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 familiarizados com algumas ferramentas e definições presentes em teoria de complexidade de algoritmos e probabilidade aconselha-se uma leitura prévia dos Capítulos I e II. Neles, são definidos alguns termos essenciais à compreensão dos algoritmos randômicos. No Capítulo II, uma atenção especial é dada ao estudo de distribuições de probabilidade envolvendo variáveis aleatórias discretas. Isto se deve ao fato de que, similarmente à máquina RAM determinística (ou máquina de Turing determinística), a máquina RAM probabilística (ou Turing probabilística) trabalha apenas com números inteiros.

III.2 - MÉTODOS DE LAS VEGAS E MONTE CARLO

Além dos comandos e estruturas usuais de repetição, decisão e atribuição presente nos algoritmos determinísticos, os algoritmos randômicos ou probabilísticos se caracterizam, fundamentalmente, pela geração de números "aleatórios" ou mais precisamente, pela geração de números pseudo-aleatórios. Como discutido anteriormente, este conceito de algoritmo vai contra àquele de algoritmo puramente determinístico ou de função onde, a cada elemento do domínio (*input*) correspondia um único elemento da imagem (*output*). Nos algoritmos determinísticos, a solução obtida e o tempo de processamento nunca se alteram a cada nova repetição do algoritmo. Ao contrário, nos algoritmos randômicos, a qualidade da solução, o tempo de processamento, ou ambos, definem variáveis aleatórias obedecendo algum tipo de distribuição estatística.

Basicamente, dois tipos de algoritmos randômicos, denominados Métodos de *Las Vegas* e *Monte Carlo* são conhecidos na literatura. O método de Las Vegas sempre produz uma solução correta. Neste caso, o tempo de processamento pode ser representado por uma variável aleatória que assume valores distintos baseado em eventos aleatórios. No método de Monte Carlo, pode-se obter uma resposta incorreta com probabilidade não-nula. Neste caso, o tempo de processamento poderá ou não depender de escolhas aleatórias feitas no algoritmo.

O método de Las Vegas pode ser visto como um caso particular do método de Monte Carlo. Isto ocorre sempre que a probabilidade de fracasso for igual a *zero.* Quando a probabilidade de falha no método de Monte Carlo for estritamente positiva, pode-se transformar Monte Carlo em Las Vegas adicionando-se um procedimento (idealmente polinomial) de checagem da solução. Sempre que a solução estiver incorreta, o processo será repetido até que uma solução exata seja encontrada. Para exemplificar esta situação, suponha um experimento ε onde um dado equilibrado é jogado e seu resultado observado. Considere que um evento *E* associado, ocorra sempre que um número *par* for observado. Neste caso, se o experimento for executado uma única vez, tem-se o método de Monte Carlo com probabilidade de sucesso $p=1/2$. Por outro lado, se o experimento for realizado

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 $T_1(n)+T_2(n)$ tem-se que $(T_1(n)+T_2(n))/p(n)$ irá representar o tempo esperado de processamento no método de Las Vegas. Apesar de interessante, esse tipo de abordagem nem sempre pode ser aplicado na construção do método de Las Vegas (a partir do método de Monte Carlo). Na verdade, não existe um procedimento padrão que se aplique a todos os casos.

A transformação de Las Vegas em Monte Carlo pode ser feita interrompendo-se o método de Las Vegas em uma dada iteração do procedimento. Em seguida, estima-se a probabilidade de sucesso.

III.3 - MÉTODO DE LAS VEGAS

Dado um problema π, o método de Las Vegas sempre assegura uma solução exata para o problema. Entretanto, ela nem sempre é obtida dentro de um tempo de execução suportável. Essa abordagem é interessante para aqueles problemas cujo tempo esperado de processamento seja suficientemente pequeno (ou polinomial) para a aplicação em questão. Pode-se dizer que, de certa forma, o método de Las Vegas "simula" o comportamento de uma máquina não-determinística com um número infinito de processadores (vide Capítulo I). Neste caso, apenas um único processador ou um número finito de processadores está disponível. Assim, são consideradas sucessivas repetições do mesmo algoritmo até que uma solução exata seja encontrada. Cada repetição deverá conter uma etapa de exibição, utilizando-se uma fonte de números "aleatórios", e uma etapa de reconhecimento (polinomial ou não) da solução. Como a probalidade de sucesso pode ser arbitrariamente próxima de *zero*, o tempo de processamento pode ser arbitrariamente grande (tendendo a infinito). De certa forma, o termo "infinitos processadores" na máquina não-deterministica pode ser substituído por "tempo infinito" em uma máquina determinística.

Um algoritmo de Las Vegas será considerado *eficiente* se o tempo esperado de processamento for polinomial no tamanho do problema. A seguir são apresentados alguns exemplos que ilustram melhor a aplicação do método de Las Vegas:

III.3.1:O problema da Coloração de Conjuntos

Suponha $S=\{x_1,x_2,...,x_n\}$ um conjunto de *n* elementos quaisquer. Seja $S_1,S_2,...,S_k$ uma coleção de subconjuntos distintos de *S* cada um com $r \geq 2$ elementos e tal que $k \leq 2^{r-2}$. Os elementos de *S* deverão ser pintados com *branco* ou *preto* de forma que cada um dos subconjuntos S_i (p/ $i=1,..,k$) contenha pelo menos *dois* elementos com cores distintas.

Um algoritmo randômico para este problema pode ser obtido, simplesmente, colorindo-se cada elemento de S aleatoriamente com probabilidade *1/2* (distribuição uniforme). Caso a coloração resultante contenha um ou mais subconjuntos com elementos de mesma cor, repete-se o processo até que uma coloração viável seja obtida. A figura seguinte sintetiza as principais etapas do algoritmo de Las Vegas para o problema da coloração de conjuntos. A variável *viavel* (presente no algoritmo) indica se uma coloração desejada foi ou não encontrada:

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 $r=5$.

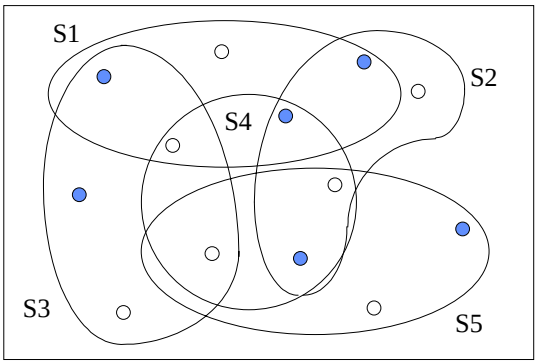


Figura III.2: Coloração válida p/ o problema de coloração de conjuntos

Para mostrar que o Algoritmo III.1 sempre encontra uma solução desejada 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 viável 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:

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

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:

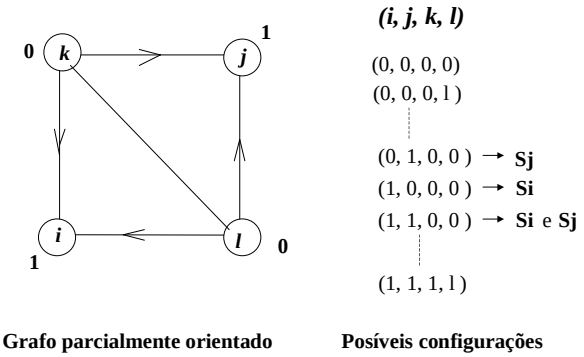


Figura III.3: Procedimento Alg-Viz.

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 $O(n^4 logn)$ que utiliza o conceito de contração de multiarestas (Karger[1993]). Karger&Stein[1993] se baseiam neste procedimento para construção de um novo algoritmo randômico de complexidade $O(n^2 log^{O(1)} n)$! Para grafos densos, este resultado melhora significativamente a complexidade de *dois* dos melhores algoritmos determinísticos conhecidos para o problema de fluxo máximo !

Considere a seguinte definição:

Definição III.1: (Contração de arestas)

Dado um multigrafo $G=(V,E)$, a *contração* da aresta (ou multiaresta) $e = [u,v] \in E$ é outro *multigrafo* $G\backslash\{e\} = (V',E')$ onde $V' = (V - \{u,v\}) \cup \{\overline{uv}\}$, $E' = (E - \{[u,v] / u$ e v são incidentes a $e\}) \cup \{[\overline{uv},x] / [u,x] \in E$ ou $[v,x] \in E\}$. O vértice $\overline{uv} \in G\backslash\{e\}$ representa a contração da aresta $[u,v]$. •

Esta definição é ilustrada no exemplo da Figura III.4. É importante observar que a contração de arestas não reduz o tamanho do corte mínimo em G.

---

[1] Um laço é todo arco (ou aresta) do tipo $[v,v]$, onde $v \in V$.

[2] Um subgrafo $G_1=(V_1, E_1)$ de G define uma *componente conexa* de G se e somente se $\forall\, i,j \in V_1$ existir um caminho unindo o vértice i ao vértice j.

[3] O par (V_1, V_2) define uma *partição* de V se e somente se $V_1 \cup V_2 = V$ e $V_1 \cap V_2 = \varnothing$. No problema do corte-mínimo considera-se ainda $|V_1|.|V_2| \neq 0.80$

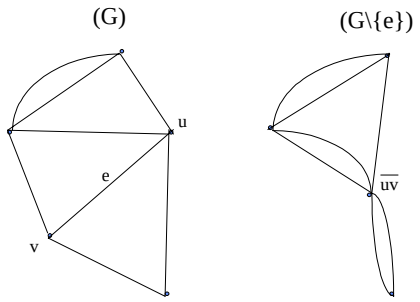


Figura III.4: Contração da aresta e=[u,v]

O algoritmo seguinte (desenvolvido por Karger[1993]) aplica sucessivas contrações de arestas (ou multiarestas) 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 G_i o multigrafo obtido após *i* contrações de *G*. O conjunto *CM* (vide Algoritmo III.2), armazena todas as arestas de G_{n-2}. Este conjunto é atualizado à medida que cortes menores forem sendo encontrados nas iterações posteriores.

Outro aspecto a ser observado é que o tempo de processamento não depende do número total de arestas pertencentes ao multigrafo G. Na verdade o número total de arestas em G não é necessariamente limitado por n(n-1)/2. Apesar de curioso, isto pode ser obtido atribuindo-se, a cada aresta, um número inteiro representando a multiplicidade de arestas entre cada par de vértices. Pode-se afirmar portanto, sem perda de generalidade, que o multigrafo G conterá no máximo $O(m)=O(n^2)$ arestas.

Algoritmo III.2: Corte-mínimo randômico
Início
leia *G=(V,E);* {grafo original}
CM ← ∅; {arestas "pertencentes" ao corte mínimo}
para *k=1* **até** *T* **faça**
- $G_0 \leftarrow G$; {copia grafo original }
para *i=1* **até** *n-2* **faça**
- escolha aleatoriamente uma aresta *e* de G_{i-1};
- $G_i \leftarrow G_i \backslash \{e\}$; {faz contração}
fim;
Atualiza *CM* se necessário;
fim;
retorna (|*CM*|);
fim.

Figura III.5: Corte-mínimo pelo método de Monte Carlo

Como discutido em Karger[1995], a escolha aleatória de uma aresta (ou multiaresta) $e \in G_{i-1}$ (p/ algum $i \in \{1,..,n-2\}$) pode ser realizada em tempo *O(n)*. As novas arestas deverão ser selecionadas com probabilidade proporcional a seu próprio peso. Deseja-se, a princípio, que as arestas de maior peso não pertençam ao corte *CM*.

A etapa de contração no Algoritmo III.2 pode ser implementada em tempo $O(n)$ (Karger[1995], Motwani & Raghavan [1995]). Esta contração é realizada garantido-se a conversão do grafo original para o grafo contraído e vice-versa. Desta forma, será possível explicitar cada um dos vértices e arestas que definem o "corte mínimo" *CM* ao final do procedimento. Conclui-se portanto que cada iteração do Algoritmo III.2 terá complexidade igual a $O(n^2)$. A determinação do número total de repetições (representado por T) será discutida mais adiante.

Este algoritmo sempre encontra um corte mínimo no grafo original? Obviamente que não, entretanto, pode-se tentar determinar qual a probabilidade de encontrá-lo. Inicialmente, deve-se avaliar a probabilidade de sucesso após uma única repetição do algoritmo (para k fixo). Para isto, considere E_i para algum $i \in \{1,..,n-2\}$, um evento indicando que uma aresta qualquer pertencente ao corte mínimo *não* tenha sido selecionada na i-ésima contração. Para que se tenha sucesso na determinação do corte mínimo todos os eventos E_i deverão ocorrer simultaneamente. Como os $n-2$ eventos *não* são independentes entre si deve-se considerar a seguinte expressão de probabilidade condicional (vide Capítulo II):

$$\Pr\left(\bigcap_{i=1}^{n-2} E_i\right) = \Pr(E_1).\Pr(E_2|E_1).\Pr(E_3|(E_1 \cap E_2)).....\Pr\left(E_{n-2}|\bigcap_{i=1}^{n-3} E_i\right)$$

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 complementar de E_1 então $Pr(E_1^c) = c/|E| \leq c/(cn/2) = 2/n$. Em outras palavras, a ocorrência de E_1^c indica que uma aresta pertencente ao corte mínimo foi selecionada na primeira iteração. Assim, $Pr(E_1) = 1-Pr(E_1^c) \geq 1-2/n$. Assumindo que o evento E_1 tenha ocorrido após a primeira contração calcula-se agora $Pr(E_2|E_1)$. Neste caso, o novo grafo G_1, resultante da contração de e pertencente a G_0 terá no mínimo $c(n-1)/2$ arestas. Seguindo raciocínio análogo ao cálculo de $Pr(E_1)$ conclui-se que:

$$\Pr(E_2|E_1) \geq 1 - \frac{2}{n-1}$$

Generalizando o processo, após a i-*ésima* contração, obtêm-se G_i com exatamente $n-i$ vértices. Segue então que:

$$\Pr\left(E_i|\bigcap_{j=1}^{i-1} E_j\right) \geq 1 - \frac{2}{n-i+1}$$

Após a substituição destes valores na expressão de probabilidade condicional obtem-se a seguinte desigualdade:

$$\Pr\left(\bigcap_{i=1}^{n-2} E_i\right) \geq \prod_{i=1}^{n-2}\left(1-\frac{2}{n-i+1}\right) = \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdots\frac{1}{3} = \frac{(n-2)!}{(n!/2)} = \frac{2}{n(n-1)} \geq \frac{2}{n^2} = \Omega(n^{-2})$$

Como a probabilidade de sucesso é maior ou igual $2/n^2$, conclui-se diretamente que a probabilidade de fracasso será menor ou igual que $1-2/n^2$. Neste caso, aumentando-se n, a probabilidade se aproxima de 1 ! Entretanto, será possível minimiza-las fazendo $T=O(n^2/2)$ no Algoritmo III.2. Assim, do cálculo tem-se que:

$$\left(1-\frac{1}{(n^2/2)}\right)^T = \left(1-\frac{2}{n^2}\right)^{n^2/2} < \frac{1}{e}$$

representa a probabilidade de fracasso após $T=O(n^2/2)$ repetições do procedimento. Desta forma, o Algoritmo III.2 terá complexidade total igual a $O(n^4)$ iterações, sempre gerando um corte mínimo com probabilidade de sucesso superior a $1-1/e!$

Apesar do aumento considerável no número de iterações do processamento, execuções adicionais deste procedimento reduzirão exponencialmente a probabilidade de fracasso. Considere 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 $\delta = \log n$ obtêm-se um novo algoritmo com complexidade igual a $O(n^4 \log n)$ e probabilidade de sucesso maior ou igual a $1-1/e^{\log n}$! 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 $O(n^2 \log n)$ 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,

[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.

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 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 $O(n^{3/4}logn)$;
  - 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.

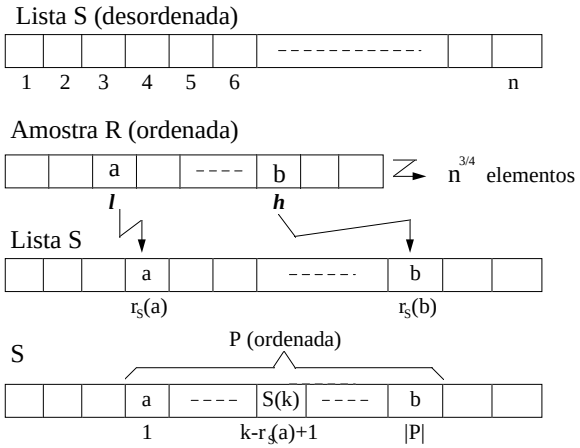


Figura III.7 Procedimento *LasySelect* para $k \in [n^{1/4}, n - n^{1/4}]$

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

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

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 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 ≤ l).* Da mesma forma pode-se concluir que: *Pr(S(k) > b) = Pr(X ≥ 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:

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

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 $(k/n)(1-k/n) \le 1/4$ (verifique).

Da desigualdade de Tchebyshev tem-se:

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

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

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

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

$$\Pr(X \ge h) = \Pr\left(X \ge kn^{-1/4} + \sqrt{n}\right) \le \left(4n^{1/4}\right)^{-1} \cdot$$

Como *Pr(A) = Pr(X ≤ l)* e *Pr(B) = Pr(X ≥ 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*.

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

A estratégia neste caso será considerar *u* e *v* (onde $1 \le u \le v \le n$) de maneira que $Pr(a<S(u)) + Pr(b>S(v))$ defina um limite superior para a probabilidade de falha. Como a sub-lista *P=[a,b]* deve ser ordenada a um baixo custo computacional, espera-se que $|P| = r_S(b) - r_S(a) + 1$ seja minimizada fazendo-se $r_S(a) = u-1$ e $r_S(b) = v+1$ respectivamente (veja Figura III.9).

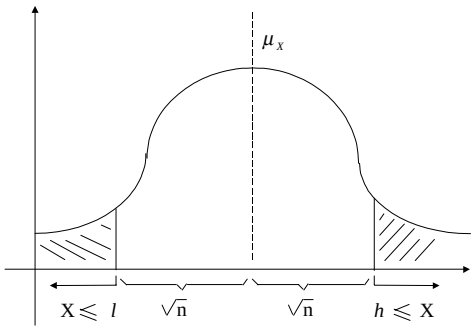


Figura III.8: Desigualdade de Tchebyshev

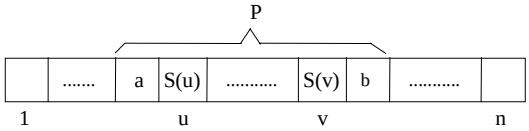


Figura III.9: Determinação de *u* e *v*

Considere inicialmente a situação onde a probabilidade de falha é igual a *Pr(a<S(u))*. Se *a<S(u)* em *S* então pelo menos *l* elementos na amostra *R* serão menores que *S(u)*.

Seja Y_i uma variável aleatória associada ao *i*-ésimo elemento da amostra *R* como abaixo:

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

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 $W \le v$ representa outra possibilidade de falha. Logo, $Pr(W \le v) = Pr(W \le kn^{-1/4} + n^{1/2}) \le (4n^{1/4})^{-1} \Rightarrow Pr(kn^{-1/4} - W \ge -n^{1/2}) = Pr(kn^{-1/4} + n^{1/2} - W \ge n^{1/2}) \le (4n^{1/4})^{-1}$ (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, $Pr(|P|>4n^{3/4}+2) = Pr(a<S(u)) + Pr(b>S(v)) \le O(n^{-1/4})$. Provou-se então o seguinte resultado:

Proposição III.6: O algoritmo *LazySelect* encontra $S(k)$ na primeira iteração do procedimento com 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).

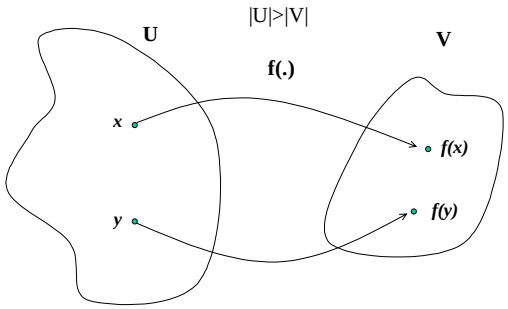


Figura III.10 : Impressão Digital (*Fingerprinting*)

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* ← *1;*
- **Repita**
  - escolhe $x \in \{0,1\}^n$ aleatoriamente;
  - **Se** $ABx \neq Cx$ **então**
    continua ← false;
  - *k* ← *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.

III.4.4.2 – Verificando igualdade entre polinômios

Considere *P(x)* e *Q(x)* dois polinômios de grau *n* 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 $O(n\log n)$ 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 comparação entre $P_1(x).P_2(x)$ e $P_3(x)$ será $O(n)$. Assim, a complexidade total deste procedimento será $O(n\log n)$.

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 $Q(x) \not\equiv 0$ e r é raiz de $Q(x)$. Por outro lado, se $Q(x) \equiv 0$ então, $Q(r)=P_1r).P_2,(r) - P_3(r)=0, \forall r \in F$. Assim, caso se tenha $Q(x) \not\equiv 0$, espera-se que $Pr(Q(r)=0)$ seja próxima de 0. Como $Q(x) \not\equiv 0$, 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:

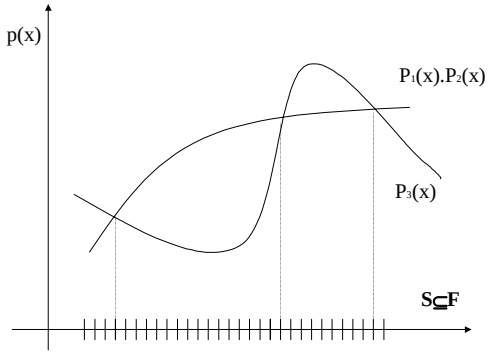


Figura III.13: Raízes de $Q(x) \not\equiv 0$

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 $Q(x) \not\equiv 0$. 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* $\sigma_I = (1,2,...,n)$. Note por exemplo que, se $\sigma = (2,3,1)$ então $sinal(\sigma) = 2$ pois:

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

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]). $\bullet$

[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.

A proposição seguinte retorna a probabilidade de falha, se $Q(x) \not\equiv 0$:

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:

$$Pr(Q(a_1, a_2,..., a_n))=0 \mid Q(x_1, x_2,..., x_n) \not\equiv 0) = d/|S|$$

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

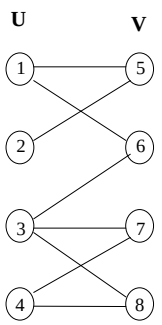
Esta seção ilustra a importância da utilização de polinômios com múltiplas variáveis (através de técnicas algébricas) na determinação da existência, exibição e enumeração de Emparelhamentos Perfeitos em Grafos.

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:



$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

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.

$$P_1=\begin{pmatrix}(1) & 1 & 0 & 0\\ 1 & 0 & 0 & (0)\\ 0 & (1) & 1 & 1\\ 0 & 0 & (1) & 1\end{pmatrix} \rightarrow P_2=\begin{pmatrix}1 & (1) & 0 & 0\\ (1) & 0 & 0 & 0\\ 0 & 1 & (1) & 1\\ 0 & 0 & 1 & (1)\end{pmatrix}$$

$$\sigma_1=(5,8,6,7) \qquad \sigma_2=(6,5,7,8)$$

$$P_3=\begin{pmatrix}1 & (1) & 0 & 0\\ (1) & 0 & 0 & 0\\ 0 & 1 & 1 & (1)\\ 0 & 0 & (1) & 1\end{pmatrix} \rightarrow P_4=\begin{pmatrix}(1) & 1 & 0 & 0\\ 1 & 0 & (0) & 0\\ 0 & 1 & 1 & (1)\\ 0 & (0) & 1 & 1\end{pmatrix} \quad \textbf{etc}$$

$$\sigma_3=(6,5,8,7) \qquad \sigma_2=(5,7,8,6)$$

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, $det(A) \not\equiv 0$.

Prova: Motwani&Raghavan[1995] •

Para responder se $det(A) \not\equiv 0$, 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:

$$Pr(det(A)=0 \mid det(A) \not\equiv 0) \leq 1/2$$ •

A aplicação da Proposição III.8 segue diretamente já que o polinômio det(A) tem grau *n* e $|S|=2n$.

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** *cada* $(u_i, v_j) \in E$ **faça**
    $x_{ij} \leftarrow$ Escolhe$\{1,2,3,..,2n\}$;
  - **Se** $det(A) \neq 0$ **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:

[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 $a_{ii}=0$, $p/\ i=1,..,n$.

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

Então, *G* admite emparelhamento perfeito se, e somente se, $det(A) \not\equiv 0$. 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:

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

•

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:

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

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

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:

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

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 $x \in \{0,1\}^*$;
    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:

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

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 $r \leq k/2$ então:

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

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

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

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

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

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)*:

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

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

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

Ou ainda:

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

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:

i) $x \in L \Rightarrow Pr(A(x) = 1) > 1/2$
ii) $x \notin L \Rightarrow Pr(A(x) = 0) > 1/2$ •

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)!! Em outras palavras, pode-se dizer um problema de decisão $\pi \in BPP$, se o número de repetições for polinomial na geração de uma solução com erro inferior a $\varepsilon > 0$. Se o número de repetições for exponencial então $\pi \in PP$.

As seguites proposições são verdadeiras: $P \subseteq ZPP$, $NP \subseteq PP$, $RP \cup co\text{-}RP \subseteq BPP$, RP $\subseteq$ NP, co-RP $\subseteq$ $co\text{-}NP$, 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].

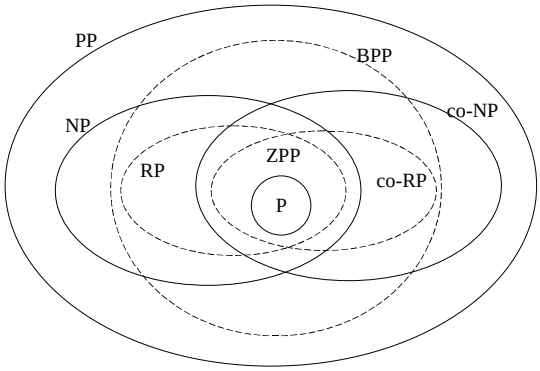


Figura III.20: Classes de Complexidade

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