Capítulo IV

Algoritmos Aproximativos: Determinísticos e Randômicos

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

IV.1 - *INTRODUÇÃO:*

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

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

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

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

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

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

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

IV.2 – ALGORITMOS DETERMINÍSTICOS APROXIMATIVOS

IV.2.1 – Conceitos Básicos e Definições:

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

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

Definição IV.1: (Algoritmos de aproximação absoluta)

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

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

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

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

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

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

Considere agora a seguinte definição:

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

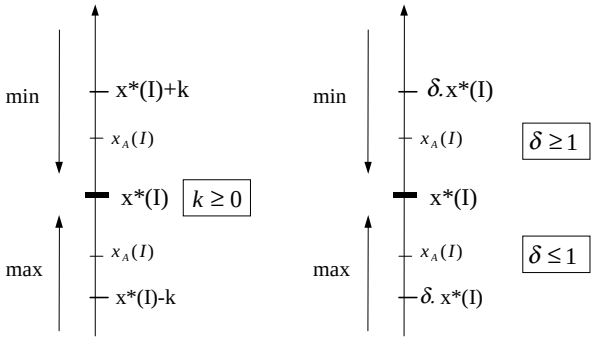
Um algoritmo A com solução $x_A(I)$ é *f(n)-aproximado* para um problema de minimização (de maximização) Π se, e somente se, qualquer que seja a instância I de tamanho *n*, a solução obtida é no máximo (no mínimo) *f(n)* vezes o valor da solução ótima *x*(I).* •

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

Definição IV.3: (Algoritmo δ-aproximado)

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

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



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

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

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

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

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

Definição IV.4: (Esquema de aproximação polinomial)

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

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

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

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

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

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

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

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

O problema da Programação de Tarefas Independentes também conhecido na literatura como *Scheduling Independent Tasks* foi o primeiro problema resolvido por algoritmos aproximativos. Nele, um conjunto de n tarefas devem ser atribuídas a um conjunto de m máquinas idênticas, sendo que, cada tarefa, consome um tempo pré-determinado de processamento. O objetivo será atribuir o conjunto de tarefas ao conjunto de máquinas minimizando o tempo total de utilização das m máquinas. Neste caso, como não se exige nenhuma relação de precedência entre as tarefas elas são chamadas de *independentes*. Como discutido em Garey e Johnson[1979], este problema é NP-Árduo mesmo para $m \geq 2$ e define uma instância particular do *Minimum Makespan Problem*.

Para exemplificar este problema, considere o seguinte exemplo apresentado em Hochbaum [1997]. Suponha que sejam 9:00h da manhã, e que um funcionário de uma empresa deseja processar 147 tarefas em 8 máquinas disponíveis. Ele espera que todas as tarefas estejam concluídas antes de uma partida de futebol a ser transmitida pela TV às 21:00h. Como um tempo de preparação das 8 máquinas é necessário, o funcionário dará início aos processos exatamente às 10:00h. O funcionário deverá definir ainda, qual o melhor pacote de programação a ser utilizado. A opção por um método exato, como *branch-and-bound* ou programação dinâmica por exemplo pode ser bastante arriscada visto que a solução a ser gerada dentro da pequena faixa de tempo disponível (no máximo 1 hora) pode ser de baixa qualidade quando comparada à de métodos heurísticos mais específicos. Uma outra estratégia poderia ser, simplesmente, a atribuição de tarefas escolhidas arbitrariamente às máquinas que forem sendo desocupadas. Suponha, entretanto, que depois de alguns cálculos ele

[2] Seja L um conjunto de índices (representados por ε). Dado um conjunto X, uma *família* de elementos de X com índices em L, é uma função $x: L \to X$. A família é representada pela notação $\{x_\varepsilon\}_{\varepsilon \in L}$ ou simplesmente $\{x_\varepsilon\}_\varepsilon$. Por exemplo, quando $L = \{1,2,..,n\}$ e $X = Z$, a *n-upla* de números inteiros $x = \{x_\varepsilon\}_{\varepsilon \in L}$ pode ser representada por $x = (x_1, x_2, \ldots, x_n)$.

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

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

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

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

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

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

Se Π é um problema de maximização então:

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

Note que, independentemente do problema Π ser de maximização ou minimização, a performance do algoritmo é melhor quando $R_A(I)$ se aproxima de 1.

Definição IV.7: (Razão de performance absoluta)

A *razão de performance absoluta* R_A de um algoritmo aproximativo *A* para um problema de otimização Π é definida por: $R_A = inf\{r \mid R_A(I) < r\ , \forall I \in \Pi\ \}$[3]. •

A Figura IV.2 ilustra bem este conceito. Note que R_A representa o ínfimo do intervalo (R_A,∞). Ainda, se um algoritmo *A* é δ-aproximado para um problema de otimização Π então ele tem razão de performance absoluta igual a R_A.

Estudos experimentais indicam que a razão de performance $R_A(I)$ observada para uma instância *I* particular é, normalmente, bem inferior ao da razão de performance absoluta R_A Hochbaum[1997]. Isto se deve à presença de instâncias cujo desempenho do algoritmo não é

[3] Se $X \subset \Re$ então $a \in \Re$ é *ínfimo* (ou inf) de X se e somente se *a* for a maior das cotas inferiores do conjunto X. Lembre-se que $a \in \Re$ é *cota inferior* de X se e somente se $a \leq x, \quad \forall x \in X$.

satisfatório (pior caso). Uma maneira de contornar esta dificuldade é através de uma análise de comportamento médio. Neste caso, é importante que se conheça uma distribuição dos dados associados ao problema. Coffman, Garey e Johnson ilustram a utilização do caso médio aplicada ao *Bin Packing problem* (vide Hochbaum[1997] - Capítulo 2).

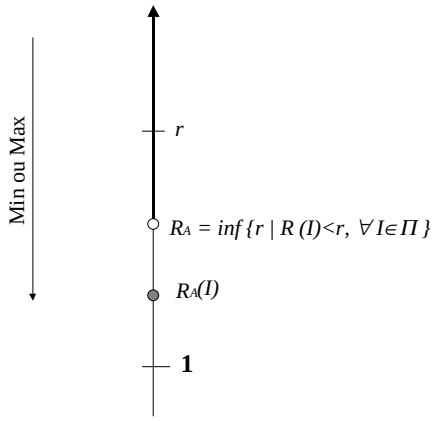


Figura IV.2: Razão de Performance Absoluta R_A

A menos que *P=NP*, a adoção algoritmos de aproximação relativa (ou absoluta), esquemas de aproximação polinomiais, totalmente polinomiais ou mesmo algoritmos δ-aproximados não pode ser aplicada indistintamente a todos os problemas combinatórios. Nas Seções IV.2.2 e IV.2.3, serão apresentados *dois* problemas clássicos da literatura resolvidos por algoritmos aproximativos e alguns resultados negativos associados (Seção IV.2.4).

VI.2.2 – O problema da Programação de Tarefas Independentes - PTI

No problema da Programação de Tarefas Independentes (instância particular do *makespan problem*) deseja-se atribuir *n* tarefas a *m* máquinas idênticas. Cada uma das *n* tarefas possuem um tempo t_i associado. O objetivo é minimizar o tempo de duração da última tarefa processada. As *m* máquinas devem funcionar em paralelo e estão inicialmente vazias. Como discutido anteriormente, este problema é NP-Árduo para $m \geq 2$.

Seja x_{ij} uma variável binária que indica se a tarefa i foi ou atribuída ao processador j. Se z representa o término de processamento da última tarefa tem-se então a seguinte formulação matemática para o problema:

$$\textit{minimizar} \quad z$$

$$\textit{sujeito a:} \begin{cases} \sum_{i=1}^{n} t_i x_{ij} \leq z, & j = 1,2,\ldots,m \\ \sum_{j=1}^{m} x_{ij} = 1, & i = 1,..,n \end{cases}$$

$$x_{ij} = 0 \text{ ou } 1 \quad \textit{onde} \quad i, j = 1,..,n$$

O conjunto de desigualdades garante um limite superior para o tempo de duração de cada uma das *m* máquinas enquanto que o conjunto de igualdades obriga que cada tarefa seja atribuída a exatamente uma máquina.

Um algoritmo bastante simples para este problema, conhecido na literatura como algoritmo de escalonamento de listas (*List Scheduling Algorithm - LSA*) consiste simplesmente na atribuição de uma tarefa qualquer ao primeiro processador que se tornar ocioso. O processo é repetido até que todas as tarefas tenham sido concluídas. Dada uma instância qualquer, Graham [1966] provou que esta heurística tem razão de performance na instância I menor ou igual a *2-1/m*. Note que, quando

$m \to \infty$ então $R_A(I) \to 2$. Pode-se dizer portanto que o algoritmo de escalonamento de listas tem razão de performance absoluta igual a 2 (algoritmo 2-aproximado). Se $m=1$, tem-se uma instância particular onde o algoritmo de escalonamento de listas se torna exato ($R_A(I)=1$).

Proposição IV.1: Seja I uma entrada qualquer e $x_A(I)$, o tempo total de processamento do algoritmo de escalonamento de listas (representado por A) das n tarefas nas m máquinas. Então,

$$R_A(I) \leq \left(2 - \frac{1}{m}\right).$$

Além disso, tem-se garantida a existência de uma instância $I^* \in \Pi$ onde:

$$R_A(I^*) = \left(2 - \frac{1}{m}\right).$$

Prova: Sem perda de generalidade, considere n a última tarefa processada e t_n seu tempo de processamento associado. Além disso, considere que a tarefa n seja processada na máquina M_1. Note que nenhuma máquina deverá estar inativa ao final de $x_A(I)-t_n$ unidades de tempo. Caso contrário, existiria uma máquina M_k (para algum $k \neq 1$) cujo término seria anterior a $x_A(I)-t_n$. Tem-se portanto um absurdo pois, do algoritmo de escalonamento de listas, a inserção da tarefa n seria em M_k e não M_1.

Note agora que:

$$\sum_{i=1}^{n} t_i - t_n \geq m.(x_A(I) - t_n), \quad \forall I \in \Pi \qquad (01)$$

A desigualdade acima pode ser melhor compreendida com o auxílio das Figuras IV.3.(a) e IV.3.(b). A região A representa a área total definida por $m(x_A(I)-t_n)$ enquanto que a região B representa a área definida por:

$$\sum_{i=1}^{n-1} t_i - t_n .$$

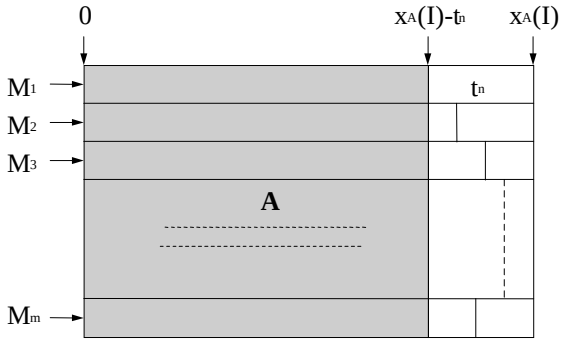


Figura VI.3.(a): Região A

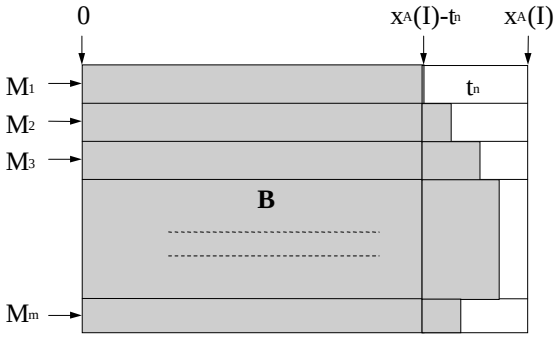


Figura VI.3.(b): Região B

É fácil ver que:

$$\frac{1}{m}\sum_{i=1}^{n} t_i \leq x^*(I), \quad \forall I \in \Pi \qquad (02)$$

define um limite inferior para o valor da solução ótima. Substituindo (02) em (01) obtém-se a seguinte expressão:

$$m.x^*(I) - t_n \geq m.(x_A(I) - t_n), \quad \forall I \in \Pi$$
$$m.x^*(I) \geq m.x_A(I) + t_n(1-m) \quad \forall I \in \Pi \qquad (03)$$

Como $t_n \leq x^*(I)$ e $m \geq 1$ então:

$$t_n(1-m) \geq x^*(I).(1-m) \qquad (04)$$

Substituindo (04) em (03) obtém-se:

$$m.x^*(I) \geq m.x_A(I) + x^*(I).(1-m) \quad \forall I \in \Pi$$

Após as devidas simplificações chega-se a:

$$\frac{x_A(I)}{x^*(I)} \leq \left(2 - \frac{1}{m}\right), \quad \forall I \in \Pi \cdot$$

Para mostrar a segunda parte da proposição considere uma instância *I** onde *n=m(m-1)+1*. Suponha ainda que o tempo de execução das *n* tarefas seja dado por $t_i=1$, para *i=1,..,n-1* e $t_n=m$. É fácil ver neste caso que *x*(I*) = m* (tempo de duração das *n* tarefas nas *m* máquinas) enquanto que $x_A(I^*) = 2m-1$ é o valor obtido pelo algoritmo de escalonamento de listas (vide Figura IV.4).

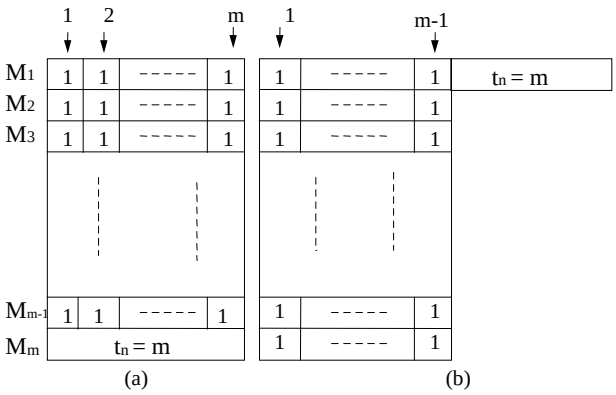


Figura IV.4: (a) Solução ótima *x*(I*)* (b) Solução heurística $x_A(I^*)$

Na heurística *LPT* (*Longest Processing Time*) , os processos são ordenados inicialmente por ordem decrescente de tamanho. Em seguida, eles são colocados em operação nas máquinas ociosas sempre respeitando-se esta ordem. A complexidade total será de *O(nlogn)* iterações (Verifique!). O seguinte exemplo ilustra bem esta situação:

Exemplo IV.2: (Heurística LPT)

Considere que *n=12* tarefas devam ser alocadas a *m=3* máquinas distintas. O tamanho de cada tarefa é dado por:

$$t_1 = 6, \quad t_2 = 6, \quad t_3 = 5, \quad t_4 = 5, \quad t_5 = 4, \quad t_6 = 4, \quad t_7 = 3, \quad t_8 = 3, \quad t_9 = 3, \quad t_{10} = 2, \quad t_{11} = 1, \quad t_{12} = 1$$

A solução esta representada na Figura IV.5. Note que, como os tempos de processamento de cada tarefa são inteiros segue que:

$$\left\lfloor \frac{1}{3}\left(\sum_{i=1}^{12} t_i\right) \right\rfloor + 1 = \left\lfloor \frac{43}{3} \right\rfloor + 1 = 15$$

é um limite inferior para o valor da solução ótima *x*(I).* Como $x_A(I)=15$, a solução obtida pela heurística também é solução ótima. Note que, $15 \leq x^*(I) \leq x_A(I) \leq 15$.

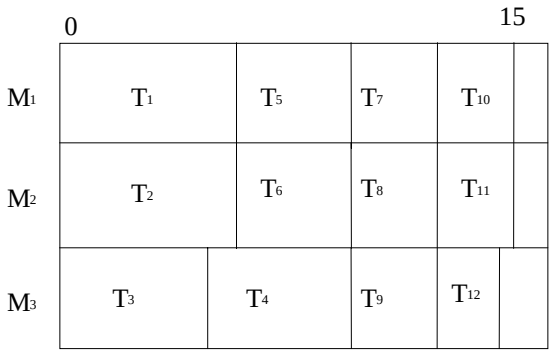


Figura IV.5: Heurística LPT

O resultado seguinte demonstrado por Graham[1966] (Proposição IV.3) garante um desempenho superior da heurística *LPT* em relação à heurística de escalonamento de listas. Considere entretanto o seguinte resultado auxiliar (Proposição IV.2). Sua demonstração será deixada como exercício.

Proposição IV.2: Suponha que, em uma solução ótima, não mais que 2 tarefas sejam associadas a cada máquina. Então, o escalonamento gerado pela heurística *LPT* também é ótimo.

Proposição IV.3: (Heurística LPT)

Seja $x_A(I)$ o tempo total de processamento da heurística *LPT* (representado por *A*) das *n* tarefas nas *m* máquinas. Então,

$$\frac{|x_A(I) - x^*(I)|}{x^*(I)} \leq \frac{1}{3} - \frac{1}{3m}.$$

Prova: A proposição se verifica imediatamente para $m=1$ (caso trivial). Considere então $m \geq 2$. Suponha, por absurdo, que exista um conjunto $S=\{t_1,t_2,...,t_n\}$ de tarefas onde a desigualdade não se verifica, ou seja:

$$\frac{|\theta_k - \theta_k^*|}{\theta_k^*} > \frac{|x_A(I) - x^*(I)|}{x^*(I)} > \frac{1}{3} - \frac{1}{3m} \qquad \text{(I)}$$

Além disso, considere S o conjunto com menor número de elementos satisfazendo essa propriedade. Sem perda de generalidade seja, $t_1 \geq t_2 \geq ... \geq t_n$ e $1,2,..,.n$ a ordem de escolha das *n* tarefas alocadas às *m* máquinas.

Seja *k* um índice associado a última tarefa processada. Pode-se mostrar neste caso que k=n. Se $k<n$ então o tempo de término das tarefas $1,2,...,k$ será dado por $x_A(I)=\theta_k$. Observando-se ainda as *k* primeiras tarefas conclui-se que $\theta_k^* \leq x^*(I)$ onde θ_k^* representa o tempo ótimo de processamento associado a estas *k* tarefas. Logo,

$$\frac{|\theta_k - \theta_k^*|}{\theta_k^*} > \frac{|x_A(I) - x^*(I)|}{x^*(I)} > \frac{1}{3} - \frac{1}{3m}$$

Observe agora que as tarefas $S'=\{t_1,t_2,...,t_k\}$ (onde $k<n$) podem ser alocadas aos *m* processadores de maneira que o resultado da proposição não se verifique. Isto é absurdo pois, considerava-se, *S* o menor conjunto satisfazendo (I). Logo $k=n$.

Como t_n é a última tarefa processada tem-se, de acordo com a heurística *LPT* que, no instante $x_A(I)-t_n$, nenhum outro processador estava ocioso naquele momento. Logo, seguindo raciocínio

análogo ao utilizado na Proposição IV.1 conclui-se que:

$$m.(x_A(I) - t_n) \leq \sum_{i=1}^{n-1} t_i$$

Então:

$$m.x_A(I) \leq \sum_{i=1}^{n-1} t_i + m.t_n = \sum_{i=1}^{n} t_i + (m-1).t_n$$

Dividindo esta desigualdade por $m \geq 2$ e lembrando que:

$$x^*(I) \geq \frac{1}{m}\sum_{i=1}^{n} t_i$$

Chega-se a:

$$x_A(I) - x^*(I) \leq \frac{m-1}{m} t_n$$

Como $x^*(I) > 0$ segue:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} \leq \frac{m-1}{m} \cdot \frac{t_n}{x^*(I)}$$

Da hipótese de absurdo tem-se:

$$\frac{1}{3} - \frac{1}{3m} < \frac{m-1}{m} \cdot \frac{t_n}{x^*(I)}$$

Multiplicando ambos os lados dessa desigualdade por $3m$ conclui-se que:

$$m - 1 < 3(m-1)\frac{t_n}{x^*(I)} \qquad \Rightarrow \qquad x^*(I) < 3t_n$$

Como t_n é a menor de todas as tarefas segue então que, em um escalonamento ótimo, não mais que 2 tarefas devem ser adicionadas a uma mesma máquina. Da Proposição IV.2, se a heurística *LPT* é aplicada a uma instância I, e um escalonamento ótimo possui no máximo 2 tarefas associadas a cada máquina então $x_A(I) = x^*(I)$. Logo:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} = 0$$

Observe que a situação acima é absurda pois considerava-se:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} > \frac{1}{3} - \frac{1}{3m} > 0, \quad \forall\ m \geq 2.$$

Conclui-se então que:

$$\frac{|x^*(I) - x_A(I)|}{x^*(I)} \leq \frac{1}{3} - \frac{1}{3m}, \quad \textit{onde} \quad I \in \Pi. \qquad \bullet$$

Observe que, para toda instância I com m é arbitrariamente grande, a razão de performance $R_A(I)$ se aproxima de *4/3* (algoritmo *4/3*-aproximado).

IV.2.3 – O problema do Caixeiro Viajante:

Considere $G=(V,E)$ um grafo completo não-orientado com pesos $c_{ij} \geq 0$ associados a cada aresta $[i,j] \in E$. No problema do caixeiro viajante, deseja-se determinar um ciclo de custo mínimo percorrendo todos os vértice de G uma única vez.

De maneira geral, se $A \subseteq E$ tem-se que:

$$c(A) = \sum_{[i,j] \in A} c_{ij}$$

representa o custo associado ao subconjunto *A* de arestas. Assim, no problema do caixeiro viajante deseja-se determinar um ciclo hamiltoniano *H** tal que $c(H^*) \leq c(H)$, qualquer que seja $H \subseteq E$.

Suponha que os pesos que definem a função de custo *c(.)* satisfaçam a desigualdade triangular, ou seja: $c_{ik} \leq c_{ij} + c_{jk}$, $\forall\, i, j, k \in V$. Este tipo de restrição é bastante comum e ocorre com freqüência em uma grande quantidade de situações práticas como, por exemplo, em problemas de transporte, energia elétrica, telecomunicações entre outros.

Considere *ΔPCV*, a notação utilizada para representação do problema do caixeiro viajante satisfazendo a desigualdade triangular. Observe por exemplo que, se os custos forem euclideanos a desigualdade triangular é satisfeita diretamente.

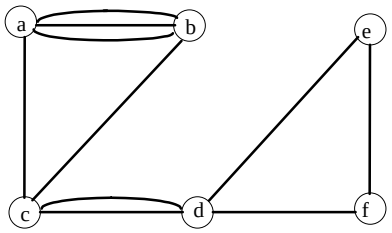
Rosenkrantz et al. [1977], descrevem várias heurísticas 2-aproximadas para o *ΔPCV*. A maior parte destas heurísticas se baseiam na geração inicial de ciclos eurelianos[4] seguida de uma seqüência de "atalhos" (*short-cutting*) visando a determinação de um ciclo hamiltoniano de custo mínimo.

A heurística seguinte, aqui representada por *AGM-CV*, constrói inicialmente uma árvore geradora mínima. Esta árvore é então utilizada na construção de um multigrafo contendo um ciclo eureliano. Em um multigrafo, a cada par de vértices podem estar associados uma ou mais arestas distintas.

O conceito de ciclo eureliano pode ser estendido facilmente para multigrafos, como na definição seguinte:

Definição IV.8: Seja *G* um multigrafo. Um ciclo eureliano em *G* é um caminho que percorre cada vértice de *G* pelo menos uma vez e cada aresta exatamente uma vez. •

A Figura IV.6 ilustra a presença de um ciclo eureliano em um multigrafo com 6 vértices.



Seqüência ---> a, b, a, b, c, d, e, f, d, c, a

Figura IV.6: Geração de um ciclo eureliano

A proposição seguinte caracteriza melhor a classe de multigrafos que contém ciclos eurelianos. A construção de um ciclo eureliano em tempo *O(m)* é conseqüência direta da prova deste teorema. Para maiores detalhes sobre sua demonstração veja Gibbons[1985] (pág. 155-157).

Proposição IV.4: Um multigrafo *G* contém um ciclo eureliano se e somente se *G* for conexo e todos os vértices tiverem grau par[5]. •

Após os conceitos e definições apresentados acima tem-se então o seguinte algoritmo

[4] Um ciclo em um grafo *G* é eureliano se todas as arestas de *G* forem visitadas exatamente uma vez.
[5] O grau de v pertencente ao multigrafo *G* é igual a soma de todas as arestas com extremidade no vértice v.

(desenvolvido por Rosenkrantz et al.[1977]) para "resolução" do *ΔPCV*. A seqüência de vértices H, inicialmente vazia, irá representar o ciclo hamiltoniano gerado pelo algoritmo.

Heurística IV.1: AGM-CV.
Início
1. Leia o grafo original *G* e faça $H \leftarrow \varnothing$.
2. Encontre uma árvore geradora mínima *T** em *G*.
3. Construa um multigrafo com arestas *T'* através da duplicação de cada uma das arestas de *T**.
4. Encontre um ciclo eureliano $\overline{E}$ em *T'* gerando uma seqüência de vértices: $v_1,v_2,...,v_k,v_1$. O vértice inicial v_1 é escolhido arbitrariamente.
5. Encontre um ciclo hamiltoniano H a partir de $\overline{E}$ da seguinte forma: Percorra cada um dos vértices da seqüência $v_1,v_2,...,v_k,v_1$. Se v_i (para $i \neq 1$) ocorre pela primeira vez, então adicione a aresta $[v_i\text{-}1, v_i]$ a ciclo *H*. Caso contrário, v_i é descartado e o próximo vértice da seqüência é pesquisado.
6. Retorna o ciclo hamiltoniano *H*.
fim.

Figura IV.7: Algoritmo AGM -CV

A execução da heurística *AGM-CV* pode ser ilustrada no exemplo da Figura IV.8. Neste caso, cada um dos vértices em *G* estão dispostos em uma "grade" com unidades de área iguais a 1. Logo, a distância euclideana pode ser calculada diretamente[6] (vide Figura IV.8(a)). Uma árvore geradora mínima está representada na Figura IV.8.(b). As arestas duplicadas e a orientação do ciclo eureliano $\overline{E}$, com origem no vértice *a*, estão representados na Figura IV.8(c). Note que a existência de $\overline{E}$ é garantida já que, no novo grafo resultante, todos os vértices possuem grau par (vide Proposição IV.4). A determinação do ciclo hamiltoniano (Figura IV.8(d)) é feita a partir da seqüência *a,b,e,g,e,b,f,d,c,h,c,d,f,b,a* após a eliminação de vértices repetidos em $\overline{E}$, excetuando-se obviamente o primeiro e o último nós (vértice *a*). É fácil ver que a solução gerada pela heurística não é ótima já que as arestas *[e,h]* e *[f,g]* se cruzam mutuamente (Verifique).

Como a etapa 4 do algoritmo tem complexidade *O(m)* (vide Gibbons[1985]), tem-se que o "maior esforço computacional" é obtido no cálculo da Árvore Geradora Mínima (etapa 3). Karger et. al [1995] por exemplo, apresentam um algoritmo randômico de complexidade *O(m)*.

A proposição seguinte garante que a solução gerada pela heurística *AGM-CV* é 2-aproximada.

Proposição IV.5: A heurística *AGM-CV*, quando aplicada ao *ΔPCV*, sempre retorna uma solução 2-aproximada.

Prova: Seja *H** uma solução ótima do problema do caixeiro viajante, e *c(H*)*, o valor da solução ótima correspondente. Se *T** é uma árvore geradora mínima, é fácil ver que $c(T^*) \leq c(T) \leq c(H^*)$, onde *T* é a árvore geradora resultante após a remoção de uma aresta qualquer de *H**. Logo, se cada uma das arestas de T* são duplicadas para a construção do ciclo eureliano $\overline{E}$, tem-se que $c(\overline{E}) = 2.c(T^*) \leq 2.c(H^*)$. Se H é a solução gerada pelo algoritmo *AGM-CV*, da utilização de "atalhos" e da desigualdade triangular conclui-se que $c(H) \leq c(\overline{E})$. Então $c(H) \leq 2.c(H^*)$. •

No exemplo da Figura IV.8(d) o custo da solução heurística *c(H)* é aproximadamente 22,997. Como esta solução é 2-aproximada pode-se afirmar que o valor ótimo *c(H*)* deverá ser superior a 11,498 (aproximadamente) e inferior a 22,997.

[6] Note que outras medidas de distância (ou métricas) podem ser utilizadas. Por definição, qualquer medida de distância deverá satisfazer a desigualdade triangular.

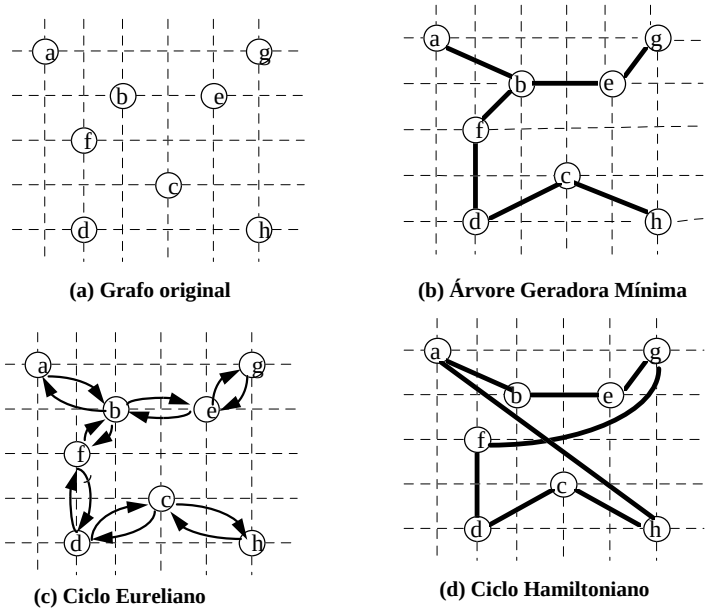


Figura IV.8: Heurística AGM-CV

Uma modificação desta heurística foi proposta por Christofides[1976]. A idéia básica neste caso, é gerar um grafo eureliano a partir de uma árvore geradora mínima evitando a duplicação de arestas, como ocorre por exemplo, na heurística *AGM-CV*. Esta abordagem de Christofides possibilitou a construção de um novo algoritmo 3/2-aproximado para o *ΔPCV*. Antes de sua apresentação entretanto, considere os seguintes resultados auxiliares:

Definição IV.9: O *grau* de v∈ V, representa o somatório de todas as arestas com uma extremidade no vértice v. Se $S \subseteq V$ então *d(S)* representa o somatório do grau de todos os vértices em S[7]. •

Proposição IV.6: Seja *G=(V,E)* um grafo qualquer não-orientado. O número de vértices de grau ímpar em *G* é sempre par.

Prova: Note que cada uma das arestas em *E* contém 2 extremidades, cada uma delas contendo um vértice de grau *um*. Logo, se $|E| = m$ então $d(V) = 2m$ é um número par. Seja $V' \subseteq V$, o conjunto de todos os vértices de grau ímpar em G. Como todos os vértices em $V \backslash V'$ possuem grau par, tem-se que $d(V \backslash V')$ também é par já que a soma de números pares é par. Mas como $d(V) = d(V') + d(V \backslash V')$, conclui-se diretamente que $d(V')$ é número par. Finalmente, $d(V')$ é par se e somente se $|V'|$ for um número par. •

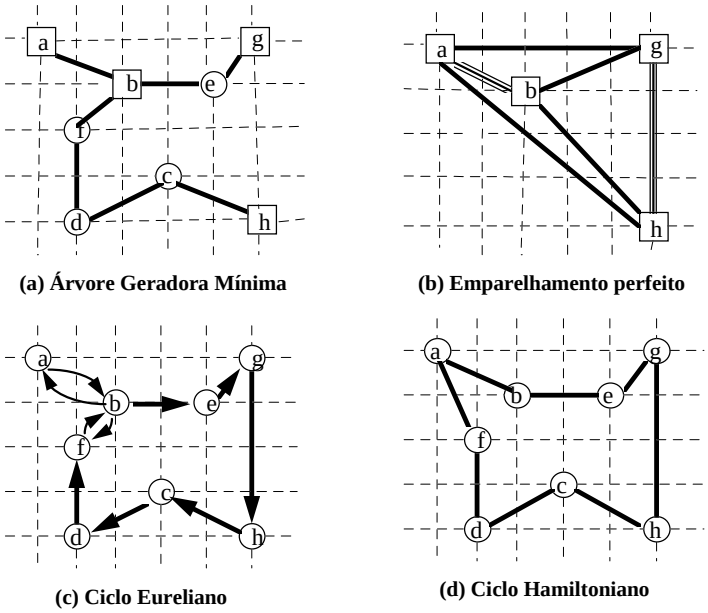
Seja *G=(V,E)* um grafo conexo qualquer. Um subconjunto $M \subseteq E$ define um *emparelhamento perfeito* se, e somente se, nenhum par de arestas de *M* tiver um vértice em comum e todo vértice de *V* for extremidade de alguma aresta em *M*. A proposição seguinte estabelece condições necessárias e suficientes que garantem a presença de um emparelhamento perfeito em grafo completo *G=(V,E)*. Lovász [1975], demonstra este resultado considerando um grafo *G* qualquer (Gibbons[1985]).

Proposição IV.7: Seja *G(V,E)* um grafo completo não-orientado. O conjunto *V* tem cardinalidade par se e somente se *G* contém um emparelhamento perfeito.

Prova: (Exercício) •

[7] Note que, quando $S=\{v\}$, o grau de v, poderá ser representado simplesmente por $d(\{v\})$ ou $d(v)$.

As proposições IV.4, IV.6 e IV.7, estabelecem as bases necessárias para construção do algoritmo de Christofides. Analogamente ao procedimento *AGM-CV*, o algoritmo proposto por Christofides determina inicialmente uma árvore geradora mínima *T** em um grafo *G* completo. Seja *V'*, o conjunto de todos os vértices de grau ímpar nesta árvore, e *G'*, o subgrafo completo com extremidades em V'. Sabe-se da Proposição IV.6 que *V'* tem cardinalidade par. O subgrafo completo *G'* (induzido por *V'*) contém um emparelhamento perfeito (vide Proposição IV.7). Seja *M** o emparelhamento perfeito de custo mínimo. A Proposição IV.4 garante que a união de *M** e *T** irá formar um multigrafo contendo um ciclo eureliano. Analogamente ao algoritmo *AGM-CV*, este novo ciclo eureliano, através da operação de "atalhos", pode ser utilizado na construção de H. A Figura IV.9 ilustra melhor estas etapas. O grafo original *G* é o mesmo da Figura IV.8.(a) visto anteriormente.



(a) Árvore Geradora Mínima **(b) Emparelhamento perfeito**

(c) Ciclo Eureliano **(d) Ciclo Hamiltoniano**

Figura IV.9: Heurística de Christofides

Na Figura IV.9(a) estão representados por quadrados, todos os vértices de grau ímpar em *T** (conjunto *V'*). A Figura IV.9(b) representa o grafo induzido por *V'* (subgrafo completo) e o emparelhamento perfeito de custo mínimo formado pelas arestas do conjunto $M^* = \{[a,b], [g,h]\}$. A união das arestas de *T** com as arestas do emparelhamento M* estão representadas na Figura IV.9(c). O ciclo hamiltoniano *H* resultante da eliminação de vértices repetidos na seqüência *a, b, e, g, h, c, d, f, b, a* está representado na Figura IV.9(d). O algoritmo da Figura IV.11, sintetiza todas estas etapas:

Heurística IV.2: Christofides
Início

1. Leia o grafo original G e faça H←∅.
2. Encontre uma árvore geradora mínima T* em G.
3. Utilizando apenas os vértices de grau ímpar em T* construa um subgrafo completo G'.
4. Determine um emparellhamento perfeito M* de custo mínimo em G' e construa um multigrafo $\overline{G}$ =(V,T*∪M*).
5. Encontre um ciclo eureliano $\overline{E}$ em $\overline{G}$.
6. Utilizando atalhos, encontre um ciclo hamiltoniano H a partir de $\overline{E}$ respeitando-se a ordem estabelecida pelos vértices de $\overline{E}$.
7. Retorna o ciclo hamiltoniano H.

fim.

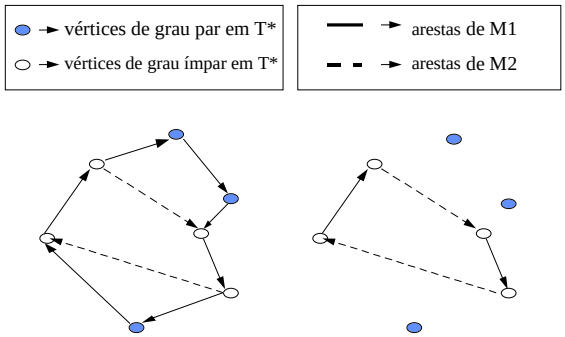
Figura IV.10 Heurística de Christofides para o ΔPCV.

Finalmente, a proposição seguinte, garante que a heurística de Christofides tem razão de performance igual a 3/2.

Proposição IV.8: A heurística de Christofides para o *ΔPCV* tem razão de performance absoluta igual a 3/2.

Prova: A idéia básica da demonstração consiste em se provar que o ciclo eureliano $\overline{E}$, obtido do multigrafo $\overline{G}$ *=(V, T*∪M*)* é tal que, $c(\overline{E}) = c(M^*) + c(T^*) \leq 3/2.c(H^*)$, onde *T** é árvore geradora mínima, *M** é emparelhamento perfeito de custo mínimo em *G'* (subgrafo completo induzido pelos vértices de *T**) e H* é uma solução ótima do *ΔPCV*.

Como *H** é solução ótima, um novo ciclo *H'* pode ser obtido através de "atalhos" eliminando-se de H* todos os vértices de grau par no grafo definido por T*. É óbvio então que $c(H') \leq c(H^*)$. Agora, o ciclo formado pelas arestas de H' define 2 emparelhamentos alternados, por exemplo M_1 e M_2 (como ilustrado no exemplo da Figura IV.11).



(a) Solução ótima H* **(b) Emparelhamentos alternados M_1 e M_2 no ciclo H'**

Figura IV.11: c(H')<c(H*)

Como *M** é emparelhamento perfeito de custo mínimo tem-se: $2c(M^*) \leq c(M_1) + c(M_2) \leq c(H^*)$ ou ainda $c(M^*) \leq (1/2).c(H^*)$. Como $c(T^*) \leq c(H^*)$, somando-se estas duas últimas desigualdades chega-se a: $c(M^*) + c(T^*) \leq 3/2.c(H^*)$.

Para finalizar a demonstração, note que a existência de um ciclo eureliano $\overline{E}$ em $\overline{G}$ é garantida já que todos os vértices de $\overline{G}$ tem grau par (Proposição IV.4). Com a utilização de "atalhos" , pode-se eliminar todos os vértices repetidos presentes na seqüência definida por $\overline{E}$. Assim, a nova solução H obtida pelo algoritmo de Christofides satisfaz $c(H) \leq c(\overline{E})$. Conclui-se portanto que: $c(H) \leq c(\overline{E}) \leq 3/2.c(H^*)$. •

Voltando ao exemplo da Figura IV.9, note que o custo da solução heurística *c(H)* é aproximadamente 18,358. Como esta solução é 3/2-aproximada pode-se afirmar que o valor ótimo *c(H*)* deverá ser superior a 12,237 (aproximadamente) e inferior a 18,358 (verifique!).

Uma desvantagem da heurística de Christofides é o custo computacional necessário para se calcular um emparelhamento perfeito de custo mínimo *M**. Como descrito em Vadya[1988], esta operação é realizada em $O(n^{2,5}log^4n)$ iterações ao passo que, na heurística *AGM-CV* (2-aproximada), a determinação do ciclo eureliano e a determinação da árvore geradora mínima são obtidas em tempo linear (vide Gibbons[1985] e Karger et. al [1995] respectivamente).

Koutsoupias et. al.[1995] desenvolveram um esquema de aproximação polinomial para o problema do caixeiro viajante em grafos planares. Mais recentemente, S. Arora[1996] apresentou um esquema de aproximação polinomial para o problema do caixeiro viajante com distâncias euclideanas em $\Re^2$. Este algoritmo de complexidade $n^{O(1/\varepsilon)}$, representa, ao menos teoricamente, um avanço significativo em relação ao trabalho de Christofides. Seu trabalho consiste basicamente, no

particionamento recursivo do plano de maneira que poucas arestas do circuito hamiltoniano se interceptam com as fronteiras da partição. Na prática, apesar de lento, Arora sugere alternativas de decomposição de instâncias do Caixeiro Viajante tornando-o atrativo para implementações em paralelo.

Seguindo de perto a abordagem utilizada por Christofides[1976] para o *ΔPCV*, Gendreau et al.[1997] apresentam um algoritmo 3/2-aproximado para uma extensão do caixeiro viajante conhecida como caixeiro viajante com pontos de coleta e entrega de mercadorias (*Traveling Salesman Problem with Backhauls*). Nele, deve-se determinar um ciclo hamiltoniano de custo mínimo de maneira que um subconjunto de vértices $L \subset V$ (*linehaul custumers*) seja visitado antes de outro conjunto $B \subset V$ (*backhaul customers*).

IV.2.4 - Resultados Negativos para Algoritmos de Aproximação Relativa

Nesta seção, serão discutidas algumas situações onde a determinação de soluções δ-aproximadas para $\delta > 0$, só será possível se *P=NP*. Uma questão fundamental neste caso é estabelecer quais problemas admitem ou não soluções aproximadas. Infelizmente, a teoria da NP-completude associada a problemas de decisão não fornece nenhum *insight* a este respeito.

A proposição seguinte, demonstrada por Sahni e Gonzalez[1976] mostra que, no Caixeiro Viajante, quando se remove a hipótese da desigualdade triangular a determinação de um algoritmo polinomial δ-aproximado só será possível se *P=NP*.

Proposição IV.9: Se $P \neq NP$ e $\delta \geq 1$ então não existe algoritmo polinomial δ-aproximado para o problema do caixeiro viajante.

Prova: Suponha, por absurdo, que exista um algoritmo polinomial *A* δ-aproximado para o problema do caixeiro viajante. Sem perda de generalidade, através de um arredondamento, pode-se supor δ inteiro, se necessário. Isto será interessante sempre que custos inteiros estiverem associados às arestas do grafo.

A idéia da demonstração consiste em, utilizando-se o algoritmo *A* como ferramenta, resolver o problema do ciclo hamiltoniano em um grafo qualquer *G=(V,E)* não-valorado. Desta forma, estar-se-á resolvendo o problema do ciclo hamiltoniano em tempo polinomial, o que é absurdo, pois, por hipótese, $P \neq NP$. Um problema NP-Completo pertence a P se, e somente se, *P=NP* (Garey& Johnson[1979]).

Seja *G=(V,E)* uma instância qualquer para o problema do ciclo hamiltoniano. Um grafo completo e valorado *G'=(V',E')* pode ser construído fazendo-se:

$$\text{V} = \text{V'}; \qquad E' = \{[i,j] : i, j \in V \ \ e \ \ i \neq j\} \qquad \text{e} \qquad c_{ij} = \begin{cases} 1, & se\ [i,j] \in E, \\ \delta|V| + 1, & caso\ contrario \end{cases}$$

Observe que *G'* é obtido facilmente, através de *G* em tempo polinomial no tamanho de *V* e *E* respectivamente. Contrariamente ao problema do caixeiro viajante com distâncias euclideanas, neste caso, os valores associados às arestas de E' não satisfazem a desigualdade triangular.

Se *G* contém algum ciclo hamiltoniano então o valor da solução ótima para o problema do caixeiro viajante com instância *G'* será igual a $|V|$. Entretanto, pode-se afirmar que, se G não contém nenhum ciclo hamiltoniano então qualquer ciclo hamiltoniano *H* em *G'* deverá conter alguma aresta não pertencente a *E*. Assim,

$$c(H) = \sum_{[i,j] \in H} c_{ij} \geq \delta(|V| + 1) + (|V| - 1) > \delta|V| \qquad \text{onde } H \subset E'.$$

Como A é algoritmo polinomial δ-aproximado, se G contém ciclo hamiltoniano, segue que

$c(H) = c(H^*) = |V|$ onde H é a solução gerada por A e H^* é solução ótima. Isto se deve ao fato de que, nenhuma outra solução H com custo maior que |V| e menor ou igual a $\delta|V|$ pode ser obtida.

Se G não contém ciclo hamiltoniano então, a solução H, gerada por A é tal que $|V| \leq c(H)$. Ainda, da desigualdade (I) conclui-se que $c(H) > \delta|V|$. Logo, o algoritmo polinomial A (aplicado a G') pode ser utilizado, para decidir se o grafo original G contém ou não ciclo hamiltoniano. O que é absurdo, pois, por hipótese, $P \neq NP$. •

IV.3 – ALGORITMOS RANDÔMICOS APROXIMATIVOS

Seja Π um problema de otimização combinatória NP-Árduo e I uma instância qualquer de Π. Neste caso, a adoção de um algoritmo randômico polinomial A pode ser interessante se sua solução for representada por uma variável aleatória $x_A(I)$ com valor esperado idealmente próximo da solução ótima $x^*(I)$.

Nos problemas resolvidos pelo método de Las Vegas (discutidos no Capítulo III), obtinha-se sempre uma solução exata com tempo esperado polinomial. No método de Monte Carlo, uma solução era obtida, com alta probabilidade de acerto e tempo (esperado) também polinomial. Ambos tratavam de problemas da classe P e o principal parâmetro observado era o tempo de processamento. Contrariamente, nesta seção, será dada uma atenção especial à solução de problemas NP-Árduos.

Na determinação de uma solução heurística $x_A(I)$, pode-se, por exemplo, atribuir valores aleatóriamente às variáveis associadas ao problema, satisfazendo obviamente alguma distribuição estatística. Outra possibilidade interessante é a adoção de Programação Linear ou Programação Semidefinida[8] na geração de limitantes inferiores para valor ótimo do problema original. Nas duas situações será possível utilizar algoritmos polinomiais baseados nos métodos de pontos interiores (vide Wright[1997][9] Esta técnica é mais conhecida na literatura como Arredondamento Randômico (*Randomized Rounding*). As soluções relaxadas em ambos os casos, definirão probabilidades a serem utilizadas na etapa randômica. Nos algoritmos randômicos aproximativos, solução esperada e tempo esperado de processamento serão parâmetros importantes a serem observados sendo representados por variáveis aleatórias discretas. Os trabalhos de Srinavasan[1995] e [2001] são uma ótima referência sobre Arredondamento Randômico e Relaxação Linear na solução de problemas combinatórios.

Considere a seguinte definição de algoritmo randômico δ-aproximado:

Definição IV.9: (Algoritmo randômico δ-aproximado)

Seja Π um problema de minimização. Um algoritmo randômico A de complexidade polinomial é δ-aproximado para Π se, e somente se, $E(x_A(I)) \leq \delta.x^*(I)$, onde $\delta \geq 1$. Se Π é um problema de maximização então A é δ-aproximado se e somente se $E(x_A(I)) \geq \delta.x^*(I)$, onde $\delta \leq 1$.•

Em outras palavras, pode-se dizer que, se A é um algoritmo randômico δ-aproximado para um problema de minimização então $Pr[x_A(I) \leq \delta.x^*(I)] \geq 1/2$ onde $\delta \geq 1$. Analogamente, $Pr[x_A(I) \geq \delta.x^*(I)] \geq 1/2$ para $\delta \leq 1$ em problemas de maximização (Vazirani[1997]). Verifique!

IV.3.1 - *O problema do Corte-Máximo em Grafos*

Suponha $G=(V,E)$ um grafo não-orientado com pesos não-negativos w_{ij} associados a cada aresta $[i,j] \in E$. Considere ainda $|V| = n$ e $|E| = m$. No problema do corte-máximo deseja-se encontrar uma partição de $V=\{v_1, v_2,...,v_n\}$ em 2 subconjuntos V_1 e V_2 de vértices de maneira que o somatório dos pesos das arestas com extremidades em V_1 e V_2 respectivamente seja maximizado (vale lembrar que V_1 e V_2 definem uma partição de V se e somente se $V=V_1 \cup V_2$ e $V_1 \cap V_2 = \varnothing$.

[8] Para maiores detalhes sobre a utilização de Programação Semidefinida e Algoritmos Randômicos em Otimização Combinatória consulte Goemans&Willianson[1994],[1995].

[9] Os métodos de pontos interiores foram introduzidos inicialmente por Karmarkar em 1984.

Contrariamente ao problema do corte-mínimo (pertencente à classe *P*) o problema do Corte Máximo é NP-Árduo e é conhecido por sua aplicação em áreas distintas como, p. ex., física estatística e Circuitos VLSI (Barahona et al [1988]), *Network Design* (Barahona [1996]) entre outras. Este problema continua NP-Árduo mesmo quando $w_{ij}=1, \ \forall \ i,j \in V$ (Garey&Johnson[1979]). Orlova e Dorfman [1972] e Hadlock[1975], discutem classes especiais de grafos resolvidas em tempo polinomial, como por exemplo, os grafos planares. Maiores detalhes sobre este problema podem ser encontrados em Poljak e Z. Tuza [1995].

Sahni e Gonzalez[1976] apresentaram um algoritmo randômico *1/2*-aproximado considerando pesos unitários associados a cada aresta $[i,j] \in E$ (vide Figura IV.13). A idéia neste caso, consiste simplesmente na escolha aleatória de elementos de V_1 e V_2 respectivamente.

Algoritmo IV.3: Corte-Máximo Randômico - CMR
Início

$V_1 \leftarrow V_2 \leftarrow \varnothing$;
Para *i=1* **até** *n* **faça**
x ← jogue uma moeda;
Se *x = cara* **então** insira v_i em V_1;
Se *x = coroa* **então** insira v_i em V_2;
fim;
Retorne V_1 e V_2;
fim.

Figura IV.12: Corte-Máximo Randômico

Como exemplo considere o grafo com 5 vértices e 7 arestas representado na figura seguinte:

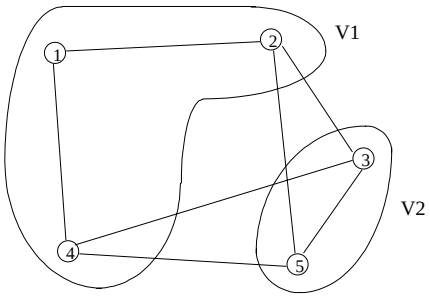


Figura IV.13: Corte-máximo randômico

Seja *X* uma variável aleatória que representa o número de arestas pertencentes ao corte(V_1,V_2). Para mostrar que o algoritmo *CMR* é randômico 0,5-aproximado basta provar que $E(X) \geq 1/2.x^*(I)$, onde $x^*(I)$ representa o número de arestas pertencentes ao corte máximo. Para demonstrar esse fato, basta notar que *m* (número de arestas de *G*) define um limite superior para o valor da solução ótima, ou seja, $x^*(I) \leq m$. Mostrando que $E(X) \geq 1/2.m$, prova-se então o resultado.

Considere $I(e_i)$ para $i \in \{1,.., m\}$, uma variável aleatória que indica se a aresta e_i de *E* pertence ou não ao corte máximo. Desta forma, $I(e_i) = 0$ ou *1*. Para determinar $Pr(I(e_i)=1)$ e $Pr(I(e_i)=0)$ respectivamente monta-se um experimento onde uma moeda equilibrada é jogada para cada uma das extremidades de $e_i=(u,v)$. Segue então que $Pr(I(e_i)=0) = 1/2$ (duas *caras* ou duas *coroas* ocorrem simultaneamente) ou $Pr(I(e_i)=1) = 1/2$ (*cara* e *coroa* ocorrem alternadamente). Assim, da definição do valor esperado tem-se:

$$E(I(e_i)) = 1.\Pr(I(e_i) = 1) + 0.\Pr(I(e_i) = 0) = 1/2$$

Como $X = I(e_1) + I(e_2) + ... + I(e_m)$ segue, da linearidade do valor esperado (Capítulo II) que:

$$E(X) = E\left(\sum_{i=1}^{m} I(e_i)\right) = \sum_{i=1}^{m} E\big(I(e_i)\big) = \frac{m}{2} \geq \frac{x^*(I)}{2}.$$

Equivalentemente, pode-se dizer que, em uma iteração do Algoritmo CMR tem-se *Pr(X>0,5.x*(I))* (probabilidade de sucesso). Assim, a garantia de uma solução *0,5*-aproximada pode ser incrementada repetindo-se o processo um número pré-determinado de vezes. Por exemplo, se *k=10* então a probabilidade de falha no algoritmo randômico será menor ou igual a 1/1024!

Utilizando técnicas mais sofisticadas, Goemans e Williamson apresentaram em 1994 um algoritmo 0,878-aproximado para o problema do Corte-Máximo em Grafos. Neste algoritmo, eles utilizaram uma relaxação baseada em Programação Semidefinida seguida de um arredondamento randômico (*Randomized Rounding*) na determinação de uma solução viável. Esta técnica inovadora vem sendo utilizada, a partir de então, como método aproximativo para outros problemas NP-Árduos. Em 2000, Goemans e Williamson foram laureados com o prêmio Fulkerson em reconhecimento a este trabalho[10]. Outros exemplos de aplicação desta técnica podem ser encontrados em Motwani et. al [1997] e Anderson[2000].

IV.3.2 - *O problema MAX-SAT (Maximum Satisfiability problem)*

Considere uma expressão booleana *B* na forma normal conjuntiva, ou seja, *B* será formada por uma conjunção de cláusulas e uma disjunção de literais. No problema *MAX-SAT* deseja-se determinar o maior número possível de cláusulas que sejam simultaneamente verdadeiras. Assume-se, sem perda de generalidade, que nenhuma cláusula contenha um literal *x* juntamente com seu complemento $\bar{x}$, caso contrário, qualquer atribuição (*V* ou *F*) a este literal tornaria a cláusula correspondente verdadeira. O problema MAX-SAT é NP-Árduo, já que o problema de decisão SAT associado é NP-Completo (Garey&Johnson[1979]).

A expressão booleana *B* com cláusulas C_j, para *j=1,2,...,m* pode ser representada sucintamente por:

$$B = \bigwedge_{j=1}^{m} C_j \ , \quad \text{onde} \quad |C_j| = k_j, \quad para \ \ j = 1, .., m.$$

Exemplo IV.3: Para ilustrar a definição acima considere a seguinte expressão booleana na forma normal conjuntiva:

$$B = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_2) \wedge (x_1)$$

É fácil ver, por simples inspeção, que o número máximo de cláusulas em *B* simultaneamente verdadeiras é igual a 3. Na versão decisão associada (*Satisfiability Problem-SAT*) deseja-se determinar simplesmente se *B* é *falsa* ou *verdadeira* para alguma atribuição a seus literais. É fácil ver neste caso que *B* será sempre *falsa* (verifique). •

Seja *x*(B)*, o número máximo de cláusulas simultaneamente verdadeiras e $x_A(B)$ o número de cláusulas verdadeiras obtidas por um algoritmo *A* qualquer após a atribuição de valores *V* ou *F* (*verdadeiro* ou *falso*) aos literais de *B*.

Um algoritmo randômico bastante simples para o *MAX-SAT* (representado aqui por A_1) pode ser obtido atribuindo-se *V* ou F aos literais de *B* com probabilidade *1/2*. Suponha neste caso que Z_j para $j \in \{1,..,m\}$, seja variável aleatória associada à cláusula C_j. Assim, $Z_j = 1$ se, e somente se, C_j é

[10] A concessão do Prêmio Fulkerson ocorre, no mínimo, após 6 anos da data de publicação do trabalho. A justificativa para este fato é que, um trabalho relevante, necessita de um prazo que seja suficiente para sua divulgação e reconhecimento na comunidade científica.

verdadeira, e $Z_j = 0$ caso contrário. É fácil ver então que:

$$\Pr(Z_j = 0) = 1/2^{k_j}, \qquad (1)$$
$$\Pr(Z_j = 1) = 1 - 1/2^{k_j}$$

Seja $x_{A1}(B)=Z_1+Z_2+...+Z_m$ o número total de clásulas verdadeiras obtidas por A_1. Segue, da linearidade do valor esperado que:

$$E(x_{A_1}(B)) = \sum_{j=1}^{m} E(Z_j) \qquad (2)$$

onde $E(Z_j)=1.Pr(Z_j=1)+0.Pr(Z_j=0)$. Fazendo-se $k = min\{k_j \mid j=1,...,m\}$ e substituindo (1) em (2) chega-se a:

$$E(x_{A_1}(B)) = \sum_{j=1}^{m} E(Z_j) = \sum_{j=1}^{m}\left(1-\frac{1}{2^{k_j}}\right) \geq m.\left(1-\frac{1}{2^{k}}\right)$$

Como $m \geq x^*(B)$ e $k \geq 1$ conclui-se que $E(x_{A1}(B)) \geq (1/2).x^*(B)$ (solução randômica 0,5-aproximada). Observe por exemplo que, se cada uma das m cláusulas contém pelo menos 2 literais então a solução obtida é 0,75-aproximada. Pode-se, portanto, constatar que a solução do problema é dificultada em função da presença de cláusulas com um único literal.

Uma outra abordagem bastante interessante, utiliza as coordenadas de uma solução de um problema de Programação Linear como probabilidades de escolha para cada um dos literais de B (para maiores detalhes sobre Programação Linear vide Bazarra et al.[1990]). Suponha que $z_j \in \{0,1\}$ seja variável binária indicando respectivamente se a cláusula C_j é falsa ou verdadeira. Analogamente, seja $y_i \in \{0,1\}$ variável binária onde $y_i=1$ se, e somente se, o literal x_i associado for *verdadeiro* e $y_i=0$, caso contrário. Considere ainda C_j^+ o conjunto de todos os índices i associados a literais x_i de C_j não representados em sua forma complementar e C_j^- , o conjunto de todos os índices i associados a literais $\bar{x}_i$ de C_j (em sua forma complementar)[11]. Suponha que o número total de literais x_i (na sua forma complementar ou não), bem como o número de variáveis y_i associadas seja igual a n. O problema *MAX-SAT* pode então ser formulado como um problema de Programação Linear Inteira (PLI) da seguinte forma:

$$\max \sum_{j=1}^{m} z_j$$

$$sujeitoa: \begin{cases} \sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq z_j \;\forall\, j \in \{1,..,m\} \\ y_i, z_j \in \{0,1\},\ p/\ i = 1,...,n\ \ e\ \ j = 1,..,m \end{cases}$$

Note que, se uma das variáveis y_i (com índices em C_j^-) for 0 ou uma das variáveis y_i (com índices em C_j^+) for 1 para que a restrição correspondente à cláusula C_j será satisfeita. Pode-se fazer então $z_j=1$.

Um modelo de Programação Linear (PL) é obtido após a substituição das restrições de integralidade $y_i, z_j \in \{0,1\}$, por $y_i, z_j \in [0,1]$, $\forall\, i,j$. Sejam $\hat{y}_i$ e $\hat{z}_j, \forall\, i,j$, os valores obtidos (solução ótima) após a resolução desta relaxação linear. É fácil ver neste caso que $\hat{z}_1+...+\hat{z}_m$ define um limite superior para o valor da solução ótima $x^*(B)$.

Um novo algoritmo randômico (representado aqui por A_2) pode ser obtido fazendo-se $Pr(y_i=1)=\hat{y}_i$ e $Pr(y_i=0)=1-\hat{y}_i$, para $i=1,2,..,n$. A linha 1 retorna um vetor solução $\hat{y} \in [0,1]^n$ do problema de Programação Linear (PL). A linha 2, utiliza o vetor $\hat{y}$ na geração de uma solução inteira

[11] Por vezes, utilizaremos um abuso de linguagem dizendo simplesmente que a cláusula C_j será representada equivalentemente por seu conjunto de índices $C_j=C_j^+ \cup C_j^-$.

$y \in \{0,1\}^n$. Finalmente, na linha 3, determina-se um vetor inteiro $z \in \{0,1\}^m$ com custo $x_{A2}(B)$ associado. Note que $x_{A2}(B)$ representa exatamente o número de cláusulas simultaneamente verdadeiras associadas à solução $y \in \{0,1\}^n$.

Procedimento IV.4: MAX-SAT Randômico (A2)
Início

1. Resolve *PL* retornando $\hat{y}$;
2. **Para** $i = 1$ **até** n **faça**
   - Joga moeda viciada c/ $Pr(cara) = \hat{y}_i$;
   - **Se** moeda = <cara> **então** $y_i = 1$
   - **senão** $y_i = 0$;

   fim;
3. **Para** $j = 1$ **até** m **faça** {*Calcula* $x_{A2}(B) = z_1 + ... + z_n$}
   - $z_j \leftarrow 0$; $i \leftarrow 1$;
   - **Enquanto** $i \leq n$ **faça**
     Se ($y_i = 1$ *e* $i \in C_j^+$) ou ($y_i = 0$ *e* $i \in C_j^-$) **então**
     $z_j \leftarrow 1$;
     $i \leftarrow n+1$;
     senão $i \leftarrow i+1$;
     fim;

   fim;
4. Utilizando o vetor inteiro $z \in \{0,1\}^n$, calcula $x_{A2}(B)$;

Fim.

Figura IV.14: MAX-SAT Randômico

Como discutido anteriormente, para se determinar a qualidade da solução heurística $x_{A2}(B)$, deve-se calcular $E(x_{A2}(B)) = E(z_1) + ... + E(z_n)$. Como $E(z_j) = Pr(z_j = 1)$, p/ $j = 1,2,...,m$, tem-se que:

$$\Pr(z_j = 0) = \prod_{i \in C_j} (1 - \hat{y}_i)$$
$$\Pr(z_j = 1) = 1 - \prod_{i \in C_j} (1 - \hat{y}_i) \qquad (3)$$

Da relaxação linear (PL) tem-se que:

$$\sum_{i \in C_j^+} \hat{y}_i + \sum_{i \in C_j^-} (1 - \hat{y}_i) \geq \hat{z}_j \quad \forall j \in \{1,..,m\}.$$

Além disso, como $|C_j| = k_j$, $Pr(z_j = 1)$ pode ser limitado inferiormente fazendo-se $\hat{y}_i = \hat{z}_j / k_j$ (verifique). Assim, de (3), tem-se que $Pr(z_j = 1) \geq 1 - (1 - \hat{z}_j / k_j)^{kj}$. A função $f(\hat{z}_j) = 1 - (1 - \hat{z}_j / k_j)^{kj}$ é côncava no intervalo $[0,1]$ já que $f'(\hat{z}_j) \geq 0$, $f''(\hat{z}_j) \leq 0$ (verifique). Como $f(0) = 0$ e $f(1) = 1 - (1 - 1/k_j)^{k_j}$ tem-se:

$$1 - (1 - \hat{z}_j / k_j)^{k_j} \geq (1 - (1 - 1/k_j)^{k_j}) \hat{z}_j \geq (1 - 1/e) \hat{z}_j .$$

A Figura IV.15 ilustra melhor esta operação. Note que a função côncava $f(\hat{z}_j)$ é limitada inferiormente pela função linear $(1 - (1 - 1/k_j)^{kj}) \hat{z}_j$ no intervalo $[0,1]$. Além disso, pode-se mostrar facilmente que $(1 - 1/k)^k < e^{-1}$, $\forall\ k > 0$. Basta que $(1 + x) < e^x$, $\forall\ x$ (aproximação linear de e^x). Assim, fazendo-se $x = -1/k$ p/ $k > 0$ conclui-se que $(1 - 1/k) < e^{-1/k}$, e portanto, $(1 - 1/k)^k < e^{-1}$. Logo, $Pr(z_j = 1) \geq (1 - 1/e) \hat{z}_j = 0{,}632 . \hat{z}_j$.

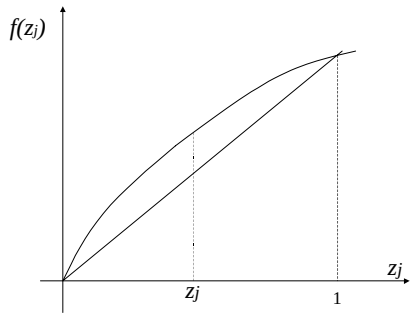


Figura IV.15: Concavidade de *f(zj)*

Da linearidade do valor esperado tem-se que:

$$E(x_{A_2}(B)) = \sum_{j=1}^{m} E(z_j)$$

Substituindo $Pr(z_j=1)$ em $E(z_j)=Pr(z_j=1)$ e lembrando que $\hat{z}_1+...+\hat{z}_m \geq x^*(B)$ conclui-se que $E(x_{A2}(B)) \geq (1\text{-}1/e).x^*(B)$. Este resultado é superior àquele obtido pelo algoritmo A_1 quando aplicado a expressões contendo cláusulas com um único literal.

A Figura IV.16 mostra a dependência entre os algoritmos A_1 e A_2 quando aplicados a expressões com cláusulas contendo no mínimo *k* literais.

k	A1	A2
1	0,5	1,0
2	0,75	0,75
3	0,875	0,704
4	0,938	0,684
5	0,969	0,672

Figura IV.16: Performance de A1 e A2 como função de k

Note que, apesar da melhoria introduzida, o algoritmo A_2 piora quando o número mínimo de cláusulas é aumentado. Este fato entretanto, sugere a criação de um terceiro algoritmo randômico A_3 baseado simplesmente na junção de A_1 e A_2 respectivamente. Sejam n_1 e n_2, o número total de cláusulas satisfeitas por A_1 e A_2 respectivamente. Tem-se então o seguinte resultado:

Proposição IV.10: $\max\{n_1, n_2\} \geq \frac{3}{4}\sum_{j=1}^{m}\hat{z}_j \geq \frac{3}{4}.x^*(B).$

Prova: Seja *B* uma expressão booleana contendo cláusulas com, no mínimo, *k* literais ($k \geq 1$). Executando-se os algoritmos A_1 e A_2 para a entrada *B* conclui-se que :

$$n_1 = E(x_{A_1}(B)) \geq \alpha_k.x^*(B) \qquad \text{e} \qquad n_2 = E(x_{A_2}(B)) \geq \beta_k.x^*(B)$$

onde $\alpha_k = \left(1-\frac{1}{2^k}\right)$ e $\beta_k = \left(1-\left(1-\frac{1}{k}\right)^k\right)$. Segue então que:

$$n_1 + n_2 \geq (\alpha_k + \beta_k).x^*(B) = \left(2-\frac{1}{2^k}-\left(1-\frac{1}{k}\right)^k\right).x^*(B) \geq \left(2-\left(\frac{1}{2^k}+\frac{1}{e}\right)\right).x^*(B) \geq \left(2-\frac{1}{2}\right).x^*(B)$$

Assim conclui-se que: $2.\max\{n_1, n_2\} \geq n_1 + n_2 \geq (3/2).x^*(B)$ e o resultado segue imediatamente. •

Observe que a execução conjunta de A_1 e A_2 respectivamente definem um algoritmo randômico 0,75-aproximado. Isto ocorre mesmo para cláusulas contendo apenas um único literal!

O problema *MAX-SAT* e o problema do corte máximo em grafos, vistos acima, sempre retornam soluções viáveis, independentemente do resultado gerado pelo algoritmo randômico. Pode ocorrer, entretanto, que as soluções obtidas não satisfaçam sempre as restrições de determinado problema sendo, portanto, inviáveis. Exemplos dessa situação podem ser encontrados em Bertsimas& Vohra[1998] e Srinivasan [1995]. Eles apresentam algoritmos randômicos para problemas conhecidos na literatura como *Packing* e *Covering*.

IV.3.3 – *O problema Geral de Recobrimento (General Covering Problem)*

Uma grande quantidade de problemas de Otimização Combinatória pode ser modelada como problemas de Recobrimento, como por exemplo, o problema de Recobrimento de Conjuntos (*Set Covering Problem*), Localização de Facilidades (*Facility Location*), *Network Design* entre outros (Bertsimas&Vohra[1998]).

De maneira geral, o problema de Recobrimento pode ser representado matematicamente por:

$$x^*(I) = Min\ c^T.x$$
$$s.t: \begin{cases} Ax \geq b \\ x \in Z_+^n \end{cases}$$

onde I representa a instância formada pelas matrizes A, b e c sendo $A \in Z_+^{mxn}$, $c, x \in Z_+^{nx1}$ e $b \in Z_+^{mx1}$. A relaxação linear deste problema (modelo de Programação Linear-PL) é obtida substituindo-se $x \in Z_+^{nx1}$ por $x \in R_+^{nx1}$.

Nesta seção será discutida a aplicação da técnica de Arredondamento Randômico na resolução do problema de Recobrimento de Conjuntos (*Set Covering Problem*), caso particular do problema geral de Recobrimento. Diferentemente dos problemas MAX-SAT e Corte-Máximo em Grafos onde soluções inteiras geradas randomicamente eram sempre viáveis, no problema de Recobrimento de Conjuntos isto pode não ocorrer. É importante que se tenha, nestes casos, um cuidado especial com a geração indesejada de soluções inviáveis. Como discutido adiante, bastará que soluções viáveis geradas aleatóriamente por este processo tenham probabilidade de sucesso estritamente positiva.

A técnica de Arredondamento Randômico pode ser resumida através do seguinte procedimento genérico:

Procedimento IV.5: Arredondamento Randômico: Prob. Geral de Recobrimento-PRG
Início

1. Resolve Relaxação Linear (PL) obtendo solução $y \in R_+^{nx1}$
2. Gera solução inteira $x \in Z_+^{nx1}$ através de uma função de arredondamento $Pr(x_j = \lfloor y_j \rfloor + 1) = f(y_j)$ e $Pr(x_j = \lfloor y_j \rfloor) = 1 - f(y_j)$, para $j=1,...,n$;
3. Encontra solução determinística *derandomizada*;

fim.

Figura IV.17: Arredondamento Randômico Genérico

A função de arredondamento *f(.)* presente no item 2 pode ser linear (vide *MAX-SAT* seção anterior) ou não-linear. Srinavasan[1995] utiliza arredondamento linear para o PGR fazendo $Pr(x_j = \lfloor y_j' \rfloor + 1) = y_j' - \lfloor y_j' \rfloor$ e $Pr(x_j = \lfloor y_j' \rfloor) = 1 - y_j' + \lfloor y_j' \rfloor$, onde $j=1,...,n$ e $y_j' = \propto y_j$ para $\propto > 0$ escolhido convenientemente. Uma solução *derandomizada* é obtida através do método dos Estimadores Pessimistas. Maiores detalhes sobre a utilização de arredondamento randômico aplicado aos problemas de Recobrimento e Empacotamento (*Covering and Packing Problems*) podem ser encontrados em Srinavasan[1995].

Um arredondamento não-linear será adotado na resolução do problema de Recobrimento de

Conjuntos (Seção IV.3.3.1). A construção probabilística de uma solução determinística (item 3) será discutida mais atentamente na seção IV.4.

IV.3.3.1 – *O problema de Recobrimento de Conjuntos (Set Covering Problem)*

No Problema de Recobrimento de Conjuntos - PRC tem-se um conjunto de n objetos $V=\{1,2,..,n\}$[12] cada um com um custo associado $c_j \geq 0$ (para $j=1,..,n$) e uma família S_i de m subconjuntos de V. O objetivo neste problema será determinar um subconjunto $S \subseteq V$ de maneira que $|S \cap S_i| \geq 1$, $\forall\ i \in \{1,2,...,m\}$ e de maneira que a função objetivo $\sum_{j \in S} c_j$ seja minimizada. Este problema pode ser formalizado através do seguinte modelo de Programação Linear Inteira - PLI:

$$Z^* = Min \sum_{j=1}^{n} c_j x_j$$

$$s.t: \begin{cases} \sum_{j \in S_i} x_j \geq 1, & i = 1,2,...,m \\ x_j \in \{0,1\}, & j = 1,2,...,n \end{cases}$$

Este problema possui inúmeras aplicações práticas importantes, entre elas pode-se destacar a construção de cadeias de DNA. Nesta aplicação, tem-se normalmente uma enorme quantidade de fragmentos de DNA (seqüências de nucleotídeos formadas pelas letras *A, C, G* e *T* respectivamente) e que devem ser utilizadas convenientemente na geração de cadeias maiores (Vazirani [1997]). O objetivo neste caso será facilitar a construção de grandes seqüências através da eliminação de fragmentos redundantes. Em nosso modelo de programação linear inteira, cada fragmento pode ser interpretado como um subconjunto $S_i \subseteq V$ onde V representa todas as possíveis subdivisões destes fragmentos (vértices). Um vértice $v \in V$, poderá pertencer a diversos fragmentos distintos simultaneamente.

Seja $y=(y_1,...,y_n)$ uma solução da relaxação linear (PL) associada ao PRC. Antes de apresentar um arredondamento não-linear para este problema considere o seguinte arredondamento linear onde se tem:

$$\Pr(x_j = 1) = y_j;$$
$$\Pr(x_j = 0) = 1 - y_j; \quad para \quad j = 1,2,...,n$$

Antes de discutir a qualidade da solução gerada por este procedimento (aproximação de Z^*), deve-se discutir se a solução inteira $x \in \{0,1\}^n$ gerada randomicamente, é de fato viável. Em outras palavras, deseja-se determinar a probabilidade de falha (inviabilidade). Para isso, considere, sem perda de generalidade, um subconjunto $S_i=\{i_1,...,i_k\} \subseteq V$ com k elementos. Além disso, considere um evento E_i que ocorre se, e somente se, o subconjunto S_i não for coberto, ou seja, se nenhum elemento de S_i fizer parte da solução do problema de Recobrimento de Conjuntos. Obviamente, a ocorrência do evento complementar E_i^c irá indicar que algum elemento de S_i foi selecionado. Observe que uma solução inteira (gerada randomicamente) será viável apenas se todos os eventos E_i^c para $i=1,2,..,m$ ocorrerem simultaneamente.

Considere um elemento j pertencente a $S_i=\{i_1,...,i_k\} \subseteq V$ para algum $i \in \{1,..,m\}$. Do arredondamento linear acima tem-se que $Pr(x_j=0)=1-y_j$. Além disso, da solução do problema relaxado (PL) tem-se que $y_{i1}+...+y_{ik} \geq 1$. Assim, fazendo $y_j=1/k$, $\forall\ j \in S_i$ pode-se limitar $Pr(E_i)$ superiormente da seguinte forma:

[12] Para simplificar a notação, adotamos aqui um pequeno abuso de linguagem representando os elementos de V por seu conjunto de índices associado.

$$\Pr(E_i) = \prod_{j\in\{i_1,\ldots,i_k\}}(1-y_j) \le \left(1-\frac{1}{k}\right)^k < \frac{1}{e}$$

Um solução inviável é gerada apenas se pelo menos um dos eventos E_i (para $i=1,..,m$) ocorrer. Assim, a probabilidade de falha (inviabilidade) será dada por:

$$\Pr\left(\bigcup_{i=1}^{m} E_j\right) \le \sum_{i=1}^{m} \Pr(E_i) \le \frac{m}{e}$$

Finalmente, conclui-se que a probabilidade de sucesso (solução viável) será igual a:

$$\Pr\left(\bigcap_{i=1}^{m} E_i^c\right) = 1-\Pr\left(\bigcup_{i=1}^{m} E_i\right) \ge 1-\sum_{i=1}^{m}\Pr(E_i) \ge 1-\frac{m}{e}$$

Como a função de probabilidade assume valores no intervalo *[0,1]*, pode-se dizer que a probabilidade de sucesso será maior ou igual a *zero (*já que *1-m/e* é negativo para $m\ge 3$). Esta situação indesejável pode ser contornada utilizando-se um arredondamento não-linear (Bertsimas& Vohra[1998]) como descrito a seguir:

$$\Pr(x_j = 0) = (1-y_j)^t;$$
$$\Pr(x_j = 1) = 1-(1-y_j)^t; \quad para \quad j = 1,2,...,n \;\; e \;\; t \in Z^+$$

É fácil ver neste caso que $Pr(x_j=0)=(1-y_j)^t$ será menor ao igual ao arredondamento linear onde se tinha $t=1$. Assim, se $1-y_j$ representa a probabilidade de *cara* em uma moeda equilibrada, serão necessárias t repetições (*t caras* seguidas) para atribuição de *zero* à variavel x_j. Caso contrário, se alguma *coroa* ocorrer em k tentativas faz-se $x_j=1$.

Considere novamente $S_i=\{i_1,...,i_k\}$ um subconjunto qualquer de V com k elementos. Analogamente à análise de viabilidade realizada no arredondamento linear tem-se que:

$$\Pr(E_i) = \prod_{j\in\{i_1,\ldots,i_k\}}(1-y_j)^t \qquad onde \;\; i \in \{1,2,..,m\}$$

Como $y_{i1}+...+y_{ik}\ge 1$ pode-se fazer $y_j=1/k \;\; \forall \; j\in S_i=\{i_1,...,i_k\}$. Assim: $Pr(E_i) \le (1-1/k)^{kt} \le (1/e)^t$. Se $t=O(logm)$ repetições forem realizadas então: $Pr(E_i) \le (1/e)^{clogm} \le 1/2m$ (verifique). Finalmente, a probabilidade de fracasso (solução inviável) será igual a:

$$\Pr\left(\bigcup_{i=1}^{m} E_i\right) \le \frac{m}{2m} = \frac{1}{2}$$

Portanto, a probabilidade de sucesso (viabilidade) será igual a:

$$\Pr\left(\bigcap_{i=1}^{m} E_i^c\right) = 1-\Pr\left(\bigcup_{i=1}^{m} E_i\right) \ge \frac{1}{2}$$

Uma vez provada que uma solução viável é gerada com probabilidade de sucesso maior ou igual a 50% deve-se responder agora à seguinte questão: qual a qualidade da solução (aproximação de Z^*) gerada pelo procedimento randômico? Antes de tratar deste assunto considere o seguinte resultado auxiliar conhecido na literatura como desigualdade de Bernoulli:

Proposição IV.11: (Desigualdade de Bernoulli)
Se $x \geq -1$ então $(1+x)^t \geq 1+tx, \ \forall\, t \in N$.

Prova: Este resultado pode ser provado por indução em t. Para $t=1$ a tese se verifica imediatamente (trivial). Suponha que a tese seja verdadeira para um t inteiro positivo qualquer. Como $x+1 \geq 0$ tem-se para $t+1$ que: $(1+x)^t(1+x) \geq (1+tx).(1+x) = 1 + tx + x + tx^2 = 1 + (t+1)x + tx^2 \geq 1 + (t+1)x$. •

Seja A, um algoritmo randômico para o PRC que utiliza o arredondamento não-linear descrito acima. Além disso, seja $x \in \{0,1\}^n$ sua solução e $x_A(I)$ sua imagem associada. Assim, como $x_A(I) = c_1x_1 + ... + c_nx_n$ (função objetivo), segue, da linearidade do valor esperado que $E(x_A(I)) = c_1E(x_1) + ... + c_nE(x_n)$. Mas $E(x_j) = Pr(x_j=1) = 1-(1-y_j)^t$ onde $y_j \in [0,1]$ (note neste caso que $x = -y_j \geq -1$). Da desigualdade de Bernoulli (Proposição IV.11) tem-se que: $1-(1-y_j)^t \leq ty_j, \ \forall\ j=1,2,...,n$. Logo:

$$E(x_A(I)) = \sum_{j=1}^{n} c_j \Pr(x_j = 1) = \sum_{j=1}^{n} c_j \left(1 - (1 - y_j)^t\right) \leq \sum_{j=1}^{n} c_j (ty_j) = t \sum_{j=1}^{n} c_j y_j$$

Como $\sum_j c_j y_j$ define um limite inferior para Z^* e $t = O(\log m)$, o algoritmo randômico A com solução $x_A(I)$ será $O(\log m)$-aproximado (pois $E(x_A(I)) \leq O(\log m).Z^*$).

Se $0 < \varepsilon \leq 1$, pode-se obter uma solução randômica $O(\log m)$-aproximada com probabilidade de sucesso superior a $1-\varepsilon$. O procedimento seguinte descreve sucintamente as principais etapas deste processo. A variável x_A representa melhor solução viável obtida a cada iteração com custo Z_A associado.

Procedimento IV.6: Monte Carlo - PRC
Dados: A, b, c e $\varepsilon > 0$.
Início
    $k \leftarrow 1$;
    $x_A(I) \leftarrow \infty$; {salva valor da melhor solução randômica}
    Resolve PL retornando solução relaxada $y \in [0,1]^n$;
    Repita
        Utilizando y e arredondamento não-linear retorna $x_A \in \{0,1\}^n$ e Z_A;
        Se (x_A é viável) e ($Z_A < x_A(I)$) **então**
            $x_A(I) \leftarrow Z_A$;
            $x \leftarrow x_A$;
        fim.
        $k \leftarrow k+1$;
    Até que $k = \lceil \log(1/\varepsilon) \rceil$;
    Retorna $x \in \{0,1\}^n$ e $x_A(I)$;
fim.

Figura IV.18: Monte Carlo – PRC.

Para calcular a complexidade do procedimento acima note inicialmente que $O(n^3L)$ representa o tempo necessário (complexidade) para solução do problema de programação linear. A constante L neste caso representa o número total de *bits* utilizado na entrada de dados (para maiores detalhes vide Wright[1997]). O arredondamento não-linear terá complexidade igual a $O(n \log m)$ ao passo que, determinar se x é ou não viável, tem complexidade de pior caso $O(nm)$. É fácil ver que o procedimento terá complexidade total igual a $O(n^3L + nm.\log(1/\varepsilon))$. Neste caso a solução gerada $x_A(I)$ será $O(\log m)$-aproximada e viável com probabilidade maior ou igual a $1-\varepsilon$. Outra alternativa interessante, diferente do Procedimento IV.6 é construir uma solução determinística utilizando-se métodos probabilísticos (*derandomização*). Esta abordagem será discutida mais adiante na seção IV.3.4.2.

Será que é possível reduzir ainda mais a aproximação do valor ótimo Z^*? Na verdade, Lund&

Yannakakis[1994] provaram que isto só será possível se *P=NP*! Apesar disso, como discutido a seguir, pode-se conseguir para instâncias particulares do PRC, uma maior aproximação do valor ótimo *Z**.

Observe que o evento $F = E_1^c \cap ... \cap E_m^c$ representa uma solução viável e ocorre com probabilidade estritamente positiva ($Pr(F) > 0$). Na verdade, apenas uma iteração do procedimento IV.6 já garante $Pr(F)>1/2$. Isto foi possível fazendo-se $t=O(logm)$ no arredondamento não-linear.

Como discutido em Bertsimas&Vohra[1998], um outro arredondamento por ser obtido fazendo-se $x_j=1$, se $y_j \geq 1/logm$. Caso contrário, se $y_j<1/logm$ faz-se $x_j=1$ com probabilidade $y_j/logm$. Analogamente ao arredondamento não-linear aqui apresentado este procedimento é também *O(logm)*-aproximado.

Considere agora o seguinte problema: ao "enfraquecer" a probabilidade de sucesso tornando-a menor, mas estritamente positiva, será possível uma maior aproximação da solução ótima *Z**? Neste caso, deseja-se calcular $E(x_A(I)|F)$ garantindo apenas que $Pr(F) >0$. A Proposição IV.13 provada por Bertsimas&Vohra[1998] resolve esta questão. Antes de tratá-la entretanto, considere a definição e proposição preliminares:

Definição IV.10: (Família monótona crescente)

Uma família Ψ de subconjuntos de um conjunto N qualquer será *monótona crescente* se, e somente se, $S \in \Psi$ implicar que $T \in \Psi$, $\forall\ S \subseteq T \subseteq N$. A definição de família *monótona decrescente* é análoga e deixada como exercício. •

A Proposição seguinte é caso particular de um importante resultado conhecido na literatura como desigualdade FKG (*FKG inequality*). Para maiores detalhes vide Fortuin et al.[1971].

Proposição IV.12: (Correlação Positiva)

Considere um conjunto finito $N=\{a_1,...,a_l\}$ e um vetor $p=(p_1,...,p_l) \in [0,1]^l$. Suponha que um subconjunto $Y \subseteq N$ seja obtido aleatoriamente onde cada elemento a_k de Y seja selecionado independentemente com probabilidade p_k. Para cada $\Psi \subseteq 2^N$ (onde 2^N é o conjunto de todas as partes de N), seja $Pr_p(\Psi)=Pr(Y \in \Psi)$. Considere ainda $F_1,F_2,...,F_s \subseteq 2^N$ uma seqüência qualquer de famílias monótonas crescentes . Então:

$$\Pr_p\left(\bigcap_{k=1}^{s} F_k\right) \geq \prod_{k=1}^{s} \Pr_p(F_k)$$

Diz-se, neste caso, que as famílias $F_1,F_2,...,F_s$ estão *correlacionadas positivamente.*

Prova: (Teorema 3.2 do Capítulo 6 de Alon&Spencer[1992]). •

Pode-se aplicar este resultado ao PRC da seguinte forma. Note que, se $l=n$, o conjunto N pode estar associado a uma restrição genérica do PRC. Assim:

$$N = \{a_1,...,a_n\} \qquad \leftrightarrow \qquad \sum_{j=1}^{n} a_j x_j \geq 1$$

Ainda, o subconjunto $Y \subseteq N$ pode estar associado a uma solução randômica $x \in \{0,1\}^n$ do PRC onde $y \in [0,1]^n$ (solução relaxada do PRC) representa o vetor de probabilidades *p*. A família $\Psi \subseteq 2^N$ estará associada ao conjunto de todas as soluções 0-1 satisfazendo uma restrição particular do PRC. Em outras palavras, cada subconjunto $S \in \Psi$ estará associado a uma solução $x \in \{0,1\}^n$ satisfazendo a restrição correspondente. O valor $Pr_p(\Psi)$ representa a probabilidade da solução x (gerada randômicamente) satisfazer a "restrição" Ψ. Note que cada uma das famílias (restrições) associadas ao PRC é monótona crescente pois $a_j=0$ ou 1, $\forall\ j \in \{1,2,...,n\}$. A família associada à restrição $x_1+x_2-x_3 \geq 1$

por exemplo, não é monótona crescente (verifique).

Como discutido anteriormente, se um evento E_k ocorre então o subconjunto $S_k \subseteq V$ não foi coberto. Pode-se mostrar portanto, que os eventos complementares $E_1^c,...,E_m^c$, associados a $S_1,...,S_m$ respectivamente estarão correlacionados positivamente. Assim, com o auxílio da Proposição IV.12 a probabilidade de sucesso pode ser limitada inferiormente da seguinte forma:

$$\Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \geq \prod_{i=1}^{m} \Pr(E_i^c)$$

O exemplo seguinte ilustra bem esta correlação positiva entre eventos:

Exemplo IV.4: (Correlação positiva entre eventos)

Seja $V=\{1,2,3,4\}$ um conjunto de vértices associado ao PRC com subconjuntos $S_a=\{1,2,3\}$ e $S_b=\{3,4\}$ respectivamente. Ao modelar este problema matematicamente obtêm-se o seguinte conjunto de restrições:

$$\begin{cases} x_1 + x_2 + x_3 \geq 1 \\ x_3 + x_4 \geq 1 \end{cases}$$
$$onde \quad x_j \in \{0,1\} \quad para \quad j = 1,2,3,4$$

Seja $x \in \{0,1\}^4$ uma solução gerada randômicamente. É fácil ver que, se a primeira restrição for satisfeita (S_a for coberto) a probabilidade da segunda restrição ser satisfeita será maior, ou seja, $Pr(E_b^c) \leq Pr(E_b^c \mid E_a^c)$. Assim, $Pr(E_a^c \cap E_b^c) = Pr(E_a^c).Pr(E_b^c \mid E_a^c) \geq Pr(E_a^c).Pr(E_b^c)$. •

Novamente, considere F um evento indicando que todas as restrições associadas ao PRC foram satisfeitas (uma solução viável foi obtida). Além disso, considere $x_A(I)$ uma solução gerada no arredondamento não-linear (algoritmo A), e Z^* seu valor ótimo associado. Tem-se então o seguinte resultado:

Proposição IV.13: (Bertsimas&Vohra[1998])

Seja $\Delta_j=\{i \in \{1,..,m\} \mid j \in S_i\}$ e $\Delta = max\{|\Delta_j| \text{ p/ } j=1,2,...,n\}$. Então $E(x_A(I)|F) \leq O(log\Delta).Z^*$.

Prova: Da linearidade do valor esperado e da definição de valor esperado condicional (vide Capítulo II) tem-se que:

$$E(x^*(I)) = E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) = \sum_{j=1}^{n} c_j E\left(x_j \mid F\right) = \sum_{j=1}^{n} c_j . \Pr(x_j = 1 \mid F) \qquad (I)$$

Considere agora $B_j=\{x \in \{0,1\}^n \mid x_j=1\}$ e $B_j^c=\{x \in \{0,1\}^n \mid x_j=0\}$ (para algum $j \in \{1,...,n\}$) *dois* eventos complementares[13]. Da fórmula de Bayes (Capítulo II) conclui-se que:

$$\Pr\left(x_j = 1 \mid F\right) = \frac{\Pr(B_j \cap F)}{\Pr(F)} = \frac{\Pr(x_j = 1).\Pr(F \mid x_j = 1)}{\Pr(F)} \qquad (II)$$

Como definido anteriormente, o evento complementar E_i^c ocorre (para algum $i \in \{1,..,m\}$) se pelo menos um elemento de S_i for selecionado (S_i é coberto). É fácil ver então que:

[13] Note que, $B_j \cap B_j^c = \varnothing$ e $B_j \cup B_j^c = \{0,1\}^n$.

$$\Pr(F) = \Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \qquad e \qquad \Pr(F \mid x_j = 1) = \Pr\left(\bigcap_{i \notin \Delta_j} E_i^c\right) \qquad (III)$$

Observe, da segunda igualdade que, como $x_j=1$, todos os subconjuntos S_i contendo o vértice j de V já foram cobertos. Além disso, como os eventos E_i^c para $i=1,..,m$ são correlacionados positivamente então:

$$\Pr(F) = \Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \geq \Pr\left(\bigcap_{i \notin \Delta_j} E_i^c\right) \Pr\left(\bigcap_{i \in \Delta_j} E_i^c\right) \qquad (IV)$$

De (III) e (IV) tem-se que:

$$\frac{\Pr(F \mid x_j = 1)}{\Pr(F)} = \frac{\Pr\left(\bigcap_{i \notin \Delta_j} E_i^c\right)}{\Pr\left(\bigcap_{i=1}^{m} E_i^c\right)} \leq \frac{1}{\Pr\left(\bigcap_{i \in \Delta_j} E_i^c\right)} \leq \frac{1}{\prod_{i \in \Delta_j} \Pr(E_i^c)}$$

Lembre-se, do arredondamento não-linear que: $Pr(x_j=0)=(1-y_j)^t$ para $j=1,..,n$. Logo, um evento E_i ocorre (p/ algum $i \in \{1,..,m\}$), se nenhum dos k elementos de $S_i=\{i_1,...,i_k\}$ for selecionado. Como $y_{i1}+...+y_{ik} \geq 1$, é fácil ver então que:

$$\Pr(E_i^c) = 1 - \Pr(E_i) = 1 - \prod_{j \in \{i_1,..,i_k\}} (1 - y_j)^t \geq 1 - \left((1 - 1/k)^k\right)^t > 1 - e^{-t}$$

Assim:

$$\prod_{i \in \Delta_j} \Pr(E_i^c) \geq \prod_{i \in \Delta_j} (1 - e^{-t}) = \left(1 - e^{-t}\right)^{|\Delta_j|} \geq \left(1 - e^{-t}\right)^{\Delta}$$

onde $\Delta = max\ \{|\Delta_j|$ para $j=1,..n\}$.
Portanto:

$$\frac{\Pr(F \mid x_j = 1)}{\Pr(F)} \leq \frac{1}{(1 - e^{-t})^{\Delta}} \qquad (V)$$

Após a substituição de (V) em (II), e com o auxílio da desigualdade de Bernoulli tem-se que:

$$\Pr(x_j = 1 \mid F) = \frac{\Pr(F \mid x_j = 1)}{\Pr(F)} . \Pr(x_j = 1) \leq \left(1 - e^{-t}\right)^{-\Delta} . \left(1 - (1 - y_j)^t\right) \leq \left(1 - e^{-t}\right)^{-\Delta} . t y_j \qquad (VI)$$

Finalmente, substituindo (VI) em (I):

$$E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) \leq \sum_{j=1}^{n} c_j \left(1 - e^{-t}\right)^{-\Delta} . t y_j \leq t . \left(1 - e^{-t}\right)^{-\Delta} \sum_{j=1}^{n} c_j y_j$$

Escolhendo $t=ln\Delta$ e assumindo $\Delta \geq 2$ segue que:

$$E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) \leq \ln n . \left(\frac{1}{(1 - e^{-\ln \Delta})^{\Delta}}\right) Z^* = \ln n . \left(\frac{1}{(1 - 1/\Delta)^{\Delta}}\right) Z^* \leq 4 . O(\ln \Delta) . Z^*$$

Ou ainda:

$$E\left(\sum_{j=1}^{n} c_j x_j \mid F\right) \leq O(\log \Delta).Z^*$$

completando a prova. •

Observe que, ao reduzir o número de repetições *t* de *O(logm)* para *O(logΔ)* no arredondamento não-linear consegue-se uma solução O(logΔ)-aproximada para o PRC. Isto representa um ganho de qualidade se $\Delta<m$. Neste caso, entretanto, a probabilidade de sucesso poderá ser menor, embora estritamente positiva. Para constatar esse fato note, da Proposição IV.13 acima que: $Pr(E_i^c)>1-e^{-t}$, para $i\in\{1,..,m\}$. Tem-se então da correlação positiva entre eventos:

$$\Pr(F) = \Pr\left(\bigcap_{i=1}^{m} E_i^c\right) \geq \prod_{i=1}^{m} \Pr(E_i^c) = \left(1 - \frac{1}{e^t}\right)^m > 0$$

Observe que a probabilidade *Pr(F)* será pequena, mas estritamente positiva, se $t=\log\Delta$ e o número de subconjuntos *m* for bastante grande. Essa probabilidade de sucesso (estritamente positiva) será fundamental no processo de *derandomização* discutido com mais detalhes na Seção IV.3.4.2.

IV.3.4 - Construção Probabilística de Algoritmos Determinísticos (Derandomization)

Em algumas situações, será possível construir algoritmos determinísticos, com o auxílio de técnicas probabilísticas. Em outras palavras, deseja-se construir um algoritmo determinístico sem que se sacrifique muito da qualidade da solução e/ou o tempo de processamento obtidos no procedimento randômico. Infelizmente, não se conhece um mecanismo universal de conversão que seja aplicável a todas as situações.

Apesar de não existir um termo apropriado em português para esta técnica, talvez o mais conveniente seja chama-la *derandomização* (semelhante ao temo inglês *derandomization*). Entre os métodos de derandomização mais conhecidos na literatura pode-se citar: o método das expectâncias condicionais, o método dos estimadores pessimistas e *k-wise independence* (para maiores detalhes sobre o método dos estimadores pessimistas consulte Raghavan&Thompson[1988]). Nesta seção será dada uma atenção especial ao método das expectâncias condicionais.

Suponha que uma variável aleatória X assuma valores $x_1,x_2,...$ (finitos ou não). Além disso, considere Y^c o evento complementar associado a um evento Y qualquer. No método das expectâncias condicionais utiliza-se freqüentemente a seguinte definição de valor esperado condicional (vide Capítulo II):

$$E(X \mid Y) = \sum_i x_i . \Pr(X = x_i \mid Y)$$

Além disso:

$$E(X) = E(X \mid Y).\Pr(Y) + E(X \mid Y^c).\Pr(Y^c) \qquad (4)$$

Note a partir da expressão acima que $E(X) \leq (Pr(Y)+(1-Pr(Y))).max\{E(X|Y), E(X|Y^c)\} = max\{E(X|Y), E(X|Y^c)\}$. Analogamente, pode-se concluir que que*: min*$\{E(X|Y), E(X|Y^c)\} \leq E(X)$. Esta abordagem é interessante já que nos permite definir um mecanismo para tomada de decisões sempre limitando inferiormente ou superiormente (conforme o caso) o valor esperado *E(X)*. Nas seções seguintes está técnica será aplicada aos problemas *MAX-SAT* e Recobrimento de conjuntos (*Covering Problem*).

IV.3.4.1 – O Problema MAX-SAT

Considere novamente o problema *MAX-SAT* discutido na Seção IV.3.2. Deseja-se, a partir da solução randômica 0,5-aproximada gerada pelo Algoritmo A_1, determinar uma solução determinística também 0,5-aproximada. O raciocínio aplicado a esta situação pode ser facilmente estendido aos procedimentos A_2 e A_3 respectivamente.

Como A_1 retorna uma solução randômica 0,5-aproximada (vide Seção IV.3.2) tem-se que:

$$0{,}5.x^*(B) \le E\left(x_{A_1}(B)\right) = E\left(\sum_{j=1}^{m} Z_j\right)$$

onde Z_j é variável aleatória 0-1 associada à cláusula C_j da expressão B na forma normal conjuntiva. Deve-se, inicialmente, decidir se a variável x_1 (associada a C_1) é *falsa* ou *verdadeira* (V ou F). Seja Y um evento indicando que $x_1=V$, e Y^c indicando que $x_1=F$. Tem-se de (4) que:

$$E\left(\sum_{j=1}^{m} Z_j\right) = E\left(\sum_{j=1}^{m} Z_j \mid x_1 = V\right).\Pr(x_1 = V) + E\left(\sum_{j=1}^{m} Z_j \mid x_1 = F\right).\Pr(x_1 = F)$$

Ou seja:

$$E(x_{A_1}(B)) \le \max\left\{E\left(x_{A_1}(B) \mid x_1 = V\right), E\left(x_{A_1}(B) \mid x_1 = F\right)\right\}$$

Considere $E(x_{A1}(B)|X_1)) = max\{E(x_{A1}(B)|x_1=V), E(x_{A1}(B)|x_2=F)\}$. Repetindo-se o processo para o literal x_2 conclui-se que:

$$E(x_{A1}(B)) \le max\{E(x_{A1}(B)|X_1, x_2 =V), E(x_{A1}(B)| X_1, x_2=F)\}.$$

Assim, após $n-1$ passos, chega-se a:

$$E(x_{A1}(B)) \le max\{E(x_{A1}(B)|X_1,...,X_{n-1}, x_n =V), E(x_{A1}(B)| X_1,...,X_{n-1}, x_n =F)\} = E(x_{A1}(B)|X_1,...,X_n)$$

Provou-se finalmente que: $0{,}5.x^*(B) \le E(x_{A1}(B)) \le E(x_{A1}(B)|X_1,...,X_n)$. O algoritmo seguinte, sintetiza todas as etapas realizadas:

Procedimento IV.7: MAX-SAT Derandomizado
Início
  Para $k=1$ **até** n **faça**
    Calcula:
      $R \leftarrow E(x_{A1}(B)|X_1,...,X_{k-1}, x_k=V);$
      $S \leftarrow E(x_{A1}(B)|X_1,..., X_{k-1}, x_k=F);$
    Se $R \ge S$ **então**
      $x_k=V$
    senão
      $x_k=F$
  fim.
  Retorna $x_{A1}(B)$ e solução $(X_1,..., X_n);$
fim.

Figura IV.17: MAX-SAT *Derandomizado*

Observe acima que, a cada iteração do procedimento, os valores R e S (ou seja,

$E(x_{A1}(B)|X_1,...,X_{k-1},x_k=V \text{ ou } F)$) são calculados. O exemplo seguinte ilustra o cálculo dos valores *R* e *S* respectivamente:

Exemplo IV.5: Considere a seguinte expressão booleana com 3 literais e 5 cláusulas:

$$B = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_3)$$

Do procedimento MAX-SAT acima (Figura IV.17) deve-se calcular inicialmente $E(x_{A1}(B)| x_k=V))$ e $E(x_{A1}(B)| x_k=F))$ respectivamente. Para faze-lo basta notar que:

$$E(x_{A1}(B) \mid x_1 = V \text{ ou } F) = E\left(\sum_{j=1}^{m} Z_j \mid x_1 = V \text{ ou } F\right) = \sum_{j=1}^{m} E(Z_j \mid x_1 = V \text{ ou } F) = \sum_{j=1}^{m} 1.\Pr(Z_j = 1 \mid x_1 = V \text{ ou } F)$$

Assim, de nosso exemplo:

$$E(x_{A1}(B) \mid x_1 = V) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid x_1 = V) = 1 + (1/2) + (3/4) + 1 + (1/2) = 15/4 \cong 3{,}75$$

Analogamente, para $x_1=F$ tem-se:

$$E(x_{A1}(B) \mid x_1 = F) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid x_1 = F) = (1/2) + 1 + 1 + (3/4) + (1/2) = 15/4 \cong 3{,}75$$

Como $E(x_{A1}(B) \mid x_1=V) = E(x_{A1}(B) \mid x_1=F)$, considere, sem perda de generalidade que $X_1=V$. Assim:

$$E(x_{A1}(B) \mid X_1, x_2 = V) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, x_2 = V) = 1 + 1 + (1/2) + 1 + (1/2) = 4$$

$$E(x_{A1}(B) \mid X_1, x_2 = F) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, x_2 = F) = 1 + 0 + 1 + 1 + (1/2) = 3{,}5$$

Logo, $X_2=V$. Finalmente, para o literal x_3 tem-se:

$$E(x_{A1}(B) \mid X_1, X_2, x_3 = V) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, X_2, x_3 = V) = 1 + 1 + 0 + 1 + 1 = 4$$

$$E(x_{A1}(B) \mid X_1, X_2, x_3 = F) = \sum_{j=1}^{5} \Pr(Z_j = 1 \mid X_1, X_2, x_3 = F) = 1 + 1 + 1 + 1 + 0 = 4$$

Note portanto que $E(x_{A1}(B)|X_1, X_2, X_3) = 4$, onde $X_1=V$, $X_2=V$ e $X_3=V$ ou F representa uma solução 0,5-aproximada. •

Como discutido a seguir, utilizando-se um raciocínio análogo será possível construir uma solução *derandomizada* para o problema de Geral de Recobrimento (*General Covering Problem*), em especial, para o problema de Recobrimento de Conjuntos (*Set Covering Problem*).

IV.3.4.2 – O Problema de Recobrimento de Conjuntos (*Set Covering Problem*)

Analogamente à seção anterior, o objetivo aqui será gerar uma solução determinística para o PRC utilizando-se técnicas e ferramentas probabilísticas. Neste caso, entretanto, a solução obtida será *O(logm)*-aproximada sendo portanto um pouco inferior àquela apresentada na Proposição IV.13.

Diferentemente do problema MAX-SAT (onde uma solução viável era obtida de maneira trivial) o objetivo aqui será gerar uma solução determinística preocupando-se obviamente em satisfazer todas as restrições. Este problema é resolvido com o auxílio da seguinte função potencial:

$$\Phi(x) = \Phi(x_1,..., x_n) = \sum_{j=1}^{n} c_j x_j + M \sum_{i=1}^{m} Y_i$$

onde:

$$Y_i = \begin{cases} 1, & \sum_{j \in S_i} x_j = 0 \\ 0, & \textit{caso contrário} \end{cases}$$

Observe que uma solução $x \in \{0,1\}^n$ será viável se, e somente se, $\Phi(x) \leq M$ para M escolhido convenientemente.Caso contrário, se $x \neq 0$ e se algum subconjunto S_i de V não for coberto, então $Y_i=1$ e $\Phi(x)>M$ (solução inviável). O valor atribuído à constante M será discutido mais adiante.

Da linearidade do valor esperado tem-se que:

$$E(\Phi(x)) = \sum_{j=1}^{n} c_j E(x_j) + M \sum_{i=1}^{m} E(Y_i)$$

onde:

$$\sum_{i=1}^{m} E(Y_i) = \sum_{i=1}^{m} 1.\Pr(Y_i = 1)$$

Sem perda de generalidade considere que apenas uma das restrições seja violada (para algum $i \in \{1,..,m\}$. Assim:

$$\Pr(Y_i = 1) = \prod_{j \in S_i} (1 - y_j)^t \leq \left(\left(1 - \frac{1}{|S_i|} \right)^{|S_i|} \right)^t \leq e^{-t}$$

Como $E(x_j)=Pr(x_j=1)=1-(1-y_j)^t \leq t.y_j$ (no arredondamento não-linear) e $Pr(F)>(1-e^{-t})^m$ (probabilidade de sucesso) então:

$$E(\Phi(x)) \leq t.Z_{RL} + M\left(1 - (1 - e^{-t})^m\right)$$

onde $Z_{RL}=\Sigma_j c_j y_j$ representa o valor ótimo associado a uma solução $y \in [0,1]^n$ (relaxação linear).

Escolhendo-se M convenientemente, a questão que se coloca agora é: como determinar uma solução determinística (aqui representada por X) de maneira que $E(\Phi(X)) \leq M$? Do método das expectâncias condicionais tem-se que:

$$\begin{aligned} E(\Phi(X)) &= E(\Phi(X) \mid x_1 = 1).\Pr(x_1 = 1) + E(\Phi(X) \mid x_1 = 0).\Pr(x_1 = 0) \\ &\geq (\Pr(x_1 = 1) + (1 - \Pr(x_1 = 1))).\min\{E(\Phi(X) \mid x_1 = 1),\ E(\Phi(X) \mid x_1 = 0)\} = E(\Phi(X) \mid X_1) \\ &\geq \min\{E(\Phi(X) \mid X_1, x_2 = 1),\ E(\Phi(X) \mid X_1, x_2 = 0)\} = E(\Phi(X) \mid X_1, X_2) \end{aligned}$$

Repetindo-se o processo, após $n-1$ passos chega-se a:

$$E(\Phi(X)) \geq \min\{E(\Phi(X) \mid X_{1,},..., X_{n-1}, x_n = 1),\ E(\Phi(X) \mid X_{1,},..., X_{n-1}, x_n = 0)\} = E(\Phi(X) \mid X_{1,},..., X_n)$$

Portanto:

$$E(\Phi(X) \mid X_{1,},..., X_n) \leq E(\Phi(X)) \leq t.Z_{RL} + M(1 - (1 - e^{-t})^m \leq M$$

A constante *M* deve ser escolhida de maneira que o seguinte problema de otimização seja resolvido:

$$\begin{aligned} & \min M \\ s.t. \quad & t.Z_{RL} + M\left(1-\left(1-e^{-t}\right)^m\right) \le M \\ & t, M \ge 0 \end{aligned}$$

Note que, escolhendo:

$$M = \frac{\log m}{(1-1/m)^m}.Z_{RL} \quad e \quad t = \log m$$

tem-se uma solução *X* gerada determinísticamente onde:

$$\sum_{j=1}^{n} c_j X_j \le \Phi(X) \le \frac{\log m}{(1-1/m)^m}.Z_{RL} = 4.\log m.Z_{Rl} = O(\log m).Z_{RL}$$

Observe que: $(1\text{-}1/m)^m \ge 1/4, \ \forall\, m \ge 2$.

Note que, após a substituição de *M* é *t* nas restrições do problema de programação linear tem-se:

$$\log m.Z_{RL} + \frac{\log m}{(1-1/m)^m}.Z_{RL}.\left(1-\left(1-1/e^{\log m}\right)\right) \le \log m.(1-1/m)^m.Z_{RL} \Rightarrow$$

$$1 + \frac{1}{(1-1/m)^m} - \left(\frac{(1-1/e^{\log m})^m}{(1-1/m)^m}\right) \le \frac{1}{(1-1/m)^m} \Rightarrow (1-1/m)^m \le (1-1/e^{\log m})^m \Rightarrow$$

$$(1-1/e^{\ln m})^m \le (1-1/e^{\log m})^m.$$

Note que a última desigualdade é verdadeira pois, $lnm = logm/loge$, ou seja, como $loge > 1$ então $lnm < logm$.