EDUCAÇÃO e TECNOLOGIA

Revista do Instituto Politécnico da Guarda

"EDUCAÇÃO E TECNOLOGIA"
Revista do Instituto Politécnico da Guarda

Director: **João Bento Raimundo**

Redacção: **Rua Comandante Salvador do Nascimento**
Telf. 21634/23982 **6300 GUARDA**

Propriedade: **Instituto Politécnico da Guarda**

Execução Gráfica: **Secção de Reprografia do IPG**

Depósito Legal Nº 17.891/87

Nº VII / Janeiro de 1991

PROGRESSO POR OBJECTIVO

O sétimo número de "*Educação e Tecnologia*" coincide com o início de mais um ano lectivo, o mesmo é dizer, com uma nova fase do Instituto Politécnico da Guarda. Nova, porque o Instituto Politécnico da Guarda cresceu em número de cursos, de alunos e professores, aumentando as exigências, qualitativas e quantitativas. Enfim, o Instituto Politécnico lançou já os seus primeiros diplomados.

Hoje são já umas dezenas; o amanhã, que é breve, os fará crescer.

Isto significa que a nossa Instituição é posta à prova em termos práticos.

Estamos a desenvolver uma formação que dê aos nossos jovens uma realização académica a par das exigências da sociedade moderna; que da justaposição de ambas surja uma adequação o mais perfeita possível à resposta interior do indivíduo no campo do estar, do fazer, do ter, do ser.

O espaço de diálogo, de abertura, de confronto de ideias, de registo de experiências que vem constituindo "*Educação e Tecnologia*", ficaria incompleto se nele não viessem a tomar lugar também aqueles que primeiro motivaram a sua existência.

Bem-vindos serão, também, os registos de quantos, como empregadores, vão testar, no terreno, o que laboriosamente proporcionámos que se ajustasse às solicitações de uma produção eficaz e digna.

Quisemos dar mais oportunidades ao nosso Distrito - por isso existimos como Instituição de Ensino Superior. Quisemos dar mais oportunidades à juventude - por isso aumentámos o número de vagas e de cursos, apostámos na qualidade e formação do corpo docente, continuamos a melhorar as instalações. Queremos dignificar o ensino e engrandecer o País - dialogar, modificar, adequar.

Parafraseando A. Comte:

"*Amor por princípio / Competência por base / Progresso por objectivo*".

João Bento Raimundo
Presidente da C.I. do I.P.G.

MÉTODO DE "BRANCH AND BOUND" APLICADO NA PROGRAMAÇÃO LINEAR INTEIRA

Amândio Pereira Baia *

RESUMO: Embora tenham sido propostas várias aproximações para resolver determinados tipos de problemas discretos, o método de "*Branch and Bound*" tem mostrado uma boa performance na resolução da maior parte dos problemas reais.

INTRODUÇÃO

O método de "*Branch and Bound*" é uma aproximação muito útil na resolução de problemas combinatórios, discretos e programação inteira em geral. Originalmente apresentado por Lang e Doig [1], enquadra-se na classe dos *métodos de enumeração*, uma das classes em que é possível classificar os vários métodos de resolução do problema de Programação Linear Inteira.

Quando o número total de soluções viáveis para o problema a ser resolvido for muito pequeno, cada solução pode ser analisada individualmente, e comparada com as outras, até ser encontrada a óptima. Este é o *método de enumeração total ou exaustivo*. Na maior parte dos problemas na vida real, este método é impraticável, uma vez que o número total de soluções viáveis, é muito grande.

O método de "*Branch and Bound*"[2] fornece uma metodologia de procura de uma solução óptima viável usando

* Professor Adjunto da ESTG
Director do Departamento de Gestão

(1) - Land, A.H., and A.G. Doig, "An Automatic Method for Solving Discrete Programming Problems", *Econometrica*, 1960, 28(3): 497-520

(2) - Lawer, E.L., and D. E. Wood, "Branch and Bound Methods: A Survey", *Operations Research*, 1966, vol 14, pp. 669-719

apenas uma *enumeração parcial*. No decurso da aplicação do método de "*Branch and Bound*", o conjunto das soluções viáveis é fraccionado em subconjuntos, mais simples. Em cada etapa, um subconjunto promissor é escolhido e um esforço é feito para encontrar a melhor solução viável. Se esta for encontrada, este subconjunto é "*fathomed*". Se não for encontrada, o subconjunto é novamente fraccionado em dois ou mais subconjuntos mais simples (esta operação é chamada de "*Branching*") e o processo é repetido novamente.

A execução do método de "*Branch and Bound*", é ajudada pelo cálculo do valor óptimo da função objectivo (se puder ser calculado sem muito esforço), ou pela limitação (Bound) desse valor em cada subconjunto resultante da partição gerada. Os limites são usados para escolher subconjuntos promissores na procura do valor óptimo e permitem desprezar alguns subconjuntos que possivelmente não poderão conter a solução óptima viável. Algumas das vezes a técnica de "*Bound*" aplicada em dado subconjunto, produz a melhor solução viável desse subconjunto, permitindo assim concluir que não vale a pena continuar a analisar este subconjunto (*fathom*).

Os métodos enumerativos de optimização em contraste com os métodos algébricos (i. e., aqueles que são baseados na prova teórica das condições necessárias e suficientes de optimização; por exemplo o método do simplex para resolver problemas de programação linear, tenta encontrar uma base que seja primal e dual viável) têm de examinar de forma enumerativa todas as soluções viáveis na procura da óptima. Esta a razão por que os métodos enumerativos são geralmente aplicados a problemas discretos.

Existem diversas situações a ponderar no método de "*Branch and Bound*"[3]. Vamos discuti-las tomando como referência um problema em que a função objectivo, z, tem de ser minimizada. Estas situações são:

1. Estratégia do limite inferior.
2. Estratégia de partição (Branching)[4]
3. Estratégia de Procura (Search)

ANÁLISE DA SITUAÇÃO

1. ESTRATÉGIA DO LIMITE INFERIOR[5]

O valor mínimo de z é obtido com precisão se o problema

(3) - G. Mitra, "Investigation of Some Branch and Bound Strategies for the Solution of Mixed Integer Linear Problems", *Mathematical Programming*, 4, 2, pp. 155-170, April 1973

(4) - J. Dakin, "A Tree Search Algorithm for Mixed Integer Programming Problem", *Computer Journal*, 8, 3, pp. 250-255, 1965

(5) - Ole Bild, Jakob Krarup, "Sharp Lower Bounds and Efficient Algorims for the Simple Plant Location Problem", Anals of Discret Mathematics 1, 1977, 79-97
Barrie M. Baker, "Linear Relaxations of the Capacitated Warehouse Location Problem", Journal of Operacional Research Society, vol 33, pp. 475-479, 1982

original for resolvido, mas pode acontecer que seja de resolução muito difícil. A estratégia do limite inferior, calcula um limite inferior para o menor valor de z . Deve ser de fácil implementação e de computação muito eficiente. Entre várias estratégias de determinação do limite inferior, a que der um valor mais próximo do valor mínimo de z, sem acarretar um acréscimo de esforço de computação, poderá conduzir ao algoritmo mais eficiente. Algumas destas estratégias são discutidas a seguir.

1.1. Relaxamento das restrições difíceis [6]

Neste método todas as restrições difíceis são relaxadas e z é minimizada sujeito apenas às restrições (fáceis de manejar) restantes. O valor mínimo de z do problema relaxado, representa um limite inferior para o valor mínimo de z do problema original.

1.2. Modificação da Função Objectivo

Neste método uma função objectivo modificada, f, é construída satisfazendo as seguintes propriedades:

1. $f \leq z$ para todas as soluções viáveis do problema original
2. Do ponto de vista de computação, deve ser mais fácil minimizar f sujeita às restrições originais.

Se a função f satisfizer estas propriedades deve ser construída e o valor mínimo de f constitui um limite inferior para o valor mínimo de z.

1.3. Relaxação Lagrangeana [7]

Suponha o problema original da seguinte forma:

Minimizar $z(x)$
Sujeito a:

$$g_i(x) \geq \emptyset \qquad i = 1, \ldots, r \qquad (1)$$

e $\quad x \in X \qquad (2)$

---

(6) - A. M. Geoffrin, "Lagrangian Relaxation for Integer Programming", *Mathematical Programming Study* 2, 1974, pp. 82-114
Marshall L. Fisher, "An Appications Oriented Guide to Lagrangean Relaxation", *Interfaces*, 15:2, March-April, 1985, pp. 10-21
Fisher M. L., "The Lagragean Relaxation Method for Solving Integer Programming Problems", *Management Science*, vol 27, Nol, pp. 1-18
(7) - M. L. Fisher and P. F. Shapiro, "Construtive Duality in Integer Programming", SIAM, *Journal of Applied Mathematics*, 27, pp. 31-52, 1974
M. Held, P. Wolfe and H. Crowder, "Validation of Subgradient Optimization", *Mathematical Programming*, 6, 1, pp. 105-109, February 1974

onde $g_i(x) \geq \emptyset$, $i = 1, \ldots, r$, são as restrições de difícil manejamento, e X o conjunto das soluções viáveis das restantes restrições. Quando a função objectivo for a minimizar, se se pretender obter um limite inferior para z, relaxando algumas restrições usando a relaxação Lagrangeana, todas as inequações a serem relaxadas devem ser escritas na forma "$g_i(x) \geq \emptyset$". Uma nova função objectivo, é construída, multiplicando o primeiro membro de cada inequação relaxada por um valor não negativo, e subtrair a sua soma do valor original da função objectivo. Os multiplicadores usados chamam-se multiplicadores de Lagrange. Se $u = (u_1, \ldots, u_r) \geq \emptyset$ for o vector multiplicativo de Lagrange usado para relaxar (1), então o problema original relaxando (1), fica:

$$\text{Minimizar} \qquad L(x, u) = z(x) - \sum_{i=1}^{r} u_i \, g_i(x)$$

$$\text{Sujeito a:} \qquad x \in X$$

O valor mínimo da função objectivo é claramente uma função do vector multiplicativo de Lagrange u. Seja L(u). Demonstra-se que para qualquer valor de $u \geq \emptyset$, L(u) representa um limite inferior do mínimo valor de z(x) do problema original. Para determinar as restrições a relaxar devem ser usadas as seguintes regras:

1. Se X for o conjunto das soluções viáveis das restantes restrições não relaxadas, então para qualquer $u >= \emptyset$, a computação de L(u) (i. e., o valor mínimo de $L(x, u)$ em $x \in X$), deve ser de cálculo fácil.
2. O subconjunto das restrições relaxadas, deve ser o subconjunto mais pequeno que satisfaça 1.

Uma vez determinado o subconjunto das restrições relaxadas, se um algoritmo[8] puder ser desenvolvido para encontrar o máximo valor de L(u) com $u \geq \emptyset$, ou pelo menos encontrar um valor para L(u) próximo do seu valor máximo com $u \geq \emptyset$, e com um esforço de computação razoável, este algoritmo poderá ser usado para obter um bom vector multiplicativo de Lagrange u, e servir como um bom limite inferior para o valor mínimo de z no problema original.

As restrições a serem relaxadas e o algoritmo a ser usado para encontrar um bom vector multiplicativo dependem da estrutura do problema original. Se se conseguirem desenvolver boas técnicas para lidar com estes

(8) - Forrest, J. J. H., Hirst, J. P. H., and Tomlin, J. A., 1974, "Practical Solution of Large Mixed Integer Programming Problems with UMPIRE", Management Science, 20, 736-773.

aspectos, este método pode produzir limites inferiores satisfatórios.

2. ESTRATÉGIA DE PARTIÇÃO (Branching)

Sendo S um conjunto, uma partição de S é uma divisão em subconjuntos $(S_1, ..., S_r)$ de modo que:

(i) $r \geq 2$, e

(ii) Os subconjuntos $S_1, ..., S_r$, são mutuamente exclusivos e a sua união é S.

A estratégia de partição divide o conjunto das soluções viáveis em dois ou mais conjuntos. Cada subconjunto resultante da partição representa um conjunto das soluções viáveis do problema, e obtém-se impondo restrições adicionais ao problema original. Estes problemas chamam-se problemas candidatos. Os problemas candidatos devem ser gerados de forma a que a estratégia do limite inferior possa ser aplicada a cada um deles.

No processo de aplicação da estratégia do limite inferior ao problema candidato pode acontecer que seja encontrada uma solução óptima. Uma solução é óptima para o problema candidato se satisfizer todas as restrições deste problema e a sua função objectivo tiver um valor igual ao seu correspondente limite inferior. Se isto acontecer, este problema diz-se que está "*fathomed*" (não vale a pena continuar a sua análise).

Se o problema não for "*fathomed*", apenas teremos um limite inferior para o valor da sua função objectivo. Deve ser possível aplicar a estratégia de partição a cada problema candidato e gerar dois ou mais subproblemas candidatos de modo a que as seguintes propriedades se mantenham:

1. Cada subproblema candidato é obtido impondo restrições adicionais ao problema candidato.
2. O conjunto das soluções viáveis do subproblema candidato forma uma partição das soluções viáveis do problema candidato.
3. Os limites inferiores dos valores mínimos da função objectivo dos subproblemas candidatos, são tão altos quanto o possível.

A operação de partição do conjunto das soluções viáveis do problema candidato, gerando os subproblemas candidatos chama-se partição. Se Q for um subproblema candidato gerado pela partição do problema P, P é chamado de problema parente de Q. Assim, será conveniente que a estratégia de partição gere dois subproblemas candidatos sempre que for aplicada. Esta estratégia chama-se estratégia binária de partição e assume-se como conduzindo a algoritmos eficientes.

A eficiência geral do método de "Branch and Bound", depende em grande parte da eficiência da estratégia de partição empregue, tendo em conta a propriedade 3.

2.1. Partição usando uma variável ø - 1

Se num problema candidato existir uma variável x_1 que apenas pode tomar o valor ø ou 1, adicionando uma ou mais restrições, "$x_1 = ø$" ou "$x_1 = 1$", respectivamente, às restrições do problema candidato, à variável x_1 chama-se variável de partição. Entre as variáveis em que a partição é possível a escolhida deve ser a que satisfizer a propriedade 3 enunciada anteriormente. A estratégia de partição deve fornecer um bom critério para a escolha das variáveis a fazer a partição.

2.2. Partição usando uma variável inteira

Considere-se um problema candidato em que uma função linear das variáveis $a_1 x_1 + ... + a_n x_n$ tem de ser inteira. Sendo α um inteiro, então juntando uma ou mais restrições, "$a_1 x_1 + ... + a_n x_n \leq \alpha$" ou "$a_1 x_1 + ... + a_n x_n \geq \alpha+1$", respectivamente, ao problema candidato, ocorre uma partição gerando dois problemas subcandidatos.

3. A ESTRATÉGIA DE PROCURA (Search)

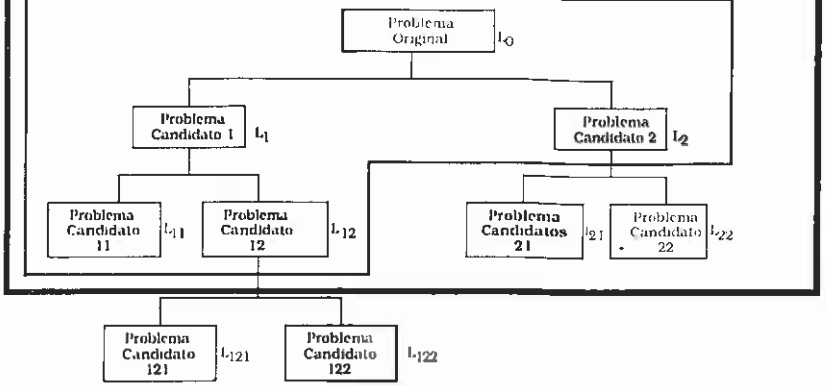
3.1. Estágio Inicial

Inicialmente, o problema original é apenas o único problema candidato. Atente-se na figura 1. Começa-se por lhe aplicar a estratégia do limite inferior. Se for encontrada uma solução óptima, aquando da aplicação da estratégia do limite inferior, então o algoritmo termina. Caso contrário, a estratégia de partição é aplicada ao problema original. Nos nós correspondentes aos problemas candidatos 1 e 2 ainda não foi feita a partição e chamam-se nós terminais, nesta etapa do algoritmo. Chama-se lista a todos os problemas candidatos, ainda não "*fathomed*", correspondentes aos nós terminais nesta etapa.

Figura 3

Figura 2

Figura 1



Suponha que $L_1 < L_2$. O conjunto das soluções viáveis dos problemas candidatos 1 e 2 formam uma partição do conjunto das soluções viáveis do problema original. Uma vez que $L_1 < L_2$, existe uma forte possibilidade que a solução óptima do problema candidato 1 tenha um valor menor que L_2 ; se isto acontecer, então a solução é óptima para o problema original. Temporariamente deixa-se de lado o problema 2 e tenta-se encontrar uma solução óptima para o problema candidato 1. No caso de já ter sido encontrada durante o cálculo do limite inferior, L_1 deve ser o valor da função objectivo deste problema original e o algoritmo termina. Doutra maneira aplica-se a estratégia do limite inferior no problema candidato 1. Suponha que este gera dois novos problemas candidatos chamados problemas candidatos 11 e 12. Remove-se o problema candidato 1 da lista e juntam-se os problemas candidatos 11 e 12. Calcula-se L_{11} e L_{12}, ou seja os seus limites inferiores.

O diagrama até este estágio será o apresentado na figura 2, e é conhecido por árvore de procura. A lista nesta etapa consiste dos problemas candidatos 2, 11 e 12.

3.2. Solução incumbente

Em qualquer estágio do algoritmo, se um ou mais dos problemas candidatos que tiver o menor valor da função objectivo, estiver "*fathomed*", a maior solução básica viável entre todas as soluções viáveis obtidas até esse momento, chama-se solução incumbente. No início pode-se não ter uma solução incumbente mas no momento em que uma solução é "*fathomed*" tem-se uma solução incumbente.

Seja $\bar{z}$ o valor da solução incumbente. Então, se até este estágio existir um problema candidato na lista com um limite inferior L associado em que $L \geq \bar{z}$, qualquer solução do problema candidato tem um valor da função objectivo maior ou igual que L (sendo também maior ou igual que $\bar{z}$), mas não é melhor que a solução incumbente corrente. Assim, deve-se apagar o problema candidato da lista. A operação de apagar o problema da lista chama-se "*pruning*".

No processo de aplicação da estratégia do limite inferior ao problema candidato, pode-se encontrar que este é inviável. Este tipo de problemas são também apagados da lista.

Quando um problema candidato é gerado, suponha que é "*fathomed*", pela estratégia do limite inferior. Suponha também que a solução óptima deste problema candidato é $\hat{x}$, que tem um valor da função objectivo de $\hat{z}$. Sendo $\bar{z}$ o valor da função objectivo incumbente corrente, então se $\hat{z} < \bar{z}$, $\hat{x}$ é uma solução melhor do que a solução incumbente corrente e a solução incumbente passa a ser igual a $\hat{x}$. Esta operação chama-se de actualização da solução incumbente. Por outro lado se $\hat{z} \geq \bar{z}$, este problema candidato é "*pruned*". Assim, sempre que um novo problema candidato é "*fathomed*", não mais deve ser junto à lista. A solução óptima deste problema serve apenas para actualizar a solução incumbente.

Assim em qualquer estágio do algoritmo, a lista consiste de todos os problemas não "*fathomed*" e não "*pruned*". As seguintes propriedades devem ser satisfeitas:

(i) O conjunto das soluções viáveis dos problemas candidatos na lista são mutuamente exclusivas.
(ii) Se existir uma solução incumbente nesta etapa, qualquer solução viável do problema original que seja melhor que a solução corrente incumbente é uma solução viável de algum problema candidato da lista.
(iii) Se não existir qualquer solução incumbente até este momento, a união do conjunto das soluções viáveis dos problemas candidatos na lista representa o conjunto das soluções viáveis do problema original.

3.3. A Procura Continuada

Na figura 1, existem três nós terminais, correspondentes aos problemas candidatos 2, 11 e 12. Se qualquer destes nós for "*fathomed*", então devemos ter uma solução incumbente e estaremos em condições de poder fazer o "*pruning*".

Identifique-se o problema candidato associado ao menor limite inferior de entre todos os problemas da lista e chame-se a este problema P. Apague-se P da lista, use-se a estratégia de partição e aplique-se a estratégia do limite inferior aos subproblemas gerados. Se algum destes problemas for inviável, faça o "*pruning*". Se algum for "*fathomed*", actualize a solução incumbente. Se houver uma mudança na solução incumbente, consulte a lista e faça o "*pruning*". Entre os novos subproblemas criados, juntem-se os problemas não "*fathomed*" e não "*pruned*" à lista e vá para a próxima etapa.

Suponha que é feita a partição no problema candidato 2, produzindo os problemas 21 e 22, tendo os limites inferiores associados L_{21} e L_{22}, respectivamente. A figura 2 representa a árvore de procura corrente. Se L_{12} é o menor valor entre L_{11}, L_{12}, L_{21} e L_{22}, o problema candidato 12 é o único em que deve ser feita a partição na próxima etapa. Depois desta partição, a árvore de procura, terá o formato da figura 3. Na prática o algoritmo pode ser operado com apenas a lista dos problemas candidatos e a solução incumbente (sempre que seja obtida), actualizando esta depois de cada etapa.

O algoritmo termina quando a lista dos problemas candidatos ficar vazia. No término, se existir uma solução incumbente, será uma solução óptima do problema original. Se não existir uma solução incumbente, então o problema original deve ser inviável.

De uma forma geral a estratégia de procura dita a sequência em que os nós gerados serão examinados. A estratégia descrita chama-se de estratégia de salto, pois "salta" entre os nós que estão na lista para examinar o nó com o limite inferior. Também é conhecida por estratégia de prioridade, com o limite inferior a representar o critério de prioridade.

3.4. Diferentes estratégias de procura

A estratégia de procura baseada no critério do limite inferior, é uma boa estratégia de procura, mas pode envolver muito trabalho. Outra estratégia de procura, conhecida por "*backtrack*", guarda um dos problemas candidatos da lista com o objectivo de fazer a procura e chama-o de problema candidato corrente. Os outros problemas na lista constituem a pilha.

Se o problema candidato corrente for "*fathomed*", a solução incumbente é actualizada e o problema candidato corrente é desprezado. Se ocorrer uma mudança da solução incumbente, o necessário "*pruning*" é feito na pilha e um

novo problema candidato é escolhido. O algoritmo continua de maneira similar.

Se o problema candidato corrente não for "*fathomed*", aplica-se-lhe a estratégia de partição e a estratégia do limite inferior é aplicada aos subproblemas gerados. Se ambos os novos subproblemas gerados forem "*fathomed*", a solução" incumbente é actualizada, estes problemas candidatos são retirados da pilha e esta é "*pruned*". Um novo problema candidato corrente é escolhido da pilha. Se apenas um dos novos sub-problemas gerados for "*fathomed*", a solução incumbente é actualizada, faz-se o "*pruning*" e, se os outros problemas candidatos não forem "*pruned*", este é considerado o novo problema candidato, e o algoritmo continua. Se nenhum dos problemas candidatos for "*fathomed*", o mais promissor destes (pode ser o asssociado ao limite inferior mais pequeno), é considerado o novo corrente problema candidato, enquanto o outro sub-problema candidato é junto à pilha e o algoritmo continua.

Quando o algoritmo tiver que escolher um problema candidato da lista, o melhor critério de selecção parece ser o de escolher o último problema adicionado. Este critério de selecção chama-se de LIFO (Last In First Out).

Se esta estratégia de procura for empregue, o algoritmo termina quando for necessário escolher um problema candidato da pilha e a pilha se encontrar vazia. Se existir uma solução incumbente nessa altura, é uma solução óptima para o problema original. Se não existir, o problema original deve ser inviável.

Algoritmo de "*Branch and Bound*" aplicado a Problemas de Programação Inteira em Geral

O algoritmo de "*Branch and Bound*" está na base da maioria dos códigos de computador relacionados com a Programação Linear Inteira. Várias versões de programas de computador, estão disponíveis para uso comercial. A maior parte destas versões apenas difere nos detalhes de escolha da variável a fazer a partição em cada nó ou na sequência em que os problemas são analisados.

Uma desvantagem apontada a este método prende-se com a necessidade de ter de se resolver um problema completo em cada nó. Em problemas de larga dimensão, mesmo com o advento de potencialidades de computação cada vez maiores, isto pode tornar-se impossível, particularmente se tivermos em linha de conta que a informação necessária em cada nó é o valor da função objectivo. Este ponto pode ser simplificado se se conseguir obter um "*bom*" limite inferior, pois muitos dos nós podem

deixar de ser analisados depois de conhecido o seu valor da função objectivo.

O ponto anterior conduziu ao desenvolvimento de um procedimento onde fosse desnecessário analisar todos os subproblemas.

Apresentamos a seguir um algoritmo de "*Branch and Bound*" que permite resolver problemas de Programação Linear Inteira gerais.

Formato geral do Problema de Programação Linear Inteira

Minimizar $z(x)$
Sujeito a:

$x \in F$
x_j inteiro, $j = 1, \ldots, n$

0. Inicialização

Resolver o problema linear relaxado do problema original; se a solução satisfazer as restrições de integralidade, então PARAR (a solução é óptima para o problema original).
Encontrar um limite superior Zu do valor óptimo da função objectivo, igual ao valor da função objectivo em dado ponto que seja viável para o problema inteiro, ou $+\infty$ se esse ponto não for conhecido.

1. Participação (*Branch*)

Seleccionar do conjunto restante, as soluções viáveis. (Na primeira iteração escolher F).
Seleccionar um componente não inteiro da solução do problema correspondente, e partir o subconjunto em dois subconjuntos mais pequenos adicionando restrições que excluam o valor não inteiro do componente escolhido.

2. Limitação (*Bound*)

Para cada subconjunto obter um limite inferior Z_l da função objectivo desse subconjunto.

3. Fathom

Examinar cada conjunto que ainda possa conter o ponto óptimo, e excluir um subconjunto de análise se:

a) $Z_l \geq Zu$
ou b) O subconjunto não tem pontos viáveis
ou c) Z_l foi obtido num ponto inteiro viável de um subconjunto e $Z_l < Zu$

No caso c)

Chamar ao ponto inteiro viável a solução incumbente

Fazer $Zu = Z_i$

IR PARA 3 e ver se outros subconjuntos podem ser "*fathomed*".

4. Teste

Se não permanecerem subconjuntos que não estejam "*fathomed*", PARE (a solução incumbente é a solução óptima). Caso contrário, IR PARA 1.

Exemplo prático

A título de ilustração do algoritmo apresentado, vamos usá-lo para resolver o seguinte problema:

Minimizar $z(x) = 3x_1 - 7x_2 - 12x_3$

Sujeito a:

$$\begin{aligned} -3x_1 + 6x_2 + 8x_3 &\leq 12 \\ 6x_1 - 3x_2 + 7x_3 &\leq 8 \\ -6x_2 + 3x_2 + 3x_3 &\leq 5 \end{aligned}$$

x_1 , x_2 , x_3 não negativos e inteiros

O conjunto F das soluções viáveis é um conjunto de todos os vectores não negativos em R^3, de modo a satisfazer as três restrições. Apresentamos a seguir apenas uma iteração (passos 1 a 4) do algoritmo.

Etapa 0 (Inicialização)

Se se ignorar a necessidade de que cada variável tem de ser inteira e usarmos o método do simplex para resolver o resultante problema relaxado de programação linear, obtemos a seguinte solução:

$$X = (0, 10/33, 14/11)^T$$

Esta solução não tem componentes inteiros, o que a torna não óptima para o problema inicial, havendo necessidade de continuar o algoritmo. No estabelecimento de um limite superior para o valor da função objectivo poder-se-á considerar $Zu = +\infty$ e o algoritmo continuará a trabalhar. Mas, considerando a importância deste valor na estratégia de limitação, a solução $X = (0, 0, 0)^T$ por exemplo, poderá aumentar a eficiência do algoritmo, já que satisfaz todas as restrições do problema e a função objectivo tem um valor mais pequeno. Vamos por isso considerá-la como a solução incumbente corrente.

Etapa 1 (Partição)

De acordo com o algoritmo, podemos escolher x_2 ou x_3 como as variáveis a fazer a partição. Arbitrariamente escolhemos x_2, vamos fraccionar o conjunto dos pontos viáveis em duas partes, sendo respectivamente, $x_2 \leq 0$ e $x_2 \geq 1$.

Etapa 2 (Limitação)

Estabelecem-se limites inferiores para o valor da função objectivo dos problemas de programação linear inteira, em cada um dos conjuntos produzidos pela partição. Estes podem-se obter resolvendo os dois problemas resultantes da adição das restrições ao problema linear relaxado original. Os problemas lineares resultantes bem como as suas soluções são os seguintes:

minimizar $z(x)$ $\quad x \in F$

Sujeito a: $x_2 \leq 0$

$X = (0, 0, 1.1)^T$
$Z = -96/7$

minimizar $z(x)$ $\quad x \in F$

Sujeito a: $x_2 \geq 1$

$X = (2/3, 1, 1)^T$
$Z = -17$

Etapa 3 (*Fathom*)

O teste das condições de "*fathoming*", mostra que nenhum dos subconjuntos pode ser excluído de análise mais aprofundada: o valor da função objectivo de qualquer dos subproblemas é inferior ao valor ($Zu = \emptyset$) da solução incumbente corrente; nenhum subproblema tem uma solução inviável ou uma solução inteira.

Etapa 4 (*Teste*)

Ambos os conjuntos permaneceram não "*fathomed*", pelo que tem de se voltar à etapa 1 do algoritmo.

O algoritmo repete-se até todos os nós estarem "*fathomed*", pois antes disso não existe garantia que não se encontre uma solução incumbente melhor que a solução corrente.

Apresentamos a seguir o diagrama de árvore utilizado na resolução do problema, bem como a solução óptima.

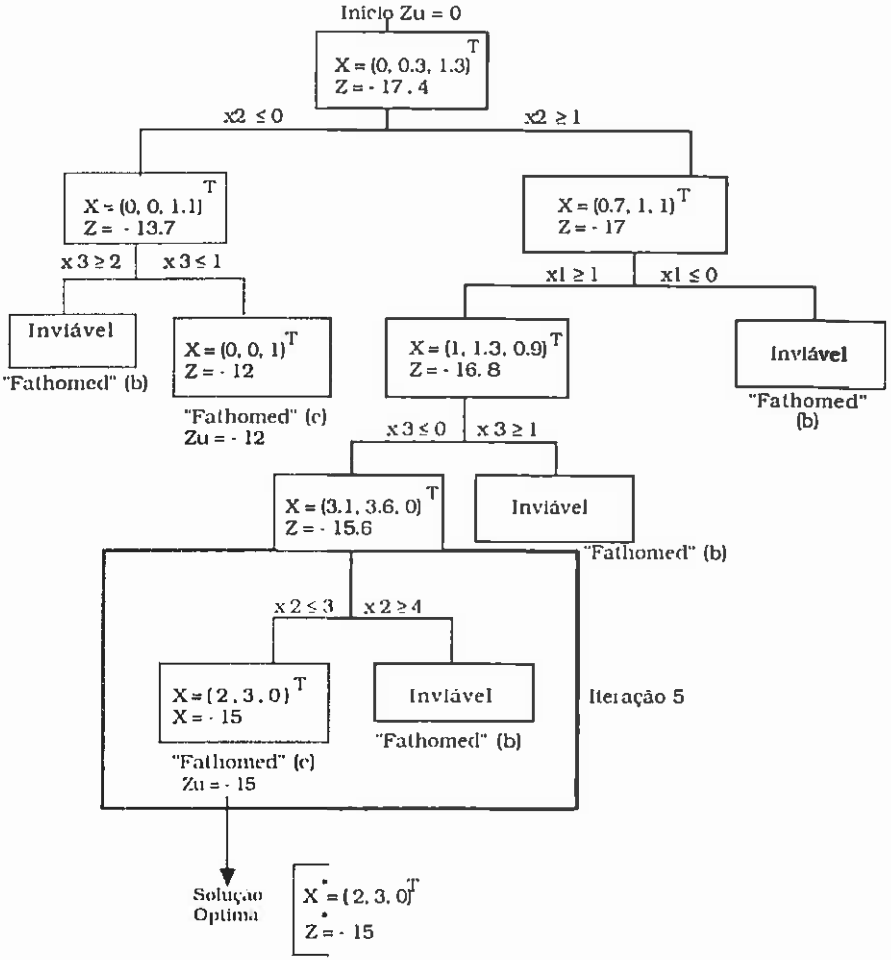


Fig. 4

No exemplo apresentado apenas houve uma vez em que teve de se escolher qual dos nós ainda não "*fathomed*" iria ser tomado em consideração. Isto aconteceu no fim da primeira iteração, e foi escolhido o subproblema da esquerda para ser analisado. Esta escolha foi feita arbitrariamente pois a etapa 1 do algoritmo requer apenas que se escolha um subconjunto e não especifica como deve ser escolhido. No nosso exemplo poderíamos ter analisado a parte direita do diagrama até ao fim, antes de voltar a analisar o problema gerado na primeira iteração. O processo de partição em que são gerados todos os nós no mesmo nível de profundidade antes de se analisarem os outros a uma maior profundidade, chama-se de estratégia de escolha do subconjunto

em lateralidade. Se, pelo contrário, se estender o diagrama o mais possivel antes de serem considerados os nós da esquerda ou direita, esta estratégia chama-se de estratégia de profundidade.

A ordem em que os subconjuntos são escolhidos pode afectar o número total de problemas e serem resolvidos e assim o trabalho necessário para resolver o problema. É praticamente impossível saber antecipadamente qual das estratégias de escolha dos subconjuntos dará melhor resultado para dado problema, mas em dado ponto do processo de solução pode ser possível fazer um palpite razoável se se deve ir em lateralidade ou em profundidade. É relativamente fácil implementar o algoritmo em computador, muito embora seja necessário ter algum cuidado na maneira como é escolhida a variável a fazer a partição. Em muitas aplicações pode satisfazer parar o método de "branch an bound" quando uma solução for encontrada que, muito embora não seja óptima, esteja suficientemente perto.

CONCLUSÃO

Para que o método de "*Branch and Bound*" trabalhe eficientemente, a estratégia de limitação deve fornecer um limite inferior muito próximo do valor mínimo da função objectivo do problema mas com um esforço de computação muito pequeno. A estratégia de partição deve gerar subproblemas candidatos que tenham limites inferiores tão altos quanto o possivel. Também os limites inferiores dos subproblemas candidatos gerados devem ser de computação muito fácil.

Um método de "*Branch and Bound*" bem desenhado deve possibilitar fazer um extensivo e efectivo "*pruning*," o que permitirá localizar o óptimo examinando apenas uma fracção muito pequena do conjunto total das soluções viáveis. Esta a razão por que estes métodos são chamados de métodos de enumeração parcial.

Experiências práticas indicam que uma procura "*backtrack*", permite um "*pruning*" extensivo, e, consequentemente, conduz a algoritmos mais eficientes.

Na maior parte dos algoritmos "*Branch and Bound*" bem desenhados, acontece frequentemente que uma solução óptima viável do problema original torna-se uma solução incumbente

num estágio muito cedo. Um bom heurístico[9] nesta situação serve para terminar o algoritmo num estágio intermédio (quando o limite do tempo disponível do computador for atingido), com a solução incumbente corrente próxima da solução óptima.

Em muitas aplicações práticas de programação inteira e optimização combinatória, o método de "Branch and Bound", tem prestado uma performance muito satisfatória. Se o problema a ser resolvido for muito grande, é muito improvável que este método ou outro qualquer, encontrem uma solução óptima sem um grande esforço de computação. Mas, em muitos destes problemas, o método de "*Branch and Bound*", produz desde muito cedo soluções incumbentes que, embora não possa ser provado que são óptimas, em muitos problemas estão muito próximo da solução óptima. Esta a razão por que a aproximação de "*Branch and Bound*", é muito útil na resolução de aplicações práticas.

(9) - Garfinkel, Robert S. and George L. Nemhauser: Integer Programming, Wiley, New York, 1972
Salkin, Harvey M.: *Integer Programming*, Addison-Wesley, Reading, Mass., 1975
Taha, Hamdy A.: *Integer Programming: Theory, Applications, and Computations*, Academic Press, New York, 1975
Zionts, Stanley: *Linear and Integer Programming*, Prentice-Hall, Englewood Cliffs, N. J., 1974
Hillier, F. S., and Lieberman, G. J., *Introduction to Operations Research*, 3rd ed. San Francisco: Holden-Day, 1980
Wagner, H. M., *Principles of Operations Research*, 2nd ed. Englewood Cliffs, N. J.: Prentice-Hall, 1975