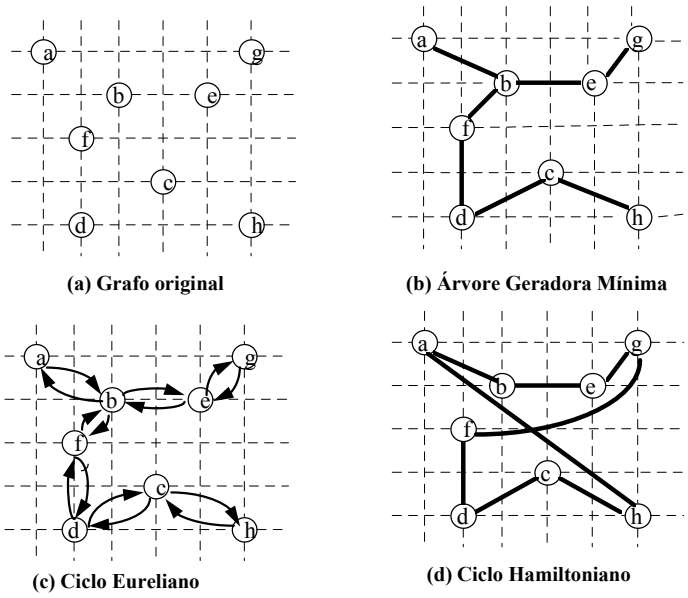
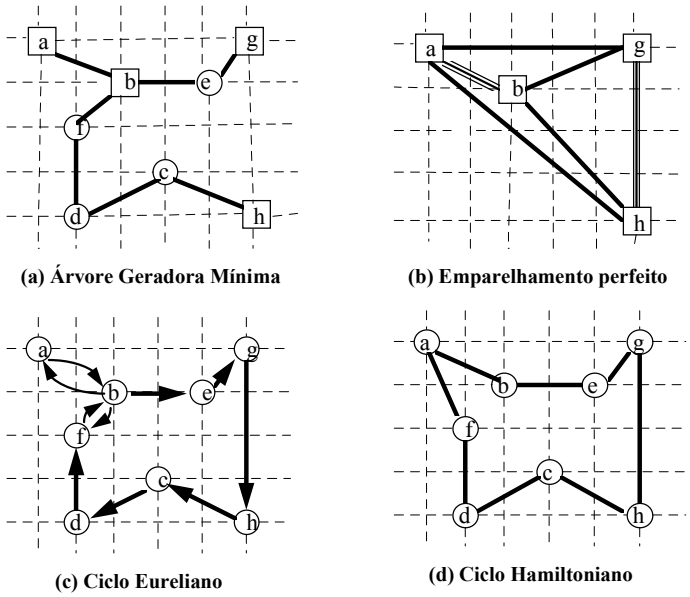


Figura IV.8: Heurística AGM-CV

Uma modificação desta heurística foi proposta por Christofides[1976]. A idéia básica neste caso, é gerar um grafo eureliano a partir de uma árvore geradora mínima evitando a duplicação de arestas, como ocorre por exemplo, na heurística *AGM-CV*. Esta abordagem de Christofides possibilitou a construção de um novo algoritmo 3/2-aproximado para o *ΔPCV*.

As proposições IV.4, IV.6 e IV.7, estabelecem as bases necessárias para construção do algoritmo de Christofides. Analogamente ao procedimento *AGM-CV,* o algoritmo proposto por Christofides determina inicialmente uma árvore geradora mínima *T** em um grafo *G* completo. Seja *V'*, o conjunto de todos os vértices de grau ímpar nesta árvore, e *G',* o subgrafo completo com extremidades em V'. Sabe-se da Proposição IV.6 que *V'* tem cardinalidade par. O subgrafo completo *G'* (induzido por *V'*) contém um emparelhamento perfeito (vide Proposição IV.7). Seja *M** o emparelhamento perfeito de custo mínimo. A Proposição IV.4 garante que a união de *M** e *T** irá formar um multigrafo contendo um ciclo eureliano. Analogamente ao algoritmo *AGM-CV*, este novo ciclo eureliano, através da operação de "atalhos", pode ser utilizado na construção de H. A Figura IV.9 ilustra melhor estas etapas. O grafo original *G* é o mesmo da Figura IV.8.(a) visto anteriormente.



(a) Árvore Geradora Mínima **(b) Emparelhamento perfeito**

(c) Ciclo Eureliano **(d) Ciclo Hamiltoniano**

Figura IV.9: Heurística de Christofides

Na Figura IV.9(a) estão representados por quadrados, todos os vértices de grau ímpar em *T** (conjunto *V'*). A Figura IV.9(b) representa o grafo induzido por *V'* (subgrafo completo) e o emparelhamento perfeito de custo mínimo formado pelas arestas do conjunto $M^* = \{[a,b], [g,h]\}$. A união das arestas de *T** com as arestas do emparelhamento M* estão representadas na Figura IV.9(c). O ciclo hamiltoniano *H* resultante da eliminação de vértices repetidos na seqüência *a, b, e, g, h, c, d, f, b, a* está representado na Figura IV.9(d). O algoritmo da Figura IV.11, sintetiza todas estas etapas:

Heurística IV.2: Christofides
Início

1. Leia o grafo original G e faça H←∅.
2. Encontre uma árvore geradora mínima T* em G.
3. Utilizando apenas os vértices de grau ímpar em T* construa um subgrafo completo G'.
4. Determine um emparellhamento perfeito M* de custo mínimo em G' e construa um multigrafo $\overline{G}$ =(V,T*∪M*).
5. Encontre um ciclo eureliano $\overline{E}$ em $\overline{G}$.
6. Utilizando atalhos, encontre um ciclo hamiltoniano H a partir de $\overline{E}$ respeitando-se a ordem estabelecida pelos vértices de $\overline{E}$.
7. Retorna o ciclo hamiltoniano H.

fim.

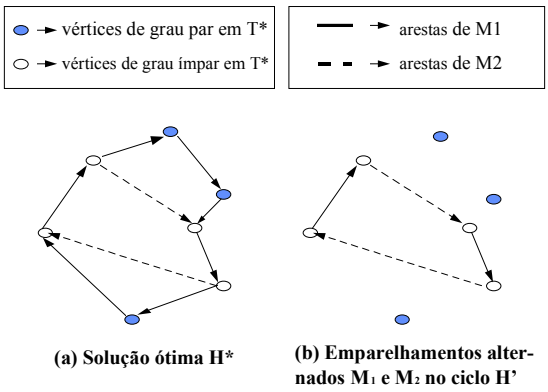
Figura IV.10 Heurística de Christofides para o ΔPCV.

Finalmente, a proposição seguinte, garante que a heurística de Christofides tem razão de performance igual a 3/2.

Proposição IV.8: A heurística de Christofides para o *ΔPCV* tem razão de performance absoluta igual a 3/2.

Prova: A idéia básica da demonstração consiste em se provar que o ciclo eureliano $\overline{E}$, obtido do multigrafo $\overline{G}$ *=(V, T*∪M*)* é tal que, $c(\overline{E}) = c(M^*) + c(T^*) \leq 3/2.c(H^*)$, onde *T** é árvore geradora mínima, *M** é emparelhamento perfeito de custo mínimo em *G'* (subgrafo completo induzido pelos vértices de *T**) e H* é uma solução ótima do *ΔPCV.*

Como *H** é solução ótima, um novo ciclo *H'* pode ser obtido através de "atalhos" eliminando-se de H* todos os vértices de grau par no grafo definido por T*. É óbvio então que $c(H') \leq c(H^*)$. Agora, o ciclo formado pelas arestas de H' define 2 emparelhamentos alternados, por exemplo M_1 e M_2 (como ilustrado no exemplo da Figura IV.11).



(a) Solução ótima H* **(b) Emparelhamentos alternados M_1 e M_2 no ciclo H'**

Figura IV.11: c(H')<c(H*)

Como *M** é emparelhamento perfeito de custo mínimo tem-se: $2c(M^*) \leq c(M_1) + c(M_2) \leq c(H^*)$ ou ainda $c(M^*) \leq (1/2).c(H^*)$. Como $c(T^*) \leq c(H^*)$, somando-se estas duas últimas desigualdades chega-se a: $c(M^*) + c(T^*) \leq 3/2.c(H^*)$.

Para finalizar a demonstração, note que a existência de um ciclo eureliano $\overline{E}$ em $\overline{G}$ é garantida já que todos os vértices de $\overline{G}$ tem grau par (Proposição IV.4). Com a utilização de "atalhos" , pode-se eliminar todos os vértices repetidos presentes na seqüência definida por $\overline{E}$. Assim, a nova solução H obtida pelo algoritmo de Christofides satisfaz $c(H) \leq c(\overline{E})$. Conclui-se portanto que: $c(H) \leq c(\overline{E}) \leq 3/2.c(H^*)$. •

Voltando ao exemplo da Figura IV.9, note que o custo da solução heurística *c(H)* é aproximadamente 18,358. Como esta solução é 3/2-aproximada pode-se afirmar que o valor ótimo *c(H*)* deverá ser superior a 12,237 (aproximadamente) e inferior a 18,358 (verifique!).

Uma desvantagem da heurística de Christofides é o custo computacional necessário para se calcular um emparelhamento perfeito de custo mínimo *M**. Como descrito em Vadya[1988], esta operação é realizada em $O(n^{2,5}log^4 n)$ iterações ao passo que, na heurística *AGM-CV* (2-aproximada), a determinação do ciclo eureliano e a determinação da árvore geradora mínima são obtidas em tempo linear (vide Gibbons[1985] e Karger et. al [1995] respectivamente).

Koutsoupias et. al.[1995] desenvolveram um esquema de aproximação polinomial para o problema do caixeiro viajante em grafos planares. Mais recentemente, S. Arora[1996] apresentou um esquema de aproximação polinomial para o problema do caixeiro viajante com distâncias euclideanas em $\Re^2$. Este algoritmo de complexidade $n^{O(1/\varepsilon)}$, representa, ao menos teoricamente, um avanço significativo em relação ao trabalho de Christofides. Seu trabalho consiste basicamente, no

particionamento recursivo do plano de maneira que poucas arestas do circuito hamiltoniano se interceptam com as fronteiras da partição. Na prática, apesar de lento, Arora sugere alternativas de decomposição de instâncias do Caixeiro Viajante tornando-o atrativo para implementações em paralelo.

Seguindo de perto a abordagem utilizada por Christofides[1976] para o *ΔPCV,* Gendreau et al.[1997] apresentam um algoritmo 3/2-aproximado para uma extensão do caixeiro viajante conhecida como caixeiro viajante com pontos de coleta e entrega de mercadorias (*Traveling Salesman Problem with Backhauls*). Nele, deve-se determinar um ciclo hamiltoniano de custo mínimo de maneira que um subconjunto de vértices $L \subset V$ (*linehaul custumers*) seja visitado antes de outro conjunto $B \subset V$ (*backhaul customers*).

IV.2.4 - Resultados Negativos para Algoritmos de Aproximação Relativa

Nesta seção, serão discutidas algumas situações onde a determinação de soluções δ-aproximadas para $\delta > 0$, só será possível se $P=NP$. Uma questão fundamental neste caso é estabelecer quais problemas admitem ou não soluções aproximadas. Infelizmente, a teoria da NP-completude associada a problemas de decisão não fornece nenhum *insight* a este respeito.

A proposição seguinte, demonstrada por Sahni e Gonzalez[1976] mostra que, no Caixeiro Viajante, quando se remove a hipótese da desigualdade triangular a determinação de um algoritmo polinomial δ-aproximado só será possível se $P=NP$.

Proposição IV.9: Se $P \neq NP$ e $\delta \geq 1$ então não existe algoritmo polinomial δ-aproximado para o problema do caixeiro viajante.

Prova: Suponha, por absurdo, que exista um algoritmo polinomial *A* δ-aproximado para o problema do caixeiro viajante. Sem perda de generalidade, através de um arredondamento, pode-se supor δ inteiro, se necessário. Isto será interessante sempre que custos inteiros estiverem associados às arestas do grafo.

A idéia da demonstração consiste em, utilizando-se o algoritmo *A* como ferramenta, resolver o problema do ciclo hamiltoniano em um grafo qualquer $G=(V,E)$ não-valorado. Desta forma, estar-se-á resolvendo o problema do ciclo hamiltoniano em tempo polinomial, o que é absurdo, pois, por hipótese, $P \neq NP$. Um problema NP-Completo pertence a P se, e somente se, $P=NP$ (Garey& Johnson[1979]).

Seja $G=(V,E)$ uma instância qualquer para o problema do ciclo hamiltoniano. Um grafo completo e valorado $G'=(V',E')$ pode ser construído fazendo-se:

$$V = V'; \qquad E' = \{[i,j]: i,j \in V \ e \ i \neq j\} \qquad \text{e} \qquad c_{ij} = \begin{cases} 1, & se\ [i,j] \in E, \\ \delta|V| + 1, & caso\ contrario \end{cases}$$

Observe que *G'* é obtido facilmente, através de *G* em tempo polinomial no tamanho de *V* e *E* respectivamente. Contrariamente ao problema do caixeiro viajante com distâncias euclideanas, neste caso, os valores associados às arestas de E' não satisfazem a desigualdade triangular.

Se *G* contém algum ciclo hamiltoniano então o valor da solução ótima para o problema do caixeiro viajante com instância *G'* será igual a $|V|$. Entretanto, pode-se afirmar que, se G não contém nenhum ciclo hamiltoniano então qualquer ciclo hamiltoniano *H* em *G'* deverá conter alguma aresta não pertencente a *E*. Assim,

$$c(H) = \sum_{[i,j] \in H} c_{ij} \geq \delta(|V|+1) + (|V|-1) > \delta|V| \qquad \text{onde } H \subset E'.$$

Como A é algoritmo polinomial δ-aproximado, se G contém ciclo hamiltoniano, segue que

$c(H) = c(H^*) = |V|$ onde H é a solução gerada por A e H^* é solução ótima. Isto se deve ao fato de que, nenhuma outra solução H com custo maior que |V| e menor ou igual a $\delta|V|$ pode ser obtida.

Se G não contém ciclo hamiltoniano então, a solução H, gerada por A é tal que $|V| \leq c(H)$. Ainda, da desigualdade (I) conclui-se que $c(H) > \delta|V|$. Logo, o algoritmo polinomial A (aplicado a G') pode ser utilizado, para decidir se o grafo original G contém ou não ciclo hamiltoniano. O que é absurdo, pois, por hipótese, $P \neq NP$. •

IV.3 – ALGORITMOS RANDÔMICOS APROXIMATIVOS

Seja Π um problema de otimização combinatória NP-Árduo e I uma instância qualquer de Π. Neste caso, a adoção de um algoritmo randômico polinomial A pode ser interessante se sua solução for representada por uma variável aleatória $x_A(I)$ com valor esperado idealmente próximo da solução ótima $x^*(I)$.

Nos problemas resolvidos pelo método de Las Vegas (discutidos no Capítulo III), obtinha-se sempre uma solução exata com tempo esperado polinomial. No método de Monte Carlo, uma solução era obtida, com alta probabilidade de acerto e tempo (esperado) também polinomial. Ambos tratavam de problemas da classe P e o principal parâmetro observado era o tempo de processamento. Contrariamente, nesta seção, será dada uma atenção especial à solução de problemas NP-Árduos.

Na determinação de uma solução heurística $x_A(I)$, pode-se, por exemplo, atribuir valores aleatóriamente às variáveis associadas ao problema, satisfazendo obviamente alguma distribuição estatística. Outra possibilidade interessante é a adoção de Programação Linear ou Programação Semidefinida[8] na geração de limitantes inferiores para valor ótimo do problema original. Nas duas situações será possível utilizar algoritmos polinomiais baseados nos métodos de pontos interiores (vide Wright[1997][9] Esta técnica é mais conhecida na literatura como Arredondamento Randômico (*Randomized Rounding*). As soluções relaxadas em ambos os casos, definirão probabilidades a serem utilizadas na etapa randômica. Nos algoritmos randômicos aproximativos, solução esperada e tempo esperado de processamento serão parâmetros importantes a serem observados sendo representados por variáveis aleatórias discretas. Os trabalhos de Srinavasan[1995] e [2001] são uma ótima referência sobre Arredondamento Randômico e Relaxação Linear na solução de problemas combinatórios.

Considere a seguinte definição de algoritmo randômico δ-aproximado:

Definição IV.9: (Algoritmo randômico δ-aproximado)

Seja Π um problema de minimização. Um algoritmo randômico A de complexidade polinomial é δ-aproximado para Π se, e somente se, $E(x_A(I)) \leq \delta.x^*(I)$, onde $\delta \geq 1$. Se Π é um problema de maximização então A é δ-aproximado se e somente se $E(x_A(I)) \geq \delta.x^*(I)$, onde $\delta \leq 1$.•

Em outras palavras, pode-se dizer que, se A é um algoritmo randômico δ-aproximado para um problema de minimização então $Pr[x_A(I) \leq \delta.x^*(I)] \geq 1/2$ onde $\delta \geq 1$. Analogamente, $Pr[x_A(I) \geq \delta.x^*(I)] \geq 1/2$ para $\delta \leq 1$ em problemas de maximização (Vazirani[1997]). Verifique!

IV.3.1 - *O problema do Corte-Máximo em Grafos*

Suponha $G=(V,E)$ um grafo não-orientado com pesos não-negativos w_{ij} associados a cada aresta $[i,j] \in E$. Considere ainda $|V| = n$ e $|E| = m$. No problema do corte-máximo deseja-se encontrar uma partição de $V=\{v_1, v_2,...,v_n\}$ em 2 subconjuntos V_1 e V_2 de vértices de maneira que o somatório dos pesos das arestas com extremidades em V_1 e V_2 respectivamente seja maximizado (vale lembrar que V_1 e V_2 definem uma partição de V se e somente se $V=V_1 \cup V_2$ e $V_1 \cap V_2 = \varnothing$.

[8] Para maiores detalhes sobre a utilização de Programação Semidefinida e Algoritmos Randômicos em Otimização Combinatória consulte Goemans&Willianson[1994],[1995].
[9] Os métodos de pontos interiores foram introduzidos inicialmente por Karmarkar em 1984.

Contrariamente ao problema do corte-mínimo (pertencente à classe *P*) o problema do Corte Máximo é NP-Árduo e é conhecido por sua aplicação em áreas distintas como, p. ex., física estatística e Circuitos VLSI (Barahona et al [1988]), *Network Design* (Barahona [1996]) entre outras. Este problema continua NP-Árduo mesmo quando $w_{ij}=1$, $\forall\ i,j \in V$ (Garey&Johnson[1979]). Orlova e Dorfman [1972] e Hadlock[1975], discutem classes especiais de grafos resolvidas em tempo polinomial, como por exemplo, os grafos planares. Maiores detalhes sobre este problema podem ser encontrados em Poljak e Z. Tuza [1995].

Sahni e Gonzalez[1976] apresentaram um algoritmo randômico *1/2*-aproximado considerando pesos unitários associados a cada aresta $[i,j] \in E$ (vide Figura IV.13). A idéia neste caso, consiste simplesmente na escolha aleatória de elementos de V_1 e V_2 respectivamente.

Algoritmo IV.3: Corte-Máximo Randômico - CMR
Início

$V_1 \leftarrow V_2 \leftarrow \varnothing$;
Para *i=1* **até** *n* **faça**
x ← jogue uma moeda;
Se *x = cara* **então** insira v_i em V_1;
Se *x = coroa* **então** insira v_i em V_2;
fim;
Retorne V_1 e V_2;
fim.

Figura IV.12: Corte-Máximo Randômico

Como exemplo considere o grafo com 5 vértices e 7 arestas representado na figura seguinte:

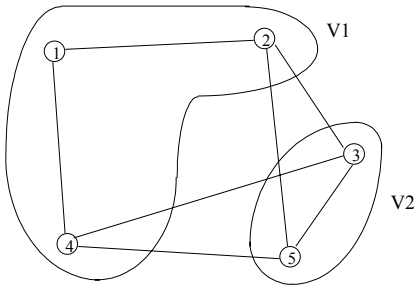


Figura IV.13: Corte-máximo randômico

Seja *X* uma variável aleatória que representa o número de arestas pertencentes ao corte(V_1,V_2). Para mostrar que o algoritmo *CMR* é randômico 0,5-aproximado basta provar que $E(X) \geq 1/2.x^*(I)$, onde $x^*(I)$ representa o número de arestas pertencentes ao corte máximo. Para demonstrar esse fato, basta notar que m (número de arestas de *G*) define um limite superior para o valor da solução ótima, ou seja, $x^*(I) \leq m$. Mostrando que $E(X) \geq 1/2.m$, prova-se então o resultado.

Considere $I(e_i)$ para $i \in \{1,.., m\}$, uma variável aleatória que indica se a aresta e_i de *E* pertence ou não ao corte máximo. Desta forma, $I(e_i) = 0$ ou *1*. Para determinar $Pr(I(e_i)=1)$ e $Pr(I(e_i)=0)$ respectivamente monta-se um experimento onde uma moeda equilibrada é jogada para cada uma das extremidades de $e_i=(u,v)$. Segue então que $Pr(I(e_i)=0) = 1/2$ (duas *caras* ou duas *coroas* ocorrem simultaneamente) ou $Pr(I(e_i)=1) = 1/2$ (*cara* e *coroa* ocorrem alternadamente). Assim, da definição do valor esperado tem-se:

$$E(I(e_i)) = 1.\Pr(I(e_i) = 1) + 0.\Pr(I(e_i) = 0) = 1/2$$

Como $X = I(e_1) + I(e_2) + ... + I(e_m)$ segue, da linearidade do valor esperado (Capítulo II) que:

$$E(X) = E\left(\sum_{i=1}^{m} I(e_i)\right) = \sum_{i=1}^{m} E\big(I(e_i)\big) = \frac{m}{2} \geq \frac{x^*(I)}{2}.$$

Equivalentemente, pode-se dizer que, em uma iteração do Algoritmo CMR tem-se *Pr(X>0,5.x*(I))* (probabilidade de sucesso). Assim, a garantia de uma solução *0,5*-aproximada pode ser incrementada repetindo-se o processo um número pré-determinado de vezes. Por exemplo, se *k=10* então a probabilidade de falha no algoritmo randômico será menor ou igual a 1/1024!

Utilizando técnicas mais sofisticadas, Goemans e Williamson apresentaram em 1994 um algoritmo 0,878-aproximado para o problema do Corte-Máximo em Grafos. Neste algoritmo, eles utilizaram uma relaxação baseada em Programação Semidefinida seguida de um arredondamento randômico (*Randomized Rounding*) na determinação de uma solução viável. Esta técnica inovadora vem sendo utilizada, a partir de então, como método aproximativo para outros problemas NP-Árduos. Em 2000, Goemans e Williamson foram laureados com o prêmio Fulkerson em reconhecimento a este trabalho[10]. Outros exemplos de aplicação desta técnica podem ser encontrados em Motwani et. al [1997] e Anderson[2000].

IV.3.2 - *O problema MAX-SAT (Maximum Satisfiability problem)*

Considere uma expressão booleana *B* na forma normal conjuntiva, ou seja, *B* será formada por uma conjunção de cláusulas e uma disjunção de literais. No problema *MAX-SAT* deseja-se determinar o maior número possível de cláusulas que sejam simultaneamente verdadeiras. Assume-se, sem perda de generalidade, que nenhuma cláusula contenha um literal *x* juntamente com seu complemento $\bar{x}$, caso contrário, qualquer atribuição (*V* ou *F*) a este literal tornaria a cláusula correspondente verdadeira. O problema MAX-SAT é NP-Árduo, já que o problema de decisão SAT associado é NP-Completo (Garey&Johnson[1979]).

A expressão booleana *B* com cláusulas C_j, para *j=1,2,...,m* pode ser representada sucintamente por:

$$B = \bigwedge_{j=1}^{m} C_j \ , \quad \text{onde} \quad \left|C_j\right| = k_j , \quad \textit{para} \ \ j = 1,..,m.$$

Exemplo IV.3: Para ilustrar a definição acima considere a seguinte expressão booleana na forma normal conjuntiva:

$$B = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_2) \wedge (x_1)$$

É fácil ver, por simples inspeção, que o número máximo de cláusulas em *B* simultaneamente verdadeiras é igual a 3. Na versão decisão associada (*Satisfiability Problem-SAT*) deseja-se determinar simplesmente se *B* é *falsa* ou *verdadeira* para alguma atribuição a seus literais. É fácil ver neste caso que *B* será sempre *falsa* (verifique). •

Seja *x*(B),* o número máximo de cláusulas simultaneamente verdadeiras e $x_A(B)$ o número de cláusulas verdadeiras obtidas por um algoritmo *A* qualquer após a atribuição de valores *V* ou *F* (*verdadeiro* ou *falso*) aos literais de *B*.

Um algoritmo randômico bastante simples para o *MAX-SAT* (representado aqui por A_1) pode ser obtido atribuindo-se *V* ou F aos literais de *B* com probabilidade *1/2*. Suponha neste caso que Z_j para $j \in \{1,..,m\}$, seja variável aleatória associada à cláusula C_j. Assim, $Z_j = 1$ se, e somente se, C_j é

[10] A concessão do Prêmio Fulkerson ocorre, no mínimo, após 6 anos da data de publicação do trabalho. A justificativa para este fato é que, um trabalho relevante, necessita de um prazo que seja suficiente para sua divulgação e reconhecimento na comunidade científica.

verdadeira, e $Z_j = 0$ caso contrário. É fácil ver então que:

$$\Pr(Z_j = 0) = 1/2^{k_j}, \qquad (1)$$
$$\Pr(Z_j = 1) = 1 - 1/2^{k_j}$$

Seja $x_{A1}(B) = Z_1 + Z_2 + ... + Z_m$ o número total de clásulas verdadeiras obtidas por A_1. Segue, da linearidade do valor esperado que:

$$E(x_{A_1}(B)) = \sum_{j=1}^{m} E(Z_j) \qquad (2)$$

onde $E(Z_j) = 1.Pr(Z_j=1) + 0.Pr(Z_j=0)$. Fazendo-se $k = min\{k_j \mid j=1,...,m\}$ e substituindo (1) em (2) chega-se a:

$$E(x_{A_1}(B)) = \sum_{j=1}^{m} E(Z_j) = \sum_{j=1}^{m} \left(1 - \frac{1}{2^{k_j}}\right) \geq m.\left(1 - \frac{1}{2^{k}}\right)$$

Como $m \geq x^*(B)$ e $k \geq 1$ conclui-se que $E(x_{A1}(B)) \geq (1/2).x^*(B)$ (solução randômica 0,5-aproximada). Observe por exemplo que, se cada uma das m cláusulas contém pelo menos 2 literais então a solução obtida é 0,75-aproximada. Pode-se, portanto, constatar que a solução do problema é dificultada em função da presença de cláusulas com um único literal.

Uma outra abordagem bastante interessante, utiliza as coordenadas de uma solução de um problema de Programação Linear como probabilidades de escolha para cada um dos literais de B (para maiores detalhes sobre Programação Linear vide Bazarra et al.[1990]). Suponha que $z_j \in \{0,1\}$ seja variável binária indicando respectivamente se a cláusula C_j é falsa ou verdadeira. Analogamente, seja $y_i \in \{0,1\}$ variável binária onde $y_i = 1$ se, e somente se, o literal x_i associado for *verdadeiro* e $y_i = 0$, caso contrário. Considere ainda C_j^+ o conjunto de todos os índices i associados a literais x_i de C_j não representados em sua forma complementar e C_j^-, o conjunto de todos os índices i associados a literais $\bar{x}_i$ de C_j (em sua forma complementar)[11]. Suponha que o número total de literais x_i (na sua forma complementar ou não), bem como o número de variáveis y_i associadas seja igual a n. O problema *MAX-SAT* pode então ser formulado como um problema de Programação Linear Inteira (PLI) da seguinte forma:

$$\max \sum_{j=1}^{m} z_j$$

$$sujeitoa: \begin{cases} \sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq z_j \; \forall j \in \{1,..,m\} \\ y_i, z_j \in \{0,1\}, \; p/ \; i = 1,...,n \; e \; j = 1,..,m \end{cases}$$

Note que, se uma das variáveis y_i (com índices em C_j^-) for 0 ou uma das variáveis y_i (com índices em C_j^+) for 1 para que a restrição correspondente à cláusula C_j será satisfeita. Pode-se fazer então $z_j = 1$.

Um modelo de Programação Linear (PL) é obtido após a substituição das restrições de integralidade $y_i, z_j \in \{0,1\}$, por $y_i, z_j \in [0,1]$, $\forall i,j$. Sejam $\hat{y}_i$ e $\hat{z}_j, \forall i,j$, os valores obtidos (solução ótima) após a resolução desta relaxação linear. É fácil ver neste caso que $\hat{z}_1 + ... + \hat{z}_m$ define um limite superior para o valor da solução ótima $x^*(B)$.

Um novo algoritmo randômico (representado aqui por A_2) pode ser obtido fazendo-se $Pr(y_i=1) = \hat{y}_i$ e $Pr(y_i=0) = 1 - \hat{y}_i$, para $i=1,2,..,n$. A linha 1 retorna um vetor solução $\hat{y} \in [0,1]^n$ do problema de Programação Linear (PL). A linha 2, utiliza o vetor $\hat{y}$ na geração de uma solução inteira

[11] Por vezes, utilizaremos um abuso de linguagem dizendo simplesmente que a cláusula C_j será representada equivalentemente por seu conjunto de índices $C_j = C_j^+ \cup C_j^-$.

$y \in \{0,1\}^n$. Finalmente, na linha 3, determina-se um vetor inteiro $z \in \{0,1\}^m$ com custo $x_{A2}(B)$ associado. Note que $x_{A2}(B)$ representa exatamente o número de cláusulas simultaneamente verdadeiras associadas à solução $y \in \{0,1\}^n$.

Procedimento IV.4: MAX-SAT Randômico (A2)
Início

1. Resolve *PL* retornando $\hat{y}$;
2. **Para** $i = 1$ **até** n **faça**
   - Joga moeda viciada c/ $Pr(cara) = \hat{y}_i$;
   - **Se** moeda = <cara> **então** $y_i = 1$
   - **senão** $y_i = 0$;

   fim;
3. **Para** $j = 1$ **até** m **faça** {*Calcula* $x_{A2}(B) = z_1 + ... + z_n$}
   - $z_j \leftarrow 0$; $i \leftarrow 1$;
   - **Enquanto** $i \leq n$ **faça**
     Se ($y_i = 1$ *e* $i \in C_j^+$) ou ($y_i = 0$ *e* $i \in C_j^-$) **então**
     $z_j \leftarrow 1$;
     $i \leftarrow n+1$;
     senão $i \leftarrow i+1$;
     fim;

   fim;
4. Utilizando o vetor inteiro $z \in \{0,1\}^n$, calcula $x_{A2}(B)$;

Fim.

Figura IV.14: MAX-SAT Randômico

Como discutido anteriormente, para se determinar a qualidade da solução heurística $x_{A2}(B)$, deve-se calcular $E(x_{A2}(B)) = E(z_1) + ... + E(z_n)$. Como $E(z_j) = Pr(z_j = 1)$, p/ $j = 1,2,...,m$, tem-se que:

$$\Pr(z_j = 0) = \prod_{i \in C_j} (1 - \hat{y}_i)$$
$$\Pr(z_j = 1) = 1 - \prod_{i \in C_j} (1 - \hat{y}_i) \qquad (3)$$

Da relaxação linear (PL) tem-se que:

$$\sum_{i \in C_j^+} \hat{y}_i + \sum_{i \in C_j^-} (1 - \hat{y}_i) \geq \hat{z}_j \quad \forall j \in \{1,..,m\}.$$

Além disso, como $|C_j| = k_j$, $Pr(z_j = 1)$ pode ser limitado inferiormente fazendo-se $\hat{y}_i = \hat{z}_j / k_j$ (verifique). Assim, de (3), tem-se que $Pr(z_j = 1) \geq 1 - (1 - \hat{z}_j / k_j)^{kj}$. A função $f(\hat{z}_j) = 1 - (1 - \hat{z}_j / k_j)^{kj}$ é côncava no intervalo $[0,1]$ já que $f'(\hat{z}_j) \geq 0$, $f''(\hat{z}_j) \leq 0$ (verifique). Como $f(0) = 0$ e $f(1) = 1 - (1 - 1/k_j)^{k_j}$ tem-se:

$$1 - \left(1 - \hat{z}_j / k_j\right)^{k_j} \geq \left(1 - (1 - 1/k_j)^{k_j}\right) \hat{z}_j \geq (1 - 1/e).\hat{z}_j .$$

A Figura IV.15 ilustra melhor esta operação. Note que a função côncava $f(\hat{z}_j)$ é limitada inferiormente pela função linear $(1 - (1 - 1/k_j)^{kj})\,\hat{z}_j$ no intervalo $[0,1]$. Além disso, pode-se mostrar facilmente que $(1 - 1/k)^k < e^{-1}$, $\forall\, k > 0$. Basta que $(1 + x) < e^x$, $\forall\, x$ (aproximação linear de e^x). Assim, fazendo-se $x = -1/k$ p/ $k > 0$ conclui-se que $(1 - 1/k) < e^{-1/k}$, e portanto, $(1 - 1/k)^k < e^{-1}$. Logo, $Pr(z_j = 1) \geq (1 - 1/e)\,\hat{z}_j = 0{,}632.\,\hat{z}_j$.

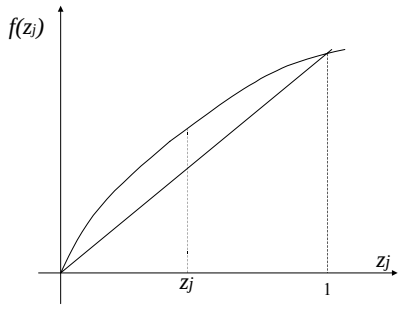


Figura IV.15: Concavidade de *f(zj)*

Da linearidade do valor esperado tem-se que:

$$E(x_{A_2}(B)) = \sum_{j=1}^{m} E(z_j)$$

Substituindo $Pr(z_j=1)$ em $E(z_j)=Pr(z_j=1)$ e lembrando que $\hat{z}_1+...+\hat{z}_m \geq x^*(B)$ conclui-se que $E(x_{A2}(B)) \geq (1\text{-}1/e).x^*(B)$. Este resultado é superior àquele obtido pelo algoritmo A_1 quando aplicado a expressões contendo cláusulas com um único literal.

A Figura IV.16 mostra a dependência entre os algoritmos A_1 e A_2 quando aplicados a expressões com cláusulas contendo no mínimo *k* literais.

k	A1	A2
1	0,5	1,0
2	0,75	0,75
3	0,875	0,704
4	0,938	0,684
5	0,969	0,672

Figura IV.16: Performance de A1 e A2 como função de k

Note que, apesar da melhoria introduzida, o algoritmo A_2 piora quando o número mínimo de cláusulas é aumentado. Este fato entretanto, sugere a criação de um terceiro algoritmo randômico A_3 baseado simplesmente na junção de A_1 e A_2 respectivamente. Sejam n_1 e n_2, o número total de cláusulas satisfeitas por A_1 e A_2 respectivamente. Tem-se então o seguinte resultado:

Proposição IV.10: $\max\{n_1, n_2\} \geq \frac{3}{4}\sum_{j=1}^{m}\hat{z}_j \geq \frac{3}{4}.x^*(B).$

Prova: Seja *B* uma expressão booleana contendo cláusulas com, no mínimo, *k* literais ($k \geq 1$). Executando-se os algoritmos A_1 e A_2 para a entrada *B* conclui-se que :

$$n_1 = E(x_{A_1}(B)) \geq \alpha_k.x^*(B) \qquad \text{e} \qquad n_2 = E(x_{A_2}(B)) \geq \beta_k.x^*(B)$$

onde $\alpha_k = \left(1-\frac{1}{2^k}\right)$ e $\beta_k = \left(1-\left(1-\frac{1}{k}\right)^k\right)$. Segue então que:

$$n_1 + n_2 \geq (\alpha_k + \beta_k).x^*(B) = \left(2-\frac{1}{2^k}-\left(1-\frac{1}{k}\right)^k\right).x^*(B) \geq \left(2-\left(\frac{1}{2^k}+\frac{1}{e}\right)\right).x^*(B) \geq \left(2-\frac{1}{2}\right).x^*(B)$$

Assim conclui-se que: $2.\max\{n_1, n_2\} \geq n_1 + n_2 \geq (3/2).x^*(B)$ e o resultado segue imediatamente. •

Observe que a execução conjunta de A_1 e A_2 respectivamente definem um algoritmo randômico 0,75-aproximado. Isto ocorre mesmo para cláusulas contendo apenas um único literal!

O problema *MAX-SAT* e o problema do corte máximo em grafos, vistos acima, sempre retornam soluções viáveis, independentemente do resultado gerado pelo algoritmo randômico. Pode ocorrer, entretanto, que as soluções obtidas não satisfaçam sempre as restrições de determinado problema sendo, portanto, inviáveis. Exemplos dessa situação podem ser encontrados em Bertsimas& Vohra[1998] e Srinivasan [1995]. Eles apresentam algoritmos randômicos para problemas conhecidos na literatura como *Packing* e *Covering*.

IV.3.3 – *O problema Geral de Recobrimento (General Covering Problem)*

Uma grande quantidade de problemas de Otimização Combinatória pode ser modelada como problemas de Recobrimento, como por exemplo, o problema de Recobrimento de Conjuntos (*Set Covering Problem*), Localização de Facilidades (*Facility Location*), *Network Design* entre outros (Bertsimas&Vohra[1998]).

De maneira geral, o problema de Recobrimento pode ser representado matematicamente por:

$$x^*(I) = Min\ c^T.x$$
$$s.t: \begin{cases} Ax \geq b \\ x \in Z_+^n \end{cases}$$

onde I representa a instância formada pelas matrizes A, b e c sendo $A \in Z_+^{mxn}$, $c, x \in Z_+^{nx1}$ *e* $b \in Z_+^{mx1}$. A relaxação linear deste problema (modelo de Programação Linear-PL) é obtida substituindo-se $x \in Z_+^{nx1}$ por $x \in R_+^{nx1}$.

Nesta seção será discutida a aplicação da técnica de Arredondamento Randômico na resolução do problema de Recobrimento de Conjuntos (*Set Covering Problem*), caso particular do problema geral de Recobrimento. Diferentemente dos problemas MAX-SAT e Corte-Máximo em Grafos onde soluções inteiras geradas randomicamente eram sempre viáveis, no problema de Recobrimento de Conjuntos isto pode não ocorrer. É importante que se tenha, nestes casos, um cuidado especial com a geração indesejada de soluções inviáveis. Como discutido adiante, bastará que soluções viáveis geradas aleatóriamente por este processo tenham probabilidade de sucesso estritamente positiva.

A técnica de Arredondamento Randômico pode ser resumida através do seguinte procedimento genérico:

Procedimento IV.5: Arredondamento Randômico: Prob. Geral de Recobrimento-PRG
Início
1. Resolve Relaxação Linear (PL) obtendo solução $y \in R_+^{nx1}$
2. Gera solução inteira $x \in Z_+^{nx1}$ através de uma função de arredondamento $Pr(x_j = \lfloor y_j \rfloor + 1) = f(y_j)$ e $Pr(x_j = \lfloor y_j \rfloor) = 1 - f(y_j)$, para $j=1,...,n$;
3. Encontra solução determinística *derandomizada*;

fim.

Figura IV.17: Arredondamento Randômico Genérico

A função de arredondamento *f(.)* presente no item 2 pode ser linear (vide *MAX-SAT* seção anterior) ou não-linear. Srinavasan[1995] utiliza arredondamento linear para o PGR fazendo $Pr(x_j = \lfloor y_j' \rfloor + 1) = y_j' - \lfloor y_j' \rfloor$ e $Pr(x_j = \lfloor y_j' \rfloor) = 1 - y_j' + \lfloor y_j' \rfloor$, onde $j=1,...,n$ e $y_j' = \propto y_j$ para $\propto > 0$ escolhido convenientemente. Uma solução *derandomizada* é obtida através do método dos Estimadores Pessimistas. Maiores detalhes sobre a utilização de arredondamento randômico aplicado aos problemas de Recobrimento e Empacotamento (*Covering and Packing Problems*) podem ser encontrados em Srinavasan[1995].

Um arredondamento não-linear será adotado na resolução do problema de Recobrimento de

Conjuntos (Seção IV.3.3.1). A construção probabilística de uma solução determinística (item 3) será discutida mais atentamente na seção IV.4.

IV.3.3.1 – *O problema de Recobrimento de Conjuntos (Set Covering Problem)*

No Problema de Recobrimento de Conjuntos - PRC tem-se um conjunto de n objetos $V=\{1,2,..,n\}$[12] cada um com um custo associado $c_j \geq 0$ (para $j=1,..,n$) e uma família S_i de m subconjuntos de V. O objetivo neste problema será determinar um subconjunto $S \subseteq V$ de maneira que $|S \cap S_i| \geq 1$, $\forall\ i \in \{1,2,...,m\}$ e de maneira que a função objetivo $\sum_{j \in S} c_j$ seja minimizada. Este problema pode ser formalizado através do seguinte modelo de Programação Linear Inteira - PLI:

$$Z^* = Min \sum_{j=1}^{n} c_j x_j$$

$$s.t: \begin{cases} \sum_{j \in S_i} x_j \geq 1, & i = 1,2,...,m \\ x_j \in \{0,1\}, & j = 1,2,...,n \end{cases}$$

Este problema possui inúmeras aplicações práticas importantes, entre elas pode-se destacar a construção de cadeias de DNA. Nesta aplicação, tem-se normalmente uma enorme quantidade de fragmentos de DNA (seqüências de nucleotídeos formadas pelas letras *A, C, G* e *T* respectivamente) e que devem ser utilizadas convenientemente na geração de cadeias maiores (Vazirani [1997]). O objetivo neste caso será facilitar a construção de grandes seqüências através da eliminação de fragmentos redundantes. Em nosso modelo de programação linear inteira, cada fragmento pode ser interpretado como um subconjunto $S_i \subseteq V$ onde V representa todas as possíveis subdivisões destes fragmentos (vértices). Um vértice $v \in V$, poderá pertencer a diversos fragmentos distintos simultaneamente.

Seja $y=(y_1,...,y_n)$ uma solução da relaxação linear (PL) associada ao PRC. Antes de apresentar um arredondamento não-linear para este problema considere o seguinte arredondamento linear onde se tem:

Antes de discutir a qualidade da solução gerada por este procedimento (aproximação de Z^*), deve-se discutir se a solução inteira $x \in \{0,1\}^n$ gerada randomicamente, é de fato viável. Em outras palavras, deseja-se determinar a probabilidade de falha (inviabilidade). Para isso, considere, sem perda de generalidade, um subconjunto $S_i=\{i_1,...,i_k\} \subseteq V$ com k elementos. Além disso, considere um evento E_i que ocorre se, e somente se, o subconjunto S_i não for coberto, ou seja, se nenhum elemento de S_i fizer parte da solução do problema de Recobrimento de Conjuntos. Obviamente, a ocorrência do evento complementar E_i^c irá indicar que algum elemento de S_i foi selecionado. Observe que uma solução inteira (gerada randomicamente) será viável apenas se todos os eventos E_i^c para $i=1,2,..,m$ ocorrerem simultaneamente.

Considere um elemento j pertencente a $S_i=\{i_1,...,i_k\} \subseteq V$ para algum $i \in \{1,..,m\}$. Do arredondamento linear acima tem-se que $Pr(x_j=0)=1-y_j$. Além disso, da solução do problema relaxado (PL) tem-se que $y_{i1}+...+y_{ik} \geq 1$. Assim, fazendo $y_j=1/k$, $\forall\ j \in S_i$ pode-se limitar $Pr(E_i)$ superiormente da seguinte forma:

$$\Pr(E_i) = \prod_{j\in\{i_1,..,i_k\}}(1-y_j) \le \left(1-\frac{1}{k}\right)^k < \frac{1}{e}$$

Um solução inviável é gerada apenas se pelo menos um dos eventos E_i (para $i=1,..,m$) ocorrer. Assim, a probabilidade de falha (inviabilidade) será dada por:

$$\Pr\left(\bigcup_{i=1}^{m} E_j\right) \le \sum_{i=1}^{m} \Pr(E_i) \le \frac{m}{e}$$

Finalmente, conclui-se que a probabilidade de sucesso (solução viável) será igual a:

$$\Pr\left(\bigcap_{i=1}^{m} E_i^c\right) = 1 - \Pr\left(\bigcup_{i=1}^{m} E_i\right) \ge 1 - \sum_{i=1}^{m} \Pr(E_i) \ge 1 - \frac{m}{e}$$

Como a função de probabilidade assume valores no intervalo *[0,1],* pode-se dizer que a probabilidade de sucesso será maior ou igual a *zero (*já que *1-m/e* é negativo para $m\ge3$). Esta situação indesejável pode ser contornada utilizando-se um arredondamento não-linear (Bertsimas& Vohra[1998]) como descrito a seguir:

$$\Pr(x_j = 0) = (1-y_j)^t;$$
$$\Pr(x_j = 1) = 1-(1-y_j)^t; \quad para \quad j = 1,2,...,n \;\; e \;\; t \in Z^+$$

É fácil ver neste caso que $Pr(x_j=0)=(1-y_j)^t$ será menor ao igual ao arredondamento linear onde se tinha $t=1$. Assim, se $1-y_j$ representa a probabilidade de *cara* em uma moeda equilibrada, serão necessárias t repetições (*t caras* seguidas) para atribuição de *zero* à variavel x_j. Caso contrário, se alguma *coroa* ocorrer em k tentativas faz-se $x_j=1$.

Considere novamente $S_i=\{i_1,...,i_k\}$ um subconjunto qualquer de V com k elementos. Analogamente à análise de viabilidade realizada no arredondamento linear tem-se que:

$$\Pr(E_i) = \prod_{j\in\{i_1,..,i_k\}}(1-y_j)^t \qquad onde \;\; i \in \{1,2,..,m\}$$

Como $y_{i1}+...+y_{ik}\ge1$ pode-se fazer $y_j=1/k \;\; \forall\, j \in S_i=\{i_1,...,i_k\}$. Assim: $Pr(E_i) \le (1-1/k)^{kt} \le (1/e)^t$. Se $t=O(logm)$ repetições forem realizadas então: $Pr(E_i) \le (1/e)^{clogm} \le 1/2m$ (verifique). Finalmente, a probabilidade de fracasso (solução inviável) será igual a:

$$\Pr\left(\bigcup_{i=1}^{m} E_i\right) \le \frac{m}{2m} = \frac{1}{2}$$

Portanto, a probabilidade de sucesso (viabilidade) será igual a:

$$\Pr\left(\bigcap_{i=1}^{m} E_i^c\right) = 1 - \Pr\left(\bigcup_{i=1}^{m} E_i\right) \ge \frac{1}{2}$$

Uma vez provada que uma solução viável é gerada com probabilidade de sucesso maior ou igual a 50% deve-se responder agora à seguinte questão: qual a qualidade da solução (aproximação de Z^*) gerada pelo procedimento randômico? Antes de tratar deste assunto considere o seguinte resultado auxiliar conhecido na literatura como desigualdade de Bernoulli:

Proposição IV.11: (Desigualdade de Bernoulli)
Se $x \geq -1$ então $(1+x)^t \geq 1+tx, \ \forall t \in N$.

Prova: Este resultado pode ser provado por indução em t. Para $t=1$ a tese se verifica imediatamente (trivial). Suponha que a tese seja verdadeira para um t inteiro positivo qualquer. Como $x+1 \geq 0$ tem-se para $t+1$ que: $(1+x)^t(1+x) \geq (1+tx).(1+x) = 1 + tx + x + tx^2 = 1 + (t+1)x + tx^2 \geq 1 + (t+1)x$. •

Seja A, um algoritmo randômico para o PRC que utiliza o arredondamento não-linear descrito acima. Além disso, seja $x \in \{0,1\}^n$ sua solução e $x_A(I)$ sua imagem associada. Assim, como $x_A(I) = c_1x_1+...+c_nx_n$ (função objetivo), segue, da linearidade do valor esperado que $E(x_A(I)) = c_1E(x_1)+...+c_nE(x_n)$. Mas $E(x_j) = Pr(x_j=1)=1-(1-y_j)^t$ onde $y_j \in [0,1]$ (note neste caso que $x=-y_j \geq -1$). Da desigualdade de Bernoulli (Proposição IV.11) tem-se que: *1-(unç2 102O664(unç2Tc(p02.80.3())-8(8)-11)-10.6(,1)2(8)-1*

Yannakakis[1994] provaram que isto só será possível se $P=NP$! Apesar disso, como discutido a seguir, pode-se conseguir para instâncias particulares do PRC, uma maior aproximação do valor ótimo Z^*.

Observe que o evento $F = E_1^c \cap ... \cap E_m^c$ representa uma solução viável e ocorre com probabilidade estritamente positiva ($Pr(F) > 0$). Na verdade, apenas uma iteração do procedimento IV.6 já garante $Pr(F)>1/2$. Isto foi possível fazendo-se $t=O(logm)$ no arredondamento não-linear.

Como discutido em Bertsimas&Vohra[1998], um outro arredondamento por ser obtido fazendo-se $x_j=1$, se $y_j \geq 1/logm$. Caso contrário, se $y_j<1/logm$ faz-se $x_j=1$ com probabilidade $y_j/logm$. Analogamente ao arredondamento não-linear aqui apresentado este procedimento é também $O(logm)$-aproximado.

Considere agora o seguinte problema: ao "enfraquecer" a probabilidade de sucesso tornando-a menor, mas estritamente positiva, será possível uma maior aproximação da solução ótima Z^*? Neste caso, deseja-se calcular $E(x_A(I)|F)$ garantindo apenas que $Pr(F) >0$. A Proposição IV.13 provada por Bertsimas&Vohra[1998] resolve esta questão. Antes de tratá-la entretanto, considere a definição e proposição preliminares:

Definição IV.10: (Família monótona crescente)

Uma família Ψ de subconjuntos de um conjunto N qualquer será *monótona crescente* se, e somente se, $S \in \Psi$ implicar que $T \in \Psi$, $\forall\ S \subseteq T \subseteq N$. A definição de família *monótona decrescente* é análoga e deixada como exercício. •

A Proposição seguinte é caso particular de um importante resultado conhecido na literatura como desigualdade FKG (*FKG inequality*). Para maiores detalhes vide Fortuin et al.[1971].

Proposição IV.12: (Correlação Positiva)

Considere um conjunto finito $N=\{a_1,...,a_l\}$ e um vetor $p=(p_1,...,p_l) \in [0,1]^l$. Suponha que um subconjunto $Y \subseteq N$ seja obtido aleatoriamente onde cada elemento a_k de Y seja selecionado independentemente com probabilidade p_k. Para cada $\Psi \subseteq 2^N$ (onde 2^N é o conjunto de todas as partes de N), seja $Pr_p(\Psi)=Pr(Y \in \Psi)$. Considere ainda $F_1,F_2,...,F_s \subseteq 2^N$ uma seqüência qualquer de famílias monótonas crescentes . Então:

$$\Pr_p\left(\bigcap_{k=1}^{s} F_k\right) \geq \prod_{k=1}^{s} \Pr_p(F_k)$$

Diz-se, neste caso, que as famílias $F_1,F_2,...,F_s$ estão *correlacionadas positivamente.*

Prova: (Teorema 3.2 do Capítulo 6 de Alon&Spencer[1992]). •

Pode-se aplicar este resultado ao PRC da seguinte forma. Note que, se $l=n$, o conjunto N pode estar associado a uma restrição genérica do PRC. Assim:

$$N = \{a_1,...,a_n\} \quad \leftrightarrow \quad \sum_{j=1}^{n} a_j x_j \geq 1$$

Ainda, o subconjunto $Y \subseteq N$ pode estar associado a uma solução randômica $x \in \{0,1\}^n$ do PRC onde $y \in [0,1]^n$ (solução relaxada do PRC) representa o vetor de probabilidades p. A família $\Psi \subseteq 2^N$ estará associada ao conjunto de todas as soluções 0-1 satisfazendo uma restrição particular do PRC. Em outras palavras, cada subconjunto $S \in \Psi$ estará associado a uma solução $x \in \{0,1\}^n$ satisfazendo a restrição correspondente. O valor $Pr_p(\Psi)$ representa a probabilidade da solução x (gerada randômicamente) satisfazer a "restrição" Ψ. Note que cada uma das famílias (restrições) associadas ao PRC é monótona crescente pois $a_j=0$ ou 1, $\forall\ j \in \{1,2,...,n\}$. A família associada à restrição $x_1+x_2-x_3 \geq 1$

por exemplo, não é monótona crescente (verifique).

Como discutido anteriormente, se um evento E_k ocorre então o subconjunto $S_k \subseteq V$ não foi coberto. Pode-se mostrar portanto, que os eventos complementares $E_1^c,...,E_m^c$, associados a $S_1,...,S_m$ respectivamente estarão correlacionados positivamente. Assim, com o auxílio da Proposição IV.12 a probabilidade de sucesso pode ser limitada inferiormente da seguinte forma:

$$\Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \geq \prod_{i=1}^{m} \Pr(E_i^c)$$

O exemplo seguinte ilustra bem esta correlação positiva entre eventos:

Exemplo IV.4: (Correlação positiva entre eventos)

Seja $V=\{1,2,3,4\}$ um conjunto de vértices associado ao PRC com subconjuntos $S_a=\{1,2,3\}$ e $S_b=\{3,4\}$ respectivamente. Ao modelar este problema matematicamente obtêm-se o seguinte conjunto de restrições:

$$\begin{cases} x_1 + x_2 + x_3 \geq 1 \\ x_3 + x_4 \geq 1 \end{cases}$$
$$onde \;\; x_j \in \{0,1\} \quad para \;\; j = 1,2,3,4$$

Seja $x \in \{0,1\}^4$ uma solução gerada randômicamente. É fácil ver que, se a primeira restrição for satisfeita (S_a for coberto) a probabilidade da segunda restrição ser satisfeita será maior, ou seja, $Pr(E_b^c) \leq Pr(E_b^c \mid E_a^c)$. Assim, $Pr(E_a^c \cap E_b^c) = Pr(E_a^c).Pr(E_b^c \mid E_a^c) \geq Pr(E_a^c).Pr(E_b^c)$. •

Novamente, considere F um evento indicando que todas as restrições associadas ao PRC foram satisfeitas (uma solução viável foi obtida). Além disso, considere $x_A(I)$ uma solução gerada no arredondamento não-linear (algoritmo A), e Z^* seu valor ótimo associado. Tem-se então o seguinte resultado:

Proposição IV.13: (Bertsimas&Vohra[1998])

Seja $\Delta_j=\{i \in \{1,..,m\} \mid j \in S_i\}$ e $\Delta=max\{|\Delta_j| \text{ p/ } j=1,2,...,n\}$. Então $E(x_A(I)|F) \leq O(log\Delta).Z^*$.

Prova: Da linearidade do valor esperado e da definição de valor esperado condicional (vide Capítulo II) tem-se que:

$$E(x^*(I)) = E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) = \sum_{j=1}^{n} c_j E\left(x_j \mid F\right) = \sum_{j=1}^{n} c_j . \Pr(x_j = 1 \mid F) \qquad (I)$$

Considere agora $B_j=\{x \in \{0,1\}^n | x_j=1\}$ e $B_j^c=\{x \in \{0,1\}^n \mid x_j=0\}$ (para algum $j \in \{1,...,n\}$) *dois* eventos complementares[13]. Da fórmula de Bayes (Capítulo II) conclui-se que:

$$\Pr\left(x_j = 1 \mid F\right) = \frac{\Pr(B_j \cap F)}{\Pr(F)} = \frac{\Pr(x_j = 1).\Pr(F \mid x_j = 1)}{\Pr(F)} \qquad (II)$$

Como definido anteriormente, o evento complementar E_i^c ocorre (para algum $i \in \{1,..,m\}$) se pelo menos um elemento de S_i for selecionado (S_i é coberto). É fácil ver então que:

[13] Note que, $B_j \cap B_j^c = \varnothing$ e $B_j \cup B_j^c = \{0,1\}^n$.

$$\Pr(F) = \Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \qquad e \qquad \Pr(F \mid x_j = 1) = \Pr\left(\bigcap_{i \notin \Delta_j} E_i^c\right) \qquad (III)$$

Observe, da segunda igualdade que, como $x_j=1$, todos os subconjuntos S_i contendo o vértice j de V já foram cobertos. Além disso, como os eventos E_i^c para $i=1,..,m$ são correlacionados positivamente então:

$$\Pr(F) = \Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \geq \Pr\left(\bigcap_{i \notin \Delta_j} E_i^c\right).\Pr\left(\bigcap_{i \in \Delta_j} E_i^c\right) \qquad (IV)$$

De (III) e (IV) tem-se que:

$$\frac{\Pr(F \mid x_j = 1)}{\Pr(F)} = \frac{\Pr\left(\bigcap_{i \notin \Delta_j} E_i^c\right)}{\Pr\left(\bigcap_{i=1}^{m} E_i^c\right)} \leq \frac{1}{\Pr\left(\bigcap_{i \in \Delta_j} E_i^c\right)} \leq \frac{1}{\prod_{i \in \Delta_j} \Pr(E_i^c)}$$

Lembre-se, do arredondamento não-linear que: $Pr(x_j=0)=(1-y_j)^t$ para $j=1,..,n$. Logo, um evento E_i ocorre (p/ algum $i \in \{1,..,m\}$), se nenhum dos k elementos de $S_i=\{i_1,...,i_k\}$ for selecionado. Como $y_{i1}+...+y_{ik} \geq 1$, é fácil ver então que:

$$\Pr(E_i^c) = 1 - \Pr(E_i) = 1 - \prod_{j \in \{i_1,..,i_k\}} (1 - y_j)^t \geq 1 - \left((1 - 1/k)^k\right)^t > 1 - e^{-t}$$

Assim:

$$\prod_{i \in \Delta_j} \Pr(E_i^c) \geq \prod_{i \in \Delta_j} (1 - e^{-t}) = \left(1 - e^{-t}\right)^{|\Delta_j|} \geq \left(1 - e^{-t}\right)^{\Delta}$$

onde $\Delta = max\ \{|\Delta_j| \text{ para } j=1,..n\}$.
Portanto:

$$\frac{\Pr(F \mid x_j = 1)}{\Pr(F)} \leq \frac{1}{(1 - e^{-t})^{\Delta}} \qquad (V)$$

Após a substituição de (V) em (II), e com o auxílio da desigualdade de Bernoulli tem-se que:

$$\Pr(x_j = 1 \mid F) = \frac{\Pr(F \mid x_j = 1)}{\Pr(F)}.\Pr(x_j = 1) \leq \left(1 - e^{-t}\right)^{-\Delta}.\left(1 - (1 - y_j)^t\right) \leq \left(1 - e^{-t}\right)^{-\Delta}.ty_j \qquad (VI)$$

Finalmente, substituindo (VI) em (I):

$$E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) \leq \sum_{j=1}^{n} c_j \left(1 - e^{-t}\right)^{-\Delta}.ty_j \leq t.\left(1 - e^{-t}\right)^{-\Delta} \sum_{j=1}^{n} c_j y_j$$

Escolhendo $t=ln\Delta$ e assumindo $\Delta \geq 2$ segue que:

$$E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) \leq \ln n.\left(\frac{1}{(1 - e^{-\ln \Delta})^{\Delta}}\right).Z^* = \ln n.\left(\frac{1}{(1 - 1/\Delta)^{\Delta}}\right).Z^* \leq 4.O(\ln \Delta).Z^*$$

Ou ainda:

$$E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) \leq O(\log \Delta).Z^*$$

completando a prova. •

Observe que, ao reduzir o número de repetições *t* de *O(logm)* para *O(logΔ)* no arredondamento não-linear consegue-se uma solução O(logΔ)-aproximada para o PRC. Isto representa um ganho de qualidade se $\Delta<m$. Neste caso, entretanto, a probabilidade de sucesso poderá ser menor, embora estritamente positiva. Para constatar esse fato note, da Proposição IV.13 acima que: $Pr(E_i^c)>1-e^{-t}$, para $i \in \{1,..,m\}$. Tem-se então da correlação positiva entre eventos:

$$\Pr(F) = \Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \geq \prod_{i=1}^{m} \Pr(E_i^c) = \left(1 - \frac{1}{e^t}\right)^m > 0$$

Observe que a probabilidade *Pr(F)* será pequena, mas estritamente positiva, se $t=\log\Delta$ e o número de subconjuntos *m* for bastante grande. Essa probabilidade de sucesso (estritamente positiva) será fundamental no processo de *derandomização* discutido com mais detalhes na Seção IV.3.4.2.

IV.3.4 - Construção Probabilística de Algoritmos Determinísticos (Derandomization)

Em algumas situações, será possível construir algoritmos determinísticos, com o auxílio de técnicas probabilísticas. Em outras palavras, deseja-se construir um algoritmo determinístico sem que se sacrifique muito da qualidade da solução e/ou o tempo de processamento obtidos no procedimento randômico. Infelizmente, não se conhece um mecanismo universal de conversão que seja aplicável a todas as situações.

Apesar de não existir um termo apropriado em português para esta técnica, talvez o mais conveniente seja chama-la simplesmente de *derandomização* (semelhante ao temo inglês *derandomization*). Entre os métodos de derandomização mais conhecidos na literatura pode-se citar: o método das expectâncias condicionais, o método dos estimadores pessimistas e *k-wise independence* (para maiores detalhes sobre o método dos estimadores pessimistas consulte Raghavan&Thompson[1988]). Nesta seção será dada uma atenção especial ao método das expectâncias condicionais.

Suponha que uma variável aleatória *X* assuma valores $x_1,x_2,...$ (finitos ou não). Além disso, considere Y^c o evento complementar associado a um evento *Y* qualquer. No método das expectâncias condicionais utiliza-se freqüentemente a seguinte definição de valor esperado condicional (vide Capítulo II):

$$E(X \mid Y) = \sum_i x_i . \Pr(X = x_i \mid Y)$$

Além disso:

$$E(X) = E(X \mid Y).\Pr(Y) + E(X \mid Y^c).\Pr(Y^c) \qquad (4)$$

Note a partir da expressão acima que $E(X) \leq (Pr(Y)+(1-Pr(Y))).max\{E(X|Y), E(X|Y^c)\} = max\{E(X|Y), E(X|Y^c)\}$. Analogamente, pode-se concluir que que: $min\{E(X|Y), E(X|Y^c)\} \leq E(X)$. Esta abordagem é interessante já que nos permite definir um mecanismo para tomada de decisões sempre limitando inferiormente ou superiormente (conforme o caso) o valor esperado *E(X)*. Nas seções seguintes está técnica será aplicada aos problemas *MAX-SAT* e Recobrimento de conjuntos (*Covering Problem*).

IV.3.4.1 – O Problema MAX-SAT

Considere novamente o problema *MAX-SAT* discutido na Seção IV.3.2. Deseja-se, a partir da solução randômica 0,5-aproximada gerada pelo Algoritmo A_1, determinar uma solução determinística também 0,5-aproximada. O raciocínio aplicado a esta situação pode ser facilmente estendido aos procedimentos A_2 e A_3 respectivamente.

Como A_1 retorna uma solução randômica 0,5-aproximada (vide Seção IV.3.2) tem-se que:

$$0{,}5.x^*(B) \le E\left(x_{A1}(B)\right) = E\left(\sum_{j=1}^{m} Z_j\right)$$

onde Z_j é variável aleatória 0-1 associada à cláusula C_j da expressão B na forma normal conjuntiva. Deve-se, inicialmente, decidir se a variável x_1 (associada a C_1) é *falsa* ou *verdadeira* (V ou F). Seja Y um evento indicando que $x_1=V$, e Y^c indicando que $x_1=F$. Tem-se de (4) que:

$$E\left(\sum_{j=1}^{m} Z_j\right) = E\left(\sum_{j=1}^{m} Z_j \mid x_1 = V\right).\Pr(x_1 = V) + E\left(\sum_{j=1}^{m} Z_j \mid x_1 = F\right).\Pr(x_1 = F)$$

Ou seja:

$$E(x_{A1}(B)) \le \max\left\{E\left(x_{A1}(B) \mid x_1 = V\right), E\left(x_{A1}(B) \mid x_1 = F\right)\right\}$$

Considere $E(x_{A1}(B)|X_1)) = max\{E(x_{A1}(B)|x_1=V), E(x_{A1}(B)|x_2=F)\}$. Repetindo-se o processo para o literal x_2 conclui-se que:

$$E(x_{A1}(B)) \le max\{E(x_{A1}(B)|X_1, x_2 =V), E(x_{A1}(B)| X_1, x_2=F)\}.$$

Assim, após $n-1$ passos, chega-se a:

$$E(x_{A1}(B)) \le max\{E(x_{A1}(B)|X_1,...,X_{n-1}, x_n =V), E(x_{A1}(B)| X_1,...,X_{n-1}, x_n =F)\} = E(x_{A1}(B)|X_1,...,X_n)$$

Provou-se finalmente que: $0{,}5.x^*(B) \le E(x_{A1}(B)) \le E(x_{A1}(B)|X_1,...,X_n)$. O algoritmo seguinte, sintetiza todas as etapas realizadas:

Procedimento IV.7: MAX-SAT Derandomizado
Início
  Para $k=1$ **até** n **faça**
    Calcula:
      $R \leftarrow E(x_{A1}(B)|X_1,...,X_{k-1}, x_k=V);$
      $S \leftarrow E(x_{A1}(B)|X_1,..., X_{k-1}, x_k=F);$
    Se $R \ge S$ **então**
      $x_k=V$
    senão
      $x_k=F$
  fim.
  Retorna $x_{A1}(B)$ e solução $(X_1,..., X_n);$
fim.

Figura IV.17: MAX-SAT *Derandomizado*

Observe acima que, a cada iteração do procedimento, os valores R e S (ou seja,

$E(x_{A1}(B)|X_1,...,X_{k-1},x_k=V \text{ ou } F)$) são calculados. O exemplo seguinte ilustra o cálculo dos valores R e S respectivamente:

Exemplo IV.5: Considere a seguinte expressão booleana com 3 literais e 5 cláusulas:

$$B = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_3)$$

Do procedimento MAX-SAT acima (Figura IV.17) deve-se calcular inicialmente $E(x_{A1}(B)|\ x_k=V))$ e $E(x_{A1}(B)|\ x_k=F))$ respectivamente. Para faze-lo basta notar que:

$$E(x_{A1}(B) \mid x_1 = V \ \textit{ou}\ F) = E\left(\sum_{j=1}^{m} Z_j \mid x_1 = V \ \textit{ou}\ F\right) = \sum_{j=1}^{m} E(Z_j \mid x_1 = V \ \textit{ou}\ F) = \sum_{j=1}^{m} 1.\Pr(Z_j = 1 \mid x_1 = V \ \textit{ou}\ F)$$

Assim, de nosso exemplo:

$$E(x_{A1}(B) \mid x_1 = V) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid x_1 = V) = 1 + (1/2) + (3/4) + 1 + (1/2) = 15/4 \cong 3{,}75$$

Analogamente, para $x_1=F$ tem-se:

$$E(x_{A1}(B) \mid x_1 = F) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid x_1 = F) = (1/2) + 1 + 1 + (3/4) + (1/2) = 15/4 \cong 3{,}75$$

Como $E(x_{A1}(B) \mid x_1=V\ ) = E(x_{A1}(B) \mid x_1=F\ )$, considere, sem perda de generalidade que $X_1=V$. Assim:

$$E(x_{A1}(B) \mid X_1, x_2 = V) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, x_2 = V) = 1 + 1 + (1/2) + 1 + (1/2) = 4$$

$$E(x_{A1}(B) \mid X_1, x_2 = F) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, x_2 = F) = 1 + 0 + 1 + 1 + (1/2) = 3{,}5$$

Logo, $X_2=V$. Finalmente, para o literal x_3 tem-se:

$$E(x_{A1}(B) \mid X_1, X_2, x_3 = V) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, X_2, x_3 = V) = 1 + 1 + 0 + 1 + 1 = 4$$

$$E(x_{A1}(B) \mid X_1, X_2, x_3 = F) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, X_2, x_3 = F) = 1 + 1 + 1 + 1 + 0 = 4$$

Note portanto que $E(x_{A1}(B)|X_1, X_2, X_3, X_4) = 4$, onde $X_1=V$, $X_2=V$ e $X_3=V$ ou F representa uma solução 0,5-aproximada. •

Como discutido a seguir, utilizando-se um raciocínio análogo será possível construir uma solução *derandomizada* para o problema de Geral de Recobrimento (*General Covering Problem*), em especial, para o problema de Recobrimento de Conjuntos (*Set Covering Problem*).

IV.3.4.2 – O Problema de Recobrimento de Conjuntos (*Set Covering Problem*)

Analogamente à seção anterior, o objetivo aqui será gerar uma solução determinística para o PRC utilizando-se técnicas e ferramentas probabilísticas. Neste caso, entretanto, a solução obtida será *O(logm)*-aproximada sendo portanto um pouco inferior àquela apresentada na Proposição IV.13.

Diferentemente do problema MAX-SAT (onde uma solução viável era obtida de maneira trivial) o objetivo aqui será gerar uma solução determinística preocupando-se obviamente em satisfazer todas as restrições. Este problema é resolvido com o auxílio da seguinte função potencial:

$$\Phi(x) = \Phi(x_1,..., x_n) = \sum_{j=1}^{n} c_j x_j + M \sum_{i=1}^{m} Y_i$$

onde:

$$Y_i = \begin{cases} 1, & \sum_{j \in S_i} x_j = 0 \\ 0, & \textit{caso contrário} \end{cases}$$

Observe que uma solução $x \in \{0,1\}^n$ será viável se, e somente se, $\Phi(x) \leq M$ para M escolhido convenientemente.Caso contrário, se $x \neq 0$ e se algum subconjunto S_i de V não for coberto, então $Y_i=1$ e $\Phi(x)>M$ (solução inviável). O valor atribuído à constante M será discutido mais adiante.

Da linearidade do valor esperado tem-se que:

$$E\big(\Phi(x)\big) = \sum_{j=1}^{n} c_j E(x_j) + M \sum_{i=1}^{m} E(Y_i)$$

onde:

$$\sum_{i=1}^{m} E(Y_i) = \sum_{i=1}^{m} 1.\Pr(Y_i = 1)$$

Sem perda de generalidade considere que apenas uma das restrições seja violada (para algum $i \in \{1,..,m\}$. Assim:

$$\Pr(Y_i = 1) = \prod_{j \in S_i} (1 - y_j)^t \leq \left(\left(1 - \frac{1}{|S_i|} \right)^{|S_i|} \right)^t \leq e^{-t}$$

Como $E(x_j)=Pr(x_j=1)=1-(1-y_j)^t \leq t.y_j$ (no arredondamento não-linear) e $Pr(F)>(1-e^{-t})^m$ (probabilidade de sucesso) então:

$$E\big(\Phi(x)\big) \leq t.Z_{RL} + M\big(1 - (1 - e^{-t})^m\big)$$

onde $Z_{RL}=\Sigma_j c_j y_j$ representa o valor ótimo associado a uma solução $y \in [0,1]^n$ (relaxação linear).

Escolhendo-se M convenientemente, a questão que se coloca agora é: como determinar uma solução determinística (aqui representada por X) de maneira que $E(\Phi(X)) \leq M$? Do método das expectâncias condicionais tem-se que:

$$\begin{aligned} E\big(\Phi(X)\big) &= E\big(\Phi(X) \mid x_1 = 1\big).\Pr(x_1 = 1) + E\big(\Phi(X) \mid x_1 = 0\big).\Pr(x_1 = 0) \\ &\geq (\Pr(x_1 = 1) + (1 - \Pr(x_1 = 1))).\min\big\{E\big(\Phi(X) \mid x_1 = 1\big),\ E\big(\Phi(X) \mid x_1 = 0\big)\big\} = E\big(\Phi(X) \mid X_1\big) \\ &\geq \min\big\{E\big(\Phi(X) \mid X_1, x_2 = 1\big),\ E\big(\Phi(X) \mid X_1, x_2 = 0\big)\big\} = E\big(\Phi(X) \mid X_1, X_2\big) \end{aligned}$$

Repetindo-se o processo, após $n-1$ passos chega-se a:

$$E(\Phi(X)) \geq \min\big\{E\big(\Phi(X) \mid X_{1,},..., X_{n-1}, x_n = 1\big),\ E\big(\Phi(X) \mid X_{1,},..., X_{n-1}, x_n = 0\big)\big\} = E\big(\Phi(X) \mid X_{1,},..., X_n\big)$$

Portanto:

$$E\big(\Phi(X) \mid X_{1,},..., X_n\big) \leq E\big(\Phi(X)\big) \leq t.Z_{RL} + M(1 - (1 - e^{-t})^m \leq M$$

A constante M deve ser escolhida de maneira que o seguinte problema de otimização seja resolvido:

$$\begin{aligned} & \min M \\ s.t. \quad & t.Z_{RL} + M\left(1-\left(1-e^{-t}\right)^m\right) \le M \\ & t, M \ge 0 \end{aligned}$$

Note que, escolhendo:

$$M = \frac{\log m}{(1-1/m)^m}.Z_{RL} \qquad e \qquad t = \log m$$

tem-se uma solução X gerada determinísticamente onde:

$$\sum_{j=1}^{n} c_j X_j \le \Phi(X) \le \frac{\log m}{(1-1/m)^m}.Z_{RL} = 4.\log m.Z_{Rl} = O(\log m).Z_{RL}$$

Observe que: $(1\text{-}1/m)^m \ge 1/4, \ \forall\, m \ge 2$.

Note que, após a substituição de M é t nas restrições do problema de programação linear tem-se:

$$\log m.Z_{RL} + \frac{\log m}{(1-1/m)^m}.Z_{RL}.\left(1-\left(1-1/e^{\log m}\right)\right) \le \log m.(1-1/m)^m.Z_{RL} \Rightarrow$$

$$1 + \frac{1}{(1-1/m)^m} - \left(\frac{(1-1/e^{\log m})^m}{(1-1/m)^m}\right) \le \frac{1}{(1-1/m)^m} \Rightarrow (1-1/m)^m \le (1-1/e^{\log m})^m \Rightarrow$$

$$(1-1/e^{\ln m})^m \le (1-1/e^{\log m})^m.$$

Note que a última desigualdade é verdadeira pois, $lnm = logm/loge$, ou seja, como $loge>1$ então $lnm<logm$.

[01] A. V. Aho, J. E. Hopcroft and J. D. Ullman [1974] "*The Design and Analysis of Computer Algorithms",* Addison-Wesley, Readings.
[02] R. K. Ahuja, T. L. Magnanti, J. B. Orlin [1993] "*Network Flows: Theory, Algorithms and Applications*", Prentice-Hall, Inc.
[03] F. Alizadeh, L. Shi [1997] "*Randomized Complexity Classes: Lecture 3*", Lecture Notes.
[04] N. Alon, J. Spencer [1992] "*The Probabilistic Method*", Wiley, New York.
[05] G. Andersson [2000] "*Some New Randomized Approximation Algorithms",* Doctoral Dissertation, Royal Institute of Technology - Stockholm / Swedan, Department of Numerical Analysis and Computer Science.
[06] G. Arantes, F. França, C. Martinhon [2002] "*Algoritmos Randômicos para Geração de Orientações Acíclicas em Sistemas Distribuídos*", Trabalho completo aceito para publicação nos Anais do XXXIV SBPO, Rio de Janeiro, RJ.
[07] S. Arora [1996] "*Polynomial Time Approximation Schemes for Euclidean TSP and other Geometric Problems*", Technical Report, Princeton University.
[08] G. Ausiello, P. Crescenzi, M. Protasi [1995] "*Approximate solution of NP optimization problems*", Theoretical Computer Science 150, 1-55.
[09] F. Barahona, M. Grötchel, M. Jünger, G. Reinelt [1988] "*An application of combinatorial optimization to statistical physics and circuit layout design*", Operations Research, 36: 493-513.
[10] F. Barahona [1996] "*Network design using cut inequalities*", SIAM Journal on optimization, to appear.
[11] M. Bazaraa, J. Jarvis, H. Sherali [1990] "*Linear Programming and Network Flows",* John Wiley&Sons, second Edition.
[12] A. Berthiaume [1996] "*Quantum Computation*", Complexity Theory Retrospective II, Springer-Verlag.
[13] Bertisimas and Voha [1998] "*Rounding Algorithms for Covering Problems*", Math. Programming, v.80 pp. 63-89.
[14] N. Blum [1990] "*A new approach to maximum matching in general graphs*", Proc. 17th ICALP LNCS 443, 586-597.
[15] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, R.E. Tarjan [1973] "*Time bounds for selection*", Journal of Computer and System Sciences, 7:448-461.
[16] J. Boldrini, S. Costa, V. Figueiredo, H. Wetzler [1984] "*Álgebra Linear",* Ed. Harbra.
[17] R.B. Boppana, R. Hirschfeld [1989] "*Pseudo-random generators and complexity classes*", In S. Micali, editor, *Randomnes and Computing* (*Advances in Computing Research*), Volume 5, pages 1-26. JAI Press, Greenwich, CT.
[18] G. Brassard, P. Bratley, [1988] "*Algorithms: Theory and Practice*", Prentice-Hall.
[19] A. Calabrese, F. França [1994] "*Randomized Distributed Primer for the Updating Control of Anonymous ANNs*". Proceedings of ICANNR94, Sorrento, Italy.
[20] A. Calabrese [1997] "Distributed Acyclic Orientation of Asynchronous Networks", 11th International Symposium of Fundamentals of Computing Theory", pp. 129-137, Krakov, Poland.
[21] R. Campello e N. Maculan, [1994] *"Algoritmos e Heurísticas, Avaliação e Análise de Performance",* Editora UFF , RJ.
[22] A. L. Chistov [1985] "Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic", In Proceedings of the International Conference on the Foundations of Computation Theory, Springer-Verlag Lecture Notes in Computer Science, 199, pages 63-69.
[23] D. Coppersmith , S. Winograd [1987] "*Matrix multiplication via arithmetic progressions*", Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pages 1-6.

[24] T. H. Cormem, C. E. Leiserson and R. L. Rivet, [1990] *"Introduction to Algorithms"*, MIT Press, Cambridge, Ma and McGraw-Hill, New York, NY.

[25] W.J. Cook, W.H. Cunninghan, W.R. Pulleyblank, A Schrijver [1998] "*Combinatorial Optimization*", Wiley-Interscience Series in Discrete Mathematics and Optimization.

[26] N. Christofides [1976] "*Worst-case analysis of a new heuristic for the travelling salesman problem*", Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

[27] M. Dell'Amico, F. Maffioli, S. Martello [1997] "*Annotated Bibliographies in Combinatorial Optimization*", John Wiley&Sons.

[28] M.D. Donsker, M. Kac [1949] "*The Monte Carlo Method and Its Applications*", Proceedings, Seminar on Scientific Computation, International Business Machines Corporation 1950, pp. 74-81.

[29] P. Erdös, J. Spencer [1974] "*The Probabilistic Method in Combinatorics"*, Academic Press, San Diego.

[30] J. Edmonds [1967] "*systems of distintic representatives and linear algebra*" Journal of Research of the National Bureau of Standards, 71B, 4:241-245.

[31] C.E. Ferreira et al. [2001] "*Uma Introdução Sucinta a Algoritmos de Aproximação*", 23° Colóquio Brasileiro de Matemática, Editores: C.Fernandes, F. Miyazawa, M. Ceriolli, P. Feofiloff.

[32] R.W. Floyd, R.L. Rivest [1975] "*Expected time bounds for selection*", Communications of the ACM, 18: 165-172,1975.

[33] C.M.Fortuin, J. Ginibre, P.N. Kasteleyn [1971] "*Correlational inequalities for partially ordered sets*", Communications of Mathematical Physics, 22:89-103.

[34] R.W. Floyd, R.L. Rivest [1975] "*Expected time bounds for selection*", Communications of the ACM, 18:165-172.

[35] L.R. Ford, D.R. Fulkerson [1956] "*Maximal flow through a network*", Canadian Journal of Mathematics, 8:399-404.

[36] R. Freivalds [1979] "*Fast probabilistic algorithms*", Vol. 74 of Lecture Notes in Computer Science, pages 57-59. Springer-Verlag.

[37] P. Gács, L. Lovász [1999] "*Complexity of Algorithms*" , Lecture Notes, Spring 1999.

[38] M.R. Garey, R.L. Graham, J.D. Ullman [1972] "*Worst case analysis of memory allocation algorithms*", in Proc. of the 4th ACM Symp. on Theory of Computing. 143-150.

[39] M. R. Garey, D. Johnson [1979] *"Computers and Intractability: A Guide to the Theory of NP-Completeness",* W. H. Freeman, San Francisco, CA.

[40] M. Gendreau, G. Laporte, A. Hertz [1997] "*An Approximation Algorithm for the Traveling Salesman Problem with Backhauls*", Technical Note, Operations Research, Vol. 45 No. 4.

[41] A. Gibbons, [1985] "*Algorithmic Graph Theory*", Cambridge University Press.

[42] J. Gill [1977] "*Computational complexity of probabilistic Turing machines*", SIAM J. Comput. 6, 675-695.

[43] M.X. Goemans, D.P. Williamson [1994] "*0.878-approximation algorithms for MAX-CUT and MAX-2SAT*", in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pages 422-431.

[44] M.X. Goemans, D.P. Williamson [1995] "*Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming*", In J. ACM., 42, 1115-1145, 1995.

[45] M.C. Goldbarg, H.P.L. Luna [2000] "*Otimização Combinatória e Programação Linear: Modelos e Algoritmos*", Editora Campus.

[46] G.H. Golub, C.F. Loan [1996] "*Matrix Computations*", The Johns Hopkins University Press, 3rd edition.

[47] R.L. Graham [1966] "*Bounds for certain multiprocessing anomalies*", Bell System Tech. J. 45, 1563-1581.

[48] R. Gupta, S.A. Smolka, S. Bhaskar [1994] "*On randomization in sequential and distributed algorithms*", ACM Computing Surveys, Vol 26, n.1.

[49] F. Hadlock [1975] "*Finding a maximum cut of a planar graph in polynomial time*", SIAM Journal on Computing, 4:221-225.

[50] C.A.R. Hoare [1962] *"Quicksort",* Computer Journal, 5(1):10-15.
[51] D.S Hochbaum [1997] "*Approximation algorithms for NP-hard problems*", PWS Publishing Company.
[52] D.S Hochbaum, D. B. Shmoys [1987] "*Using dual approximation algorithms for scheduling problems: practical and theoretical results*", Journal of ACM 34: 144-162.
[53] J.E. Hopcroft, J.D. Ullman [1979] "*Introduction to automata theory, languages and computation*"; Addison-Wesley Publishing Company.
[54] E. Horowitz, S. Sahni, [1978] "*Fundamentals of Computer Algorithms*", Computer Science Press, Inc.
[55] A.S. Householder, G.E. Forsythe, H.H. Germond Eds [1951] "Monte Carlo Methods", NBS Applied Mathematics Series, Vol. 12, 6.
[56] B.R. James [1981] "*Probabilidade: um curso em nível intermediário*", Projeto Euclides.
[57] D.S. Johnson [1974] "*Approximation algorithms for combinatorial problems*", J. Comput. System Sci., 9, 256-278.
[58] G.M.A Júnior, F.M.G. França [2001] "*Geração Quase-Instantânea de Orientações Acíclicas em Sistemas Distribuídos* " Relatório Técnico – Eng. de Sistemas COPPE/UFRJ.
[59] M.H. Kalos, P.A. Whitlock [1986] "*Monte Carlo Methods: Volume I - Basics*", John Wiley & Sons.
[60] D.R. Karger [1993] *"Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm*", In Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 21-30.
[61] D. R. Karger [1995] "*Random sampling in graph optimization problems*", PhD Dissertation - Department of Graduate Studies of Stanford University.
[62] D.R. Karger, C. Stein [1993] "*An $\tilde{O}(n^2)$ algorithm for minimum cuts*", In Proceedings of the 25th Annual ACM Symposium on Theory of Computing, pages 757-765.
[63] D.R. Karger, P. N. Klein, R. E. Tarjan [1995] "*A randomized linear-time algorithm for finding minimum spanning trees*", Journal of the ACM, 42 (2): 321-328.
[64] N. Karmarkar [1984] "*A new polynomial-time algorithm for linear programming*", Combinatorica, 4:373-395.
[65] R.M. Karp [1991] "*An introduction to randomized algorithms*", Discrete Applied Mathematics 34, 165-201, North-Holland.
[66] R.M. Karp [1994] "*Probabilistic Recurrence Relations*", Jornal of ACM, Vol 41, n.6, pp. 1100-1136.
[67] R.M. Karp, M.O. Rabin [1987] "*Eficient Randomized Pattern Matching Algorithms*", IBM Jornal of Research and Development. 31:249-260.
[68] L.G. Khachiyan [1979] "*A polynomial algorithm for linear programming*", Soviet Mathematical Doklady, 20:191-194.
[69] V.King, S. Rao, R.E. Tarjan [1993] "*A faster deterministic maximum flow algorithm*", In Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, pages 157-164.
[70] D.E. Knuth [1971] "*Seminumerical Algorithms*", vol. 2 of The Art of Computer Programming, Addison-Wesley, Reading, MA.
[71] D.E. Knuth [1973] "*Sorting and searching*", vol.3 of The Art of Computer Programming, Addison-Wesley.
[72] A. N. Kolmogorov [1933] "*Foundations of the theory of probability*", Chelsea, N. York, 1950. Tradução para o inglês de *Grundbegriffe der Wahrscheinlichkeits rechnung*.
[73] E. Koutsoupias, M. Grigni, C. H. Papadimitriou[1995] "An Approximation Scheme for Planar Graph TSP", In. Proc. IEEE Symposium on Foundations of Computer Science, pp. 640-645.
[74] H. R. Lewis, C. H. Papadimitriou; [1981] "*Elements of the Theory of Computation*", Prenctice-Hall, Inc.
[75] E.L. Lima [1982] "*Curso de análise. Vol.1*", Projeto Euclides.
[76] L. Lovász [1975] "*Three short proofs in graph theory*", J. Combinatorial Theory, B, 19, 111-113.

[77] C. Lund, M. Yannakakis [1994] "*On the hardness of approximating minimization problems*", Journal of ACM, 41:960-981.
[78] U. Manber, [1989] "*Introduction to Algorithms: A Creative Approach*", Addison-Wesley Publishing Company.
[79] N. Metropolis, S. Ulam [1949] "*The Monte Carlo Method*", J. Amer. Stat. Assoc., 44, 335.
[80] P. Meyer [1983] "*Probabilidade: Aplicações à Estatística*", (2ª edição) Livros Técnicos e Científicos Editora S.A.
[81] S. Micali, V. V. Vazirani [1980] "*An $O(|V|^{1/2}|E|)$ algorithm for finding maximum matching in general graphs*", Proc. 21st IEEE FOCS, 17-27.
[82] R. Motwani [1992] "*Lecture Notes on Approximation Algorithms - Volume I*", Department of Computer Science Stanford University, Stanford, CA 94305-2140.
[83] R. Motwani, P. Raghavan, [1995] "*Randomized Algorithms*", Cambridge University Press.
[84] R. Motwani, J. Naor, P. Raghavan, [1997] "*Randomized approximation algorithms in combinatorial optimization*", in Approximation Algorithms for NP-hard problems, Hochbaum (ed), PWS.
[85] V. Natarajan [2000] "*Randomization in approximation algorithms*", CP237: Term Paper.
[86] S. Naor [1992] "*Probabilistic Methods in Computer Science*", Technical Report, International Computer Science Institute/ Berkeley and Computer Science Department/ Stanford University.
[87] G. Nemhauser, L. Wolsey [1988] "Integer and Combinatorial Optimization", John Wiley&Sons.
[88] G. I. Orlova, Y.G. Dorfman [1972] "*Finding the maximal cut in a graph*", Engineering Cybernetics, pages 502-506.
[89] C. H. Papadimitriou [1994] "*Complexity Theory*", Addison-Wesley, Reading, MA.
[90] C. Papadimitriou, K. Steiglitz [1982] "*Combinatorial Optmization: Algorithms and Complexity*", Prentice-Hall, Englewood Cliffs, NJ.
[91] S.Phillips, J.Westbrook [1991] "*Online load balancing and network flow*", In Proceedings of the 25th Annual ACM Symposium on Theory of Computing, pages 402-411.
[92] S. Poljak, Z. Tuza [1995] "*The max-cut problem - a survey*", In W. Cook, L. Lovász, and P. Seymour, editors, *Special Year on Combinatorial Optimization,* DIMACS series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.
[93] M.O. Rabin [1976] "*Probabilistic algorithms*", In J.F. Tranb, editor, Algorithms and complexity: New Directions and Recent Results, pages 21-39, Academic Press.
[94] P. Raghavan [1988] "*Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs*", Journal of Computer and System Sciences 37, 130-143.
[95] P. Raghavan and C.D. Thompson [1987] "*Randomized Rounding: Provably good algorithms and algorithmic proofs*", Combinatorica 7, 365-374.
[96] M.G.C. Resende [1998] "*Greedy Randomized Adaptive Search Procedure (GRASP*) "; AT&T Labs Research Technical Report: 98.41.1
[97] D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis [1977] "*An analysis of several heuristics for the traveling salesman problem*", SIAM Journal on Computing, 6, pp. 565-581.
[98] H. Ryser [1963] "*Combinatorial Mathematics*", The Mathematical Association of America.
[99] S. Sahni, T. Gonzalez [1976] "*P-complete approximation problems*", Journal of the ACM, 23: 555-565.
[100] A. Schönhage, M. Paterson and N. Pippenger. [1976] "*Finding de median*", Journal of Computer and System Sciences, 13:184-199.
[101] R. Sedgewick, [1986] *"Algorithms"*, Addison-Wesley, Readings, Ma.
[102] J. Shallit [1992] "*Randomized algorithms in "primitive cultures"",* SIGACT News, 23(4):77-80.
[103] S. Singh [1997] "*O último teorema de Fermat*", Ed. Record.
[104] R. Solovay, V. Strassen [1977] "*A fast Monte-Carlo test for primality"* SIAM J. Comput. 6, 84-85.
[105] Spencer [1987] "*Ten Lectures on the Probabilistic Method*", CBMS-NSF Regional Conference Series in App. Math., N.52, SIAM.

[106] A. Srinivasan [1995*]* "*Improved Approximation Guarantees for Packing and Covering Integer Programs*", DIMACS Technical Report 95-37, September 1995.
[107] A. Srinivasan [2001] "*Approximation algorithms via randomized rounding: a survey"* , Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974-0636, USA.
[108] V. Strassen [1969] "*Gaussian elimination is not optimal*", Numerische Mathematik, 14 (3): 354-356.
[109] M. Syslo, N. Deo, J. Kowalik, [1983] "*Discrete Optimization Algorithms with Pascal Programs*", Prentice-Hall, Inc.
[110] J. L. Szwarcfiter, [1984] *"Grafos e Algoritmos Computacionais"*, Editora Campus, RJ.
[111] J. L. Szwarcfiter e Lilian Markezon, [1994] "*Estruturas de Dados e seus Algoritmos"*, LTC Editora , Rio de Janeiro RJ.
[112] R. Terada, [1991] "*Desenvolvimento de Algoritmos e Estruturas de Dados*", Ed. McGraw-Hill e Makron do Brasil, São Paulo SP.
[113] R. Terada [1990] "*Introdução à Complexidade de Algoritmos Paralelos*", VII Escola de Computação - São Paulo.
[114] M. Tompa [1991] "*Lecture Notes on Probabilistic Algorithms and Pseudorandom Generators*", Technical Report #91-07-05, Department of Computer Science and Engineering, Univ. of Washington.
[115] W.T. Tutte [1947] "*The factorization of linear graphs*", J. London Math. Soc. 22, 107-111.
[116] P. Vaidya [1988] "*Geometry helps in matching*", In Proc. 20 th ACM Symp. Theory of Computing, pages 422-425.
[117] L.G. Valiant[1979] "*The complexity of computing the permanent*", Theorical Computer Science, 8:189-201.
[118] L.G. Valiant[1982] "*The complexity of enumeration and reliability*", SIAM Journal on Computing, 11:350-361.
[119] V.V. Vazirani [1994] "*A theory of alternating paths and blossons for proving correctness of the O($V^{1/2}E$) graph maximum matching algorithms",* Combinatorica, 14(1):71-109.
[120] V.V. Vazirani [1997] "*Approximation algorithms*", Preliminary and Incomplete, College of Computing, Georgia Institute of Technology.
[121] E. Waisbard, G. Weiss [1998] "*Introduction to Complexity Theory: Lecture 7:Randomized Computations",* Lecture Notes for a course given by O. Goldreich at the Weizmann Institute of Science, Israel.
[122] D.J.A. Welsh [1983] "*Randomized algorithms*"; Discrete Applied Mathematics 5, 133-145. North-Holland Publishing Company.
[123] E. Welzl [2000] "*Basic Exemples of Probabilistic Analysis, Parts I and II*", Reading Assignment for PreDoc Course on Randomized Algorithms, ETH Zürich.
[124] L. A. Wolsey [1998] "Integer Programming", Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley&Sons.
[125] S.J. Wright [1997] "*Primal-Dual Interior-Point Methods*"; SIAM - Society for Industrial and Applied Mathematics.