UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA

MS777 - Projeto Supervisionado

Programação Linear Inteira Mista (MILP): modelagem e algoritmos heurísticos

Aluno: Marcel Alves Moro
Orientadora: Prof. Dra. Márcia A. Gomes Ruggiero

Campinas
2013

1 Introdução

O objetivo neste projeto foi trabalhar com problemas de Programação Linear Inteira Mista (MILP: Mixed Integer Linear Programming). Problemas reais de diversas áreas são formulados como MILP, e em geral estão relacionados a problemas de tomada de decisão em atividades industriais, de negócios e área médica. Cada aplicação resultará em um modelo específico, com seu conjunto de variáveis inteiras, binárias e contínuas, além dos conjunto de igualdades e desigualdades lineares e função objetivo.

Problemas formulados como MILP pertencem a classe NP-hard (Nondeterministic Polynomial-time class) e portanto, métodos exatos de resolução têm complexidade exponencial e requerem um tempo excessivo para resolucão, mesmo para problemas de pequeno e médio porte. Por isso é crescente a proposta de métodos heurísticos para resolução de problemas formulados como MILP. Esta abordagem tem como objetivo a obtenção de uma solução próxima da solução ótima, em tempo razoável de execução.

Este projeto consta de duas etapas: *Modelagem* e *Algoritmos*.

Na etapa *Modelagem* pesquisou-se problemas de várias áreas que resultam em modelos MILP. Fez parte deste estudo o entendimento do problema, definição das variáveis, inequações e equações e função objetivo. Foram estudados também procedimentos para realizar transformações na formulação e técnicas de pré-processamento.

Na etapa de *Algoritmos*, realizou-se um estudo teórico do métodos exato *branch-and-bound*, além dos conceitos, objetivos e classificação de métodos heurísticos.

2 Modelagem

Nesta seção veremos alguns exemplos de problemas formulados como MILP e técnicas de pré processamento nas formulações.

2.1 Problema da Mochila

Dado um certo número de itens, e a cada um deles associado um valor e seu peso, o problema da mochila consiste em escolher a melhor combinação de itens restrita a um peso máximo de forma a maximizar o valor agregado a todos eles.

O nome, problema da mochila, vem do exemplo de um andarilho que tem que escolher os itens para levar em sua mochila, a qual tem uma capacidade

máxima de peso, de acordo com a contribuição do valor que cada um deles representa em sua caminhada.

Os parâmetros do problema são:

- n: número de itens;
- a_j: peso do item j;
- c_j: valor associado (lucro) ao item j;
- b: peso máximo que a mochila suporta;

Definimos a variável de decisão y_j como binária, que terá valor 1 se o item j for escolhido para entrar na mochila e 0 caso contrário. Sendo assim, podemos formular o problema da seguinte maneira:

$$\max z = \sum_{j=1}^{n} c_j y_j$$

$$s.a. \begin{cases} \sum_{j=1}^{n} a_j y_j \leq b & (1) \\ y_j = 0 \text{ ou } 1 \qquad\qquad j = 1, \ldots, n & (2) \end{cases}$$

A função objetivo visa maximizar o lucro associado à combinação de itens escolhidos e a única restrição é que a soma dos pesos dos itens escolhidos não ultrapasse a capacidade máxima da mochila.

O problema da mochila é um problema de otimização, onde poderíamos avaliar todas as combinações de itens possíveis e escolher dentre as que não excedem a capacidade da mochila, a melhor. Entretanto, esta não é uma boa abordagem, dado que o número de combinações cresce exponencialmente com o número de itens. Para cada item temos duas opções, incluir ele na mochila ou não, logo para n itens temos 2^n possibilidades de solução.

2.2 Problemas de Planejamento de Produção

Em um problema de planejamento de produção estamos interessados em saber a quantidade que devemos produzir de um determinado produto, de forma a atender a demanda em cada período de tempo t ao longo de um horizonte de tempo T. A demanda em cada período t pode ser satisfeita pelos produtos da produção do período t e os produtos em estoque, ou seja, aqueles que sobraram da produção do período $t-1$. Assumimos aqui que atrasos nas entregas dos pedidos não são permitidos.

Nesta situação estão associados diversos custos:

- custo de *setup*: relacionado à inicialização das máquinas para uma rodada de produção;

- custo unitário de produção: o custo para produzir uma unidade do produto, no qual estão envolvidos mão de obra, matéria prima e tempo de execução das máquinas;

- custo de estoque: custo por unidade mantida em estoque, em decorrência de despesas com armazenagem, risco de ficar obsoleto, impostos, entre outros.

Assumimos que o custo de produção é proporcional à quantidade produzida e o custo de estoque é proporcional a quantidade retida no estoque da produção do período anterior.

O objetivo do problema é minimizar a soma destes três custos em todos os períodos, satisfazendo a demanda em todos eles.

Citamos aqui três tipos de problemas de planejamento de produção: Lote não capacitado, Lote capacitado e Just-in-time.

2.2.1 Lote não capacitado

No lote não capacitado assumimos capacidade de produção ilimitada em cada período, ou seja, haverá apenas uma rodada de produção por período.

Definindo os parâmetros:

- T: número de períodos;

- d_t: demanda no período t;

- f_t: custo de setup no período t;

- c_t: custo de produção por produto no período t;

- h_t: custo de estoque por produto mantido no estoque no período t;

As variáveis de decisão serão: y_t que decide se haverá ou não produção no período t, e x_t que indica o quanto produzir. Logo $x_t = 0$ quando $y_t = 0$ (não há produção) e $x_t > 0$ se $y_t = 1$ (há produção).

A variável de estado no nosso problema é o nível de produtos no estoque ao final do período t. Denotamos esta variável por s_t e assumimos que $s_0 = 0$, ou seja, não há estoque no início da produção do primeiro período.

Sendo assim, podemos formular o problema da seguinte maneira:

$$\min z = \sum_{t=1}^{T}(c_t x_t + f_t y_t + h_t s_t)$$

$$s.a. \begin{cases} s_{t-1} + x_t - s_t = d_t & t = 1, 2, \ldots, T \quad (3) \\ x_t \leq M y_t & t = 1, 2, \ldots, T \quad (4) \\ x_t \geq 0 & t = 1, 2, \ldots, T \quad (5) \\ s_t \geq 0 & t = 0, 1, \ldots, T \quad (6) \\ y_t = 0 \text{ ou } 1 & t = 1, 2, \ldots, T \quad (7) \end{cases}$$

onde M é um número muito grande que podemos definir como $M = \sum_t d_t$, de forma que nenhum x_t ultrapasse esse valor.

A restrição (3) indica que a quantidade mantida em estoque mais a quantidade produzida deve ser igual a demanda mais a sobra do período atual.

A restrição (4) é uma maneira de relacionar as variáveis y_t e x_t, de maneira que force x_t ser igual a zero quando y_t também é. Vejamos: se $y_t = 0$, pela restrição (4) temos que $x_t \leq 0$. Mas por (5) sabemos que $x_t \geq 0$, logo $x_t = 0$, o que era esperado pois $y_t = 0$ indica que não há produção. Já se $y_t = 1$, a restrição (4) se torna redundante ao problema, pois obviamente x_t é menor que M.

2.2.2 Lote capacitado

A diferença do lote capacitado para o não capacitado é que agora há um limite de produção u_t para cada período de tempo, por isso trocamos M por u_t na restrição (4) da formulação acima e podemos modelar o problema da seguinte maneira:

$$\min z = \sum_{t=1}^{T}(c_t x_t + f_t y_t + h_t s_t)$$

$$s.a. \begin{cases} s_{t-1} + x_t - s_t = d_t & t = 1, 2, \ldots, T \quad (8) \\ x_t \leq u_t y_t & t = 1, 2, \ldots, T \quad (9) \\ x_t \geq 0 & t = 1, 2, \ldots, T \quad (10) \\ s_t \geq 0 & t = 0, 1, \ldots, T \quad (11) \\ y_t = 0 \text{ ou } 1 & t = 1, 2, \ldots, T \quad (12) \end{cases}$$

Como agora temos uma capacidade máxima de produção em cada período, quando tivermos $y_t = 1$, a restrição (9) assegura que a produção do período

não ultrapasse essa capacidade. Enquanto que (9) e (10) juntas, continuam relacionando as variáveis x_t e y_t como na seção anterior.

2.2.3 Plano de produção Just-in-time

No plano de produção Just-in-time desejamos atender a demanda de mais de um produto em diversos períodos de tempo. O princípio básico do Just-in-time é manter o nível de estoque zero, ou seja, assim que o produto é fabricado ele já é enviado para o cliente. Porém na prática isso não acontece, pois geralmente acaba sobrando ou faltando alguma quantia do produto ao final de um período. Nesse caso, o excesso significa que a produção começou muito cedo e o pedido ficou pronto antes da data prometida ao cliente, enquanto que a falta sugere que produção iniciou-se mais tarde do que deveria e faltou tempo para fabricar toda demanda.

Portanto devemos impor uma penalidade para a falta do produto e outra para o excesso do mesmo. Sendo assim, nosso objetivo é modelar o problema de maneira a minimizar as perdas causadas pelas penalidades ao longo de todos os períodos.

Definindo:

- n: número de tipos de produtos;
- T: número de períodos;
- d_{jt}: demanda do produto j no período t;
- l_{jt}: tamanho do lote do produto j no período t;
- p_j: penalidade por excesso de cada unidade do produto j;
- q_j: penalidade pela falta de cada unidade do produto j;
- d_{jt}^+: nível do produto j em excesso no período t;
- d_{jt}^-: nível do produto j em falta no período t;

Nesse problema as variáveis de decisão serão: y_{jt}, o número de rodadas de produção do produto j no período t (portanto inteira) e x_{jt}, o nível de produção do produto j no período t.

Podemos formular o problema de produção Just-in-time da seguinte maneira:

$$\min z = \sum_{j=1}^{n} \left(p_j \sum_{t=1}^{T} d_{jt}^+ + q_j \sum_{t=1}^{T} d_{jt}^- \right)$$

$$
s.a. \begin{cases} s_{j,t-1} + x_{jt} - d_{jt} = d_{jt}^{+} - d_{jt}^{-} & j = 1, \ldots, n;\ t = 1, \ldots, T \quad (13) \\ x_{jt} \leq l_{jt} y_{jt} & j = 1, \ldots, n;\ t = 1, \ldots, T \quad (14) \\ x_{jt} \geq l_{jt}(y_{jt} - 1) & j = 1, \ldots, n;\ t = 1, \ldots, T \quad (15) \\ y_{jt} \geq 0 \text{ e inteiro} & j = 1, \ldots, n;\ t = 1, \ldots, T \quad (16) \end{cases}
$$

A restrição (13) controla o nível de produção em cada período e de cada produto. Nela verificamos se o nível de estoque (resultante do excesso de produção do período anterior) mais a produção do atual período foi suficiente ou não para satisfazer a demanda do período. Como as variáveis d_{jt}^{+} e d_{jt}^{-} são não negativas, o lado direito da igualdade positivo indica que houve sobra na produção, enquanto que a negatividade indica que a demanda não foi atendida completamente, pois houve falta do produto, e a igualdade em zero reflete que a demanda foi atendida sem sobras de produção.

As restrições (14) e (15) asseguram a divisibilidade entre as variáveis. De acordo com a definição dos parâmetros e das variáveis, sabemos que $y_{jt} = \frac{x_{jt}}{l_{jt}}$. Como x_{jt} é uma variável contínua, y_{jt} uma varíavel inteira e l_{jt} uma constante inteira, se x_{jt} não for divisível por l_{jt}, y_{jt} não será um número inteiro. Por exemplo, suponha um nível de produção $x_{jt} = 1200$ e o tamanho de lote $l_{jt} = 350$, isso resultará em $y_{jt} = 3,43$ rodadas de produção, ou seja, três lotes de 350 e um de 150. Afim de arredondar esse número para cima, adicionamos as duas restrições. Nesse exemplo, a restrição (14) implica $y_{jt} \geq 3,43$ e a restrição (15) $y_{jt} \leq 4,43$, mas como y_{jt} é inteiro, teremos $y_{jt} = 4$.

2.3 Alocação de Trabalhadores/Tarefas

O problema de Alocação de Trabalhadores/Tarefas está presente em muitas empresas ou instituições, principalmente aquelas que funcionam 24h (hospitais, call-centers, departamento de polícia, entre outros), onde esse período é dividido em várias janelas de tempo T. Cada janela de tempo requer uma demanda mínima de trabalhadores (d_t) para realizar diferentes tarefas, porém cada trabalhador é escalado em m janelas de tempo consecutivas, desde que $m < T$, para o problema não se reduzir a um turno único. Os trabalhadores são pagos de acordo com a tarefa que eles foram designados, ou seja, para cada tarefa está associado um salário. A maneira de alocar os trabalhadores pode ser feita de duas formas: somente trabalhadores de tempo integral ou trabalhadores de meio período e tempo integral juntos. O objetivo do problema é designar os trabalhadores para as tarefas, atendendo a demanda de cada janela de tempo, de forma a minimizar as despesas com o salário.

2.3.1 Alocação de Trabalhadores em tempo integral

Neste caso são permitidos somente trabalhadores de tempo integral. O salário pago corresponde ao turno de m janelas de tempo consecutivas. Sendo assim, nossos parâmetros serão:

- n: número de trabalhadores a serem designados;
- T: número de janelas de tempo;
- k: número de total de tarefas;
- d_t: demanda mínima de trabalhadores na janela de tempo t;
- w_j: salário pago para a realização da tarefa j;

Lembrando que nem sempre em uma janela de tempo há k tarefas para serem realizadas. O número k refere-se a quantidade total de tarefas existentes naquela empresa ou instituição, portanto o número de tarefas a realizar em cada janela de tempo pode variar de 1 à k.

Já a variável de decisão, y_j, será o número de trabalhadores alocados para a tarefa j. Como estamos trabalhando somente com trabalhadores de tempo integral, devemos impor y_j inteiro e maior que zero. Sendo assim a formulação fica:

$$\min z = \sum_{j=1}^{k} w_j y_j$$

$$s.a. \begin{cases} \sum_{j=1}^{k} a_{jt} y_j \geq d_t & t = 1, \ldots, T \quad (17) \\ y_j \geq 0 \text{ e inteiro} & t = 1, \ldots, T \quad (18) \end{cases}$$

onde

$$a_{jt} = \begin{cases} 1, & \text{se é necessário executar a tarefa } j \text{ na janela de tempo } t \\ 0, & \text{caso contrário} \end{cases}$$

Na restrição (17) temos a desigualdade $\geq$ para garantir que a demanda de cada janela de tempo será atendida. Lembrando que as tarefas variam de uma janela de tempo para outra e que o trabalhador designado na tarefa j é o mesmo em todas as janelas de tempo t, seria difícil encontrar uma solução na igualdade, pois poderíamos cair em um sistema de equações sobredeterminado, por exemplo, dependendo dos valores de k e m.

2.3.2 Alocação de Trabalhadores em tempo integral e em meio período

Agora, permitiremos que sejam alocados trabalhadores de meio período durante as janelas de tempo t. Porém para que ele seja contratado, deve haver pelo menos um trabalhador de período integral presente. O valor pago aos trabalhadores de meio período será c_t.

Além desse novo parâmetro teremos uma nova variável de decisão, que determinará o número de trabalhadores de meio período necessários para cada janela de tempo t, e será denotada por x_t. De maneira análoga à variável y_j devemos impor que x_t seja inteiro e maior que zero.

O objetivo é minimizar os salários gastos com os dois tipos de trabalhadores, e atender a demanda mínima de trabalhadores em cada janela com ambos os tipos também. Podemos formular o problema de maneira parecida ao da seção anterior:

$$\min z = \sum_{j=1}^{k} w_j y_j + \sum_{t=1}^{T} c_t x_t$$

$$s.a. \begin{cases} \sum_{j=1}^{k} a_{jt} y_j + x_t \geq d_t & t = 1, \ldots, T \quad (19) \\ x_t \leq M \sum_{j=1}^{k} a_{jt} y_j & t = 1, \ldots, T \quad (20) \\ y_j \geq 0 \text{ e inteiro} & j = 1, \ldots, n \quad (21) \\ x_t \geq 0 \text{ e inteiro} & t = 1, \ldots, T \quad (22) \end{cases}$$

onde M é um número muito grande que podemos definir por $M = \sum_t d_t$.

A diferença em relação ao problema anterior é que na função objetivo adicionamos os gastos com os trabalhadores de meio período, e adicionamos a restrição (20) que garante a presença de um trabalhador de tempo integral para contratar um de meio período. Podemos ver isso lembrando que, se $y_j = 0$, então $x_t \leq 0$, porém em (22) $x_t \geq 0$, portanto $x_t = 0$. Já se $y_j = 1$, a restrição (20) se torna redundante, já que M é muito maior que x_t.

2.4 Problemas de Transporte com Custo Fixo e Distribuição

2.4.1 Transporte com Custo Fixo

O problema de Transporte com Custo Fixo consiste em transportar uma certa quantidade de produtos, partindo de diferentes lugares (as fontes de produção) em direção aos seus destinos de entrega (consumidores ou armazéns), de maneira a atender a demanda de cada um deles. Uma fonte pode transportar produtos para cada um dos destinos. Neste transporte há um custo fixo de se transportar a carga, independentemente da quantidade de produtos, mais um custo por unidade transportada. No nosso caso, assumimos que cada fonte é capaz de satisfazer a demanda de todos os destinos juntos.

Para uma visualização do problema, basta pensar em um grafo composto por $m + n$ nós e mn arcos, onde m é o múmero de fontes e n o de destinos. O número total de arcos (i, j) vem da definição acima do problema, pois de cada fonte i sai um arco que incide em cada um dos destinos j.

O objetivo do problema é minimizar os custos com o transporte (custo fixo e custo unitário) para satisfazer a demanda em todos os destinos. Os parâmetros nesse problema serão:

- m: número de fontes de produção;
- n: número de destinos;
- f_{ij}: custo fixo para se transportar produtos da fonte i até o destino j;
- c_{ij}: custo para se transportar um produto da fonte i até o destino j;
- d_j: demanda do destino j;

De posse dos parâmetros acima, resta saber quais as fontes irão suprir a demanda dos destinos, e com quantos produtos elas irão colaborar. Daí surgem as variáveis de decisão y_{ij} e x_{ij}. Neste caso, y_{ij} será uma variável binária que assumirá valor 1 se a fonte i atender o destino j, e 0 caso contrário. Já x_{ij} será a quantida de produtos transportados caso a fonte i atenda a demanda j, ou seja, o quanto será transportado caso $y_{ij} = 1$.

A formulação do problema será:

$$\min z = \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} x_{ij} + f_{ij} y_{ij})$$

$$s.a. \begin{cases} \sum_{i=1}^{m} x_{ij} = d_j & j = 1, \ldots, n \quad (23) \\ x_{ij} \leq M y_{ij} & i = 1, \ldots, m;\ j = 1, \ldots, n \quad (24) \\ x_{ij} \geq 0 & i = 1, \ldots, m;\ j = 1, \ldots, n \quad (25) \\ y_{ij} = 0 \text{ ou } 1 & i = 1, \ldots, m;\ j = 1, \ldots, n \quad (26) \end{cases}$$

onde M é um número muito grande, que nesse caso definimos como $M = \sum_j d_j$ para que este valor seja muito maior quando comparado com x_{ij}, possibilitando assim amarrar as variáveis x_{ij} e y_{ij}

A restrição (23) garante que a demanda de todos os nós será atendida, enquanto que as restrições (24) e (25) juntas, asseguram que se a fonte i não atende o destino j, então a quantidade de produtos transportada é zero, ou ainda, se $y_{ij} = 0$ então x_{ij} deve ser zero. Isso se verifica observando que se $y_{ij} = 0$, pela restrição (24) teremos $x_{ij} \leq 0$; mas por (25) sabemos que $x_{ij} \geq 0$, logo conclui-se que $x_{ij} = 0$. Mas se $y_{ij} = 1$, a restrição (24) se torna redundante e de (25) percebemos que haverá transporte, como era de se esperar já que $y_{ij} = 1$.

2.4.2 Localização de Armazéns Não Capacitado

A Localização de Armazéns Não Capacitado é parecido com o problema anterior, mas agora ao invés das fontes de produção temos m possíveis centros de distribuição para se construir e que enviarão produtos aos destinos do problema. Os destinos nesse caso serão as lojas revendedoras. Logo queremos saber quais localizações escolher para construir os centros.

Existe um custo fixo para se construir um centro de distribuição, e depois de construído ainda temos um custo para transportar cada produto do centro i até uma loja j. Sabendo disso, a escolha de qual centro construir é aquela que minimiza todos os custos e ainda atende a demanda de todas as lojas revendedoras.

Antes de formular o problema precisamos definir:

- m: número de localizações possíveis para a construção do centro de distribuição;
- n: número de lojas revendedoras;
- f_i: custo fixo de se construir o centro de distribuição na localização i;
- c_{ij}: custo para se transportar um produto do centro i até a loja j;

- d_j: demanda da loja j;

Só resta agora definir as variáveis de decisão: y_i variável binária representando a construção ou não do centro i; se sim, x_{ij} indica o quanto transportar do centro i para cada uma das lojas j. Eis a formulação:

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i$$

$$s.a. \begin{cases} \sum_{i=1}^{m} x_{ij} = d_j, & j = 1,\ldots,n \quad (27) \\ \sum_{j=1}^{n} x_{ij} \leq M y_i & i = 1,\ldots,m \quad (28) \\ x_{ij} \geq 0 \text{ e inteiro} & i = 1,\ldots,m;\ j = 1,\ldots,n \quad (29) \\ y_i = 0 \text{ ou } 1 & i = 1,\ldots,m \quad (30) \end{cases}$$

onde M é um número suficientemente grande, $M = \sum_j d_j$ por exemplo.

A restrição (27) garante que a demanda de todas as lojas são atendidas. Já (28) e (29) relacionam x_{ij} e y_i de modo que não haja transporte quando do centro i até os destinos j, se o centro i não foi construído. Notamos que se $y_i = 0$, pela restrição (28) a soma dos x_{ij} fixado um i é menor igual a zero. Mas de (29) sabemos que todos x_{ij} são maiores ou iguais a zero, portanto $x_{ij} = 0$.

2.4.3 Localização de Armazéns Capacitado

A diferença da Localização de Armazéns Capacitado para o não Capacitado, é que agora existe um limite de distribuição de cada centro i, denominado u_i. Portanto, devemos trocar M por u_i na restrição (28) da seção anterior. Assim ficamos com a seguinte formulação:

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i$$

$$
s.a. \begin{cases} \displaystyle\sum_{i=1}^{m} x_{ij} = d_j & j = 1, \ldots, n \quad (31) \\ \displaystyle\sum_{j=1}^{n} x_{ij} \leq u_i y_i & i = 1, \ldots, m \quad (32) \\ x_{ij} \geq 0 \text{ e inteiro} & i = 1, \ldots, m;\ j = 1, \ldots, n \quad (33) \\ y_i = 0 \text{ ou } 1 & i = 1, \ldots, m \quad (34) \end{cases}
$$

2.5 Transformação de variáveis não binárias para variáveis 0-1

Existem dois tipos de variáveis não binárias. As variáveis inteiras e as variáveis discretas. Variáveis inteiras são aquelas que assumem somente valores inteiros e consecutivos entre 0 e infinito, enquanto que nas variáveis discretas eles são inteiros e não consecutivos. Por exemplo, $k = \{0, 1, 2, \ldots\}$ representa variáveis inteiras, já $d = \{2, 7, 15, 19\}$ variáveis discretas.

Quando variáveis inteiras apresentam um valor mínimo, diferente de zero, adicionamos um limite inferior à variável ao modelarmos o problema. Já se o seu valor máximo é diferente de infinito, adicionamos um limite superior. Pode ocorrer da variável ser limitada superiormente e inferiormente.

2.5.1 Transformação de variáveis inteiras

Toda variável inteira $x \geq 0$ limitada superiormente, pode ser representada por uma soma de variáveis binárias da seguinte forma:

$$x = 2^0 y_0 + 2^1 y_1 + 2^2 y_2 + \ldots + 2^k y_k$$

onde $k + 1$ é o número de variáveis binárias (y_k) necessárias.

Por exemplo, se x é uma variável inteira que pode assumir os valores $x = 0, 1, 2, \ldots, 35$, podemos representá-la por:

$$
\begin{aligned} x &= 2^0 y_0 + 2^1 y_1 + 2^2 y_2 + 2^3 y_3 + 2^4 y_4 + 2^5 y_5 \quad (35) \\ &= 1y_0 + 2y_1 + 4y_2 + 8y_3 + 16y_4 + 32y_5 \quad (36) \end{aligned}
$$

Quando $x = 25$, então $y_0 = y_1 = y_2 = y_5 = 0$ e $y_3 = y_4 = 1$. No entanto esta representação para x não se aplica somente para as variáveis inteiras com valores de 0 à 35, mas sim, de 0 à 63. Para ver isso, basta somar os coeficientes da equação (36).

Resta saber como determinar o valor de k. Dada uma variável inteira $x \geq 0$ e denotando-se por u o seu limite superior, deseja-se encontrar o valor de k, de tal forma que u esteja dentro do intervalo de duas potências de 2 consecutivas:

$$2^k \leq u < 2^{k+1}$$

Tomando-se $\log_2$ na equação acima, temos:

$$k \leq \log_2 u < k + 1 \tag{37}$$

onde k e $k+1$ são os inteiros obtidos arredondando para cima ou para baixo o valor de $\log_2 u$. No exemplo dado, o limite superior é $u = 35$. Como $\log_2 35 = 5,13$, da desigualdade (37) resulta-se o par de inequações $k \leq 5,13$ e $k > 5,13 - 1$. Logo $k = 5$, uma vez que k é inteiro. Portanto o número de variáveis binárias necessárias é $k + 1 = 6$, pois a representação inicia-se com índice 0.

Pode-se fazer uma extensão para o caso de transformação de limites de uma variável. Suponha uma variável inteira com limite inferior b diferente ou menor que zero, e um limite superior u finito:

$$b \leq x \leq u$$

Subtraindo-se b da desigualdade:

$$0 \leq x - b \leq u - b$$

e denotando-se $x' = x - b$ e $u' = u - b$ temos:

$$0 \leq x' \leq u'$$

voltando ao caso de variáveis inteiras não negativas. No entanto o número de variáveis do problema aumenta conforme o tamanho de u. Essa transformação é vantajosa quando temos poucas variáveis inteiras , cada uma com limite superior baixo, ou quando o algoritmo 0-1 é mais eficiente que o algoritmo para variáveis inteiras.

2.5.2 Transformação de variáveis discretas

Se uma variável discreta assume um número finito de valores, digamos k, podemos representá-la por um conjunto de variáveis binárias. Considerando a variável discreta $d = \{2, 7, 15, 19, 28\}$, podemos representá-la por:

$$d = 2y_1 + 7y_2 + 15y_3 + 19y_4 + 28y_5 \tag{38}$$
$$y_1 + y_2 + y_3 + y_4 + y_5 = 1 \tag{39}$$
$$y_i = 0 \text{ ou } 1 \text{ para } i = 1, 2, \ldots, 5 \tag{40}$$

onde y_i é uma variável binária que tem valor 1 quando d assume o valor do coeficiente associado à ela; e zero caso contrário. A restrição (39) garante que apenas uma y_i será igual a 1, enquanto que as demais serão nulas.

2.6 Transformação de funções lineares por partes

Uma função linear por partes é uma função dividida em subintervalos, onde cada intervalo apresenta uma inclinação diferente. Formalmente:

Seja uma função $f(x)$ definida no intervalo $a_1 \leq x \leq a_{r+1}$, a função $f(x)$ é dita linear por partes se pode ser subdivida em intervalos, onde cada intervalo $f(x)$ é afim tal que:

$$f(x) = b_i + s_i x \text{ para cada } a_i \leq x \leq a_{i+1}, \quad i = 1, 2, \ldots, r$$

onde s_i é a inclinação e b_i é o coeficiente linear. Já os a_i são denominados os *breakpoints* da função, os pontos onde muda a inclinação da curva, excetuando-se os extremos do intervalo onde $f(x)$ está definida.

Essas funções são comumente usadas na venda de produtos em atacado, onde quanto maior a quantidade comprada, menor o preço unitário. Tomemos como exemplo um determinado produto, em que as primeiras 200 unidades custam 15 reais, as próximas 300 unidades 10 reais e as outras 500 custam 7 reais cada.

A função de custo será $f(x) = 15x$ para as primeiras 200 unidades. Já para o próximo intervalo, há o custo das primeiras 200 unidades $(15 * 200)$ mais o custo de 10 reais para as unidades excedentes de 200, ou matematicamente, $f(x) = 15*200+10(x-200)$. Analogamente para o terceiro intervalo: $f(x) = 15 * 200 + 10 * 300 + 7(x - 500)$. Sendo assim obtem-se a seguinte função de custo:

$$f(x) = \begin{cases} 15x, & \text{se} \quad 0 \leq x \leq 200 \\ 10x + 1000, & \text{se} \quad 200 \leq x \leq 500 \\ 7x + 2500, & \text{se} \quad 500 \leq x \leq 1000 \end{cases}$$

2.6.1 Transformação de funções lineares por partes arbitrárias

Considerando uma função linear por partes $f(x)$ e um ponto x pertencente ao intervalo onde a função está definida, sabemos pela definição de $f(x)$ que x está situado entre dois *breakpoints* consecutivos, a_k e a_{k+1}. Logo ele pode ser escrito como combinação linear deles:

$$x = \lambda_k a_k + (1 - \lambda_k) a_{k+1}$$

onde $0 \leq \lambda_k \leq 1$. Como $f(x)$ é linear em cada subintervalo da função, então:

$$f(x) = \lambda_k f(a_k) + (1 - \lambda_k) f(a_{k+1})$$

Generalizando a idéia para todos os breakpoints:

$$x = \lambda_1 a_1 + \lambda_2 a_2 + \ldots + \lambda_{r+1} a_{r+1}$$

$$f(x) = \lambda_1 f(a_1) + \lambda_2 f(a_2) + \ldots + \lambda_{r+1} f(a_{r+1})$$

onde $\lambda_1 + \lambda_2 + \ldots + \lambda_{r+1} = 1$, $\lambda_k \geq 0$ para todo k e no máximo dois λ_k adjacentes podem ser positivos. Esta última condição se deve ao fato de que se x está situado entre os *breakpoints* a_k e a_{k+1}, então apenas λ_k e λ_{k+1} serão positivos, enquanto que os demais serão iguais a zero. No entanto esta condição precisa ser formulada matematicamente. Para isso introduzimos a variável binária y_k para controlar os valores de λ_k e λ_{k+1}. O conjunto de restrições será:

$$\lambda_1 \leq y_1 \tag{41}$$
$$\lambda_2 \leq y_1 + y_2 \tag{42}$$
$$\lambda_3 \leq y_2 + y_3 \tag{43}$$
$$\vdots$$
$$\lambda_r \leq y_{r-1} + y_r \tag{44}$$
$$\lambda_{r+1} \leq y_r \tag{45}$$
$$\sum_{k=1}^{r} y_k = 1 \tag{46}$$
$$\lambda_k \geq 0 \text{ para todo } k \tag{47}$$
$$y_k = 0 \text{ ou } 1 \text{ para todo } k \tag{48}$$

Se $y_k = 0$, ou seja, x não está entre a_k e a_{k+1}, então λ_k e/ou λ_{k+1} devem ser zero, pois:

- se x estiver entre a_{k+1} e a_{k+2}, então λ_{k+1} e λ_{k+2} serão positivos e os demais serão iguais a zero;
- se x estiver entre a_{k-1} e a_k, então λ_{k-1} e λ_k serão positivos e os demais serão iguais a zero;
- se x não estiver entre nenhum dos intervalos acima, então λ_k e λ_{k+1} serão iguais a zero;

Já se $y_k = 1$ (x está entre a_k e a_{k+1}), então λ_k e λ_{k+1} serão positivos entre 0 e 1.

A restrição (46) garante que apenas um y_k seja igual a 1, enquanto que as restrições de (41) à (45) junto com o fato da soma dos λ_k ser igual a ser 1, implicarão em dois λ_k com valores entre 0 e 1, e os demais nulos.

A função linear por partes dada como exemplo neste texto, seria convertida em:

$$
\begin{aligned}
& x = 0\lambda_1 + 200\lambda_2 + 500\lambda_3 + 1000\lambda_4 \\
& f(x) = 0\lambda_1 + 3000\lambda_2 + 6000\lambda_3 + 9500\lambda_4 \\
& \lambda_1 \leq y_1 \\
& \lambda_2 \leq y_1 + y_2 \\
& \lambda_3 \leq y_2 + y_3 \\
& \lambda_4 \leq y_3 \\
& y_1 + y_2 + y_3 = 1 \\
& \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \\
& \lambda_k \geq 0 \text{ para todo } k \\
& y_k = 0 \text{ ou } 1 \text{ para todo } k
\end{aligned}
$$

2.7 Transformação de funções com produto entre variáveis binárias e variáveis contínuas: Problema do pacote de preços

Se na função objetivo de um modelo de programação inteira mista, existir um produto de variável binária com variável contínua, podemos transformá-lo em um conjunto de restrições lineares. Um exemplo desse caso ocorre no problema do Pacote de Preços, cuja formulação é explicada a seguir:

O problema do Pacote de Preços consiste em determinar o preço de venda dos produtos de uma loja, sejam eles vendidos separadamente ou em pacote de produtos, de tal maneira que o lucro seja maximizado. No caso de n

produtos teremos $2^n - 1$ possibilidades de pacotes de produtos, porém na prática apenas uma pequena parte das possibilidades são utilizadas.

Na formulação consideramos que existem diversos tipos de consumidores que podem comprar o produto, e existe uma demanda esperada para cada classe de consumidor. Um consumidor compra apenas uma opção de pacote ou não compra nada. Para ajudar na elaboração dos preços, realiza-se uma pesquisa para saber o máximo que o consumidor está disposto a pagar por cada opção de pacote.

Definimos como *preço de reserva*, o máximo que um consumidor está disposto a pagar pelo produto; e *consumer surplus* como a diferença entre o *preço de reserva* e o preço de venda. Assumimos que o consumidor escolhe o produto que maximiza o *consumer surplus*. Sendo assim, os parâmetros do problema serão:

- k: número de tipos de consumidores;
- m: número de pacotes de produtos;
- n_i: número de consumidores para cada tipo de consumidor i;
- r_{ij}: *preço de reserva* para o consumidor i para o pacote j;
- s_i: maior lucro do consumidor i;

As variáveis de decisão serão: x_j, preço de venda para o pacote j; e y_{ij}, variável binária que assume valor 1 se o consumidor i compra o produto j, e 0 caso contrário. Nota-se que o preço de venda depende apenas do índice j, ou seja, o preço de venda do pacote j é o mesmo para todos os tipos de consumidores.

Eis a formulação do problema do Pacote de Preços:

$$\max \sum_{i=1}^{k} \left(n_i \sum_{j=1}^{m} y_{ij} x_j \right)$$

$$s.a. \begin{cases} \sum_{i=1}^{k} y_{ij} = 1 & i = 1, \ldots, k \quad (49) \\ s_i = \sum_{j=1}^{m} (r_{ij} - x_{ij}) y_{ij} & i = 1, \ldots, k; \quad (50) \\ s_i \geq r_{ij} - x_{ij} & i = 1, \ldots, k;\ j = 1, \ldots, m \quad (51) \\ y_{ij} = 0 & i = 1, \ldots, k;\ j = 1, \ldots, m \quad (52) \\ x_j \geq 0 & j = 1, \ldots, m \quad (53) \end{cases}$$

Nota-se a presença do termo $y_{ij}x_j$ na função objetivo, que pode ser linearizado substituindo-o por z_{ij} e fazendo as seguintes modificações:

$$\max \sum_{i=1}^{k} \left(n_i \sum_{j=1}^{m} z_{ij} \right)$$

$$s.a. \begin{cases} \sum_{j=1}^{m} (r_{ij} - x_{ij}) y_{ij} + x_j \geq r_{ij} & i = 1, \ldots, k; \quad (54) \\ z_{ij} \leq x_j & i = 1, \ldots, k;\ j = 1, \ldots, m \quad (55) \\ z_{ij} \leq r_{ij} y_{ij} & i = 1, \ldots, k;\ j = 1, \ldots, m \quad (56) \\ z_{ij} \geq x_j - (1 - y_{ij}) M_j & j = 1, \ldots, m \quad (57) \end{cases}$$

onde M_j é um limite superior para x_j.

2.8 Transformação de restrições não simultâneas

Ao formularmos um problema de programação linear inteira mista, devemos verificar se todas as restrições estão sendo satisfeitas simultaneamente. Caso isso não ocorra, ou seja, as restrições são não simultâneas, então devemos convertê-las em simultâneas. Alguns exemplos são ilustrado nas próximas seções.

2.8.1 Restrições Ou/Ou

Uma variável pode estar definida em duas regiões distintas, como no caso de estar fora de um intervalo real. Por exemplo, uma variável x que está definida fora do intervalo $(7, 13)$, ou seja, podemos ter tanto $x \leq 7$, quanto $x \geq 13$. Para converter em restrições simultâneas, reescrevemos o par de inequações como:

$$\begin{aligned} x - 7 &\leq 0 \\ -x + 13 &\leq 0 \end{aligned}$$

Denotando M por: $M \geq \max\{x - 3, -x + 10\}$ de tal forma que ele seja um número muito grande; e definindo y como uma variável binária, podemos transformar as duas restrições disjuntas em simultâneas fazendo:

$$x - 7 \leq My \quad (58)$$

$$-x + 13 \leq M(1 - y) \quad (59)$$

Se $y = 1$, teremos $-x + 13 \leq 0$, ou $x \geq 13$, enquanto que a restrição (58) é redundante. Já se $y = 0$, então $x - 7 \leq 0$, ou seja, $x \leq 7$, ao passo que agora a restrição (59) é redundante.

Um outro exemplo de restrições não simultâneas é o de alocação de trabalhadores em uma única máquina. Seja x_i e x_j o tempo de início do trabalho i e j respectivamente; e t_i e t_j o tempo de duração dos trabalhos i e j, então o tempo que denota o fim da tarefa será $x_i + t_i$ e $x_j + t_j$ para as tarefas i e j, respectivamente. Para iniciar um trabalho na máquina é necessário que ela esteja livre, porém se já estiver sendo realizado algum trabalho nela, precisa-se esperar o fim do mesmo, ou seja, dois trabalhos não podem ser alocados durante o mesmo intervalo de tempo. Devemos então satisfazer as duas restrições não simultâneas:

$$x_i + t_i \leq x_j \tag{60}$$
$$x_j + t_j \leq x_i \tag{61}$$

A primeira diz que o trabalho j não pode se iniciar antes do fim do trabalho i, enquanto que a segunda não permite o trabalho i começar antes do término do trabalho j. Reescrevendo as restrições, temos:

$$x_i - x_j + t_i \leq 0 \tag{62}$$
$$x_j - x_i + t_j \leq 0 \tag{63}$$

Assim como no exemplo anterior, introduzimos um número muito grande M e uma variável binária y, transformando as restrições em simultâneas da seguinte maneira:

$$x_i - x_j + t_i \leq My \tag{64}$$
$$x_j - x_i + t_j \leq M(1 - y) \tag{65}$$

Se $y = 0$ a restrição (64) mostra que a tarefa i é realizada antes da tarefa j, enquanto que a restrição (65) é redundante. De maneira análoga, se $y = 1$, então a tarefa j antecede a tarefa i e a restrição (64) é redundante.

2.8.2 p restrições, num total de m, devem ser satisfeitas

Em um modelo com m restrições, onde apenas p ($p < m$) são necessárias, podemos selecionar qualquer combinação de p restrições de tal forma que otimize a função objetivo do modelo. As $m - p$ restrições que não foram

selecionadas podem ser reduzidas impondo restrições redundantes do tipo: $f_i(x) - b_i \leq M$, onde M é um número muito grande. Introduzindo uma variável binária y_i, que tem valor 1 se a restrição i é selecionada e 0 caso contrário. Podemos reduzir o problema da seguinte maneira:

$$f_i(x) - b_i \leq M(1 - y_i) \;\; i = 1, 2, \ldots, m \tag{66}$$

$$\sum_{i=1}^{m} y_i = p \tag{67}$$

$$y_i = 0 \text{ ou } 1 \text{ para todo } i \tag{68}$$

A restrição (67) garante que serão selecionadas apenas p restrições, enquanto que (66) implica $f_i(x) \leq b_i$ se a restrição i é selecionada ($y_i = 1$) ou se torna redundante se $y_i = 0$.

2.8.3 Conjuntos de restrições disjuntos

Dois subconjuntos de restrições são disjuntos quando: ou um conjunto ou outro deve ser satisfeito, mas não ambos. Dado dois subconjuntos disjuntos, podemos transformá-los em restrições simultâneas. Considerando os dois subconjuntos:

$$\{a_i^T x - b_i \leq 0, \;\; i = 1, 2, \ldots, m_1\} \tag{69}$$

$$\{c_i^T x - d_i \leq 0, \;\; i = 1, 2, \ldots, m_2\} \tag{70}$$

podemos definir um número muito grande e uma variável binária y para tornar as restrições simultâneas:

$$a_i^T x - b_i \leq My, \;\; i = 1, 2, \ldots, m_1 \tag{71}$$

$$c_i^T x - d_i \leq M(1 - y), \;\; i = 1, 2, \ldots, m_2 \tag{72}$$

2.8.4 Negação de uma restrição

Suponha a restrição $f(x_1, x_2, \ldots, x_n) = b$, ou seja, $f(x) = b$, onde b é uma constante. Reescrevendo a restrição como $f(x) - b \leq 0$, a negação desta restrição será:

$$f(x) - b > 0 \tag{73}$$

ou ainda:

$$-f(x) + b < 0 \tag{74}$$

resultado obtido multiplicando por -1 ambos os lados da desigualdade (73).

2.8.5 Restrições Se/Então

Podemos ver cada restrição de um problema como uma afirmação. Da teoria de lógica, sabemos que dada duas afirmações A e B:

" Se A então B é equivalente à (não A) $\cup$ B "

Sendo assim, as restrições $f_1(x) - b_1 \leq 0$ e $f_2(x) - b_2 \leq 0$ podem ser vistas como as afirmações A e B respectivamente. Como a negação de A é $-f_1(x) + b_1 < 0$, a simples implicação:

Se $f_1(x) - b_1 \leq 0$ então $f_2(x) - b_2 \leq 0$

é equivalente a:

Ou $-f_1(x) + b_1 < 0$ ou $f_2(x) - b_2 \leq 0$

ou seja, voltamos ao caso das restrições Ou/Ou, que podem ser transformadas em simultâneas da seguinte maneira:

$$-f_1(x) + b_1 \leq My \tag{75}$$
$$f_2(x) - b_2 \leq M(1 - y) \tag{76}$$

onde M é um número muito grande e y é uma variável binária.

3 Algoritmos

Nesta seção veremos o *branch-and-bound*, um método exato de solução, além dos conceitos e classificação das heurísticas. Exibiremos dois exemplos: GRASP e o Algoritmo Colônia de Formigas.

3.1 Métodos clássicos de solução

Um Problema de Programação Linear Inteira com variáveis limitadas pode ser resolvido por enumeração, ou seja, enumeramos todas as soluções possíveis e escolhemos, dentre as factíveis, aquela que minimiza ou maximiza (dependendo do tipo de problema) a função objetivo. Dado um problema com n variáveis inteiras, cada uma podendo assumir m valores, temos um total de m^n possibilidades de soluções, factíveis ou não.

Em problemas de grande escala isso requer um esforço computacional muito grande. Para resolver isso utilizamos um método de enumeração implícita, que permite encontrar a solução ótima do problema sem ter que avaliar todas as soluções. Tal método é baseado em dois princípios básicos: *dividir* e *conquistar*, pois dividimos o problema original em subproblemas simples de se resolver (conquistar), e a partir das soluções desses subproblemas obtemos a solução do problema original.

3.2 *Branch-and-Bound*

O *Branch-and-Bound* é um método de resolver Problemas de Programação Linear Inteira (PI), seja ele um PI puro (somente variáveis inteiras), PI misto (variáveis inteiras e lineares) ou PI binário (variáveis binárias e inteiras). Trataremos nesse texto do caso puro com função objetivo de maximizar.

Ele se baseia no método de enumeração implícita, por isso o termo *branch* (ramificar), está associado ao fato de dividirmos o problema original em subproblemas (processo de ramificação). Já o termo *bound* (limite), refere-se ao valor limite da função objetivo de cada subproblema. Definimos um limite superior e um inferior em cada ramificação, que permitirá avaliarmos somente uma determinada quantidade de soluções do problema original.

A divisão do problema original é obtida a partir da relaxação das variáveis inteiras para variáveis lineares, seguida da imposição de inequações simples, de tal forma que tenhamos dois subproblemas, onde a região de factibilidade inteira dos dois juntos, é igual a do problema original. Esse processo de ramificação continua a se repetir nos outros subproblemas, de forma a diminuir a região de factibilidade em cada divisão, até se encontrar uma solução inteira, uma solução pior que uma já existente ou cairmos em um subproblema infactível.

A representação do processo de solução do Branch-and-Bound é feita através de uma árvore, onde cada nó indica a solução de um subproblema. A ele estão associados o limite inferior e o limite superior. Iniciamos a árvore com o nó raiz, que representa a solução do PI relaxado. Se esta solução for inteira, então ela é a solução do PI. Caso contrário o valor da função

objetivo da solução se torna o limite superior inicial, e iniciamos o processo de ramificação.

Devemos escolher uma variável, entre as que não possuem valor inteiro, para ramificar a partir dela. Nessa escolha seguimos algum critério, ele pode ser:

- a variável com valor fracionário mais próximo de 0.5;
- a variável com maior impacto na função objetivo;
- a variável com o menor índice;

Não se pode afirmar qual critério é melhor, o desempenho de cada um depende do problema ao qual ele está sendo aplicado.

Ramificado o nó raiz, temos dois nós a serem explorados. Novamente, a decisão de qual examinar primeiro, deve seguir uma estratégia de busca. As mais conhecidas são:

- busca em profundidade: explora o nó gerado mais recente;
- busca em largura: só passa a explorar os nós de um nível, depois de explorar todos do nível anterior;
- busca por melhor limite (*bound*): explora o nó com melhor limite superior;

Aqui a melhor estratégia também depende do tipo de problema, cada uma tem suas vantagens e desvantagens. Na busca em profundidade, por exemplo, tentamos encontrar uma solução rapidamente, para que nas próximas ramificações deixemos de examinar alguns nós. Já na busca em largura, examinamos muitos nós e a árvore vai crescendo rapidamente em largura.

Escolhido qual nó será examinado, devemos decidir se ramificaremos ou não. Como mencionado acima, existem casos em que devemos parar a ramificação, nesses casos dizemos que o nó será "cortado" ou "podado" (por se tratar de uma árvore). As três situações em que cortamos o nó são:

- cortar por infactibilidade: o subproblema não possui solução factível;
- cortar por otimalidade: o subproblema tem solução inteira, logo é um candidato a solução do problema original;
- cortar por limite (bound): o limite superior do subproblema é menor ou igual ao limite inferior do problema original;

Depois de avaliado todos os nós possíveis, escolhemos o nó com solução inteira que proporciona o maior valor na função objetivo.

Consideremos o exemplo a seguir e sua representação gráfica da solução.

$$\max z = y_1 + 3y_2$$

$$s.a. \begin{cases} y_1 + 5y_2 \leq 12 \\ y_1 + 2y_2 \leq 8 \\ y_1, y_2 \geq 0 \text{ e inteiro} \end{cases}$$

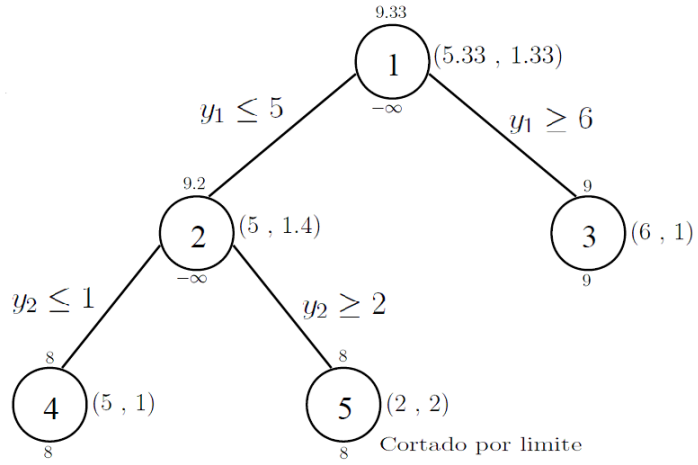

Figura 1: Árvore Branch-and-Bound.

Os números dentro dos nós representam a ordem em que eles foram ramificados. A solução de cada nó está indicada ao lado entre parênteses. Os valores acima e abaixo de cada nó são respectivamente, o limite superior e o limite inferior. As inequações ao lado dos ramos mostra a partir de qual variável o nó foi ramificado.

O nó 5 foi cortado por limite, pois o limite superior dele, 8, é igual ao limite inferior do nó 4.

Olhando na árvore, percebemos que temos dois candidatos a solução: os nós 3 e 4. Como o valor da função objetivo é maior no ponto (6 , 1) do que em (5 , 1), temos que a solução do problema original é (6 , 1) com função objetivo $z = 9$.

3.3 Heurísticas

Os métodos exatos como *branch-and-bound* podem resultar em tempos excessivos e até proibitivos. Por esta razão, os métodos heurísticos têm sido propostos e aplicados a problema de programação inteira. A desvantagem é que não há garantia de obtenção da solução ótima. As heurísticas são classificadas em: *heurísticas construtivas* e *heurísticas de busca*.

Nas *heurísticas construtivas* definimos os elementos que compõe a solução do problema (itens da mochila, trabalhadores, armazéns) e os classificamos seguindo algum critério (lucro por item, salário a ser pago, custo de instalação). Os elementos mais promissores são acrescentados à solução de modo a atender as restrições.

Já as *heurísticas de busca* são classificadas em:

- *Construção por soluções repetidas*: construímos uma nova solução a partir do zero. Podemos fazer isso de duas maneiras: busca aleatória ou baseada na memória. Na busca aleatória criamos uma lista de candidatos a serem incluídos seguindo algum critério e escolhemos um elemento aleatoriamente na lista, e assim sucessivamente até que não seja mais possível não violar as restrições. Na busca baseada na memória, geramos um conjunto de soluções baseadas na probabilidade de cada elemento aparecer na solução. Essas probabilidades são obtidas atribuindo pesos às informações heurísticas (classificação por lucro, classificação por salário, classificação por custo) e a memória de iterações passadas (que levam em conta não só quantas vezes o elemento apareceu na solução, mas se a solução em que ele apareceu era de alta qualidade ou não). Após várias iterações, observamos um aumento na probabilidade de alguns elementos e a quase extinção de outros.

- *Modificação por soluções repetidas*: a partir de uma solução, buscamos melhorá-la adicionando ou removendo itens, a fim de chegarmos próximos da solução ótima. Porém este procedimento pode levar a *dead-ends* (quando chegamos em uma iteração onde não conseguimos mais melhorá-la) ou ciclos (voltando sempre a uma certa solução após algumas iterações). Podemos ficar travados em boas soluções, mas se não ficarmos ou quisermos melhorar ainda mais uma solução boa, precisamos permitir deteriorações na solução, ou seja, permitimos uma queda na qualidade da solução para que em iterações futuras possamos obter um resultado melhor.

- *Recombinação por soluções repetidas*: combinamos duas soluções ou mais para obtermos uma solução melhor, mas tomando o cuidado de

não torná-la infactível. Depois de gerar essas combinações, começamos a fazer recombinação e assim sucessivamente até convergirem para uma solução.

3.3.1 GRASP

O GRASP é um método de busca construtiva, ou seja, partindo de uma solução vazia, constrói-se uma solução adicionando elementos a fim de melhorar a solução parcial, até que não seja mais possível melhorá-la. Porém ele é um método mais aprimorado que os outros da mesma classe.

Enquanto outros métodos preocupam-se somente em escolher sempre o melhor elemento (segundo algum critério) em cada iteração, o GRASP utiliza uma lista de candidatos a serem adicionados, e escolhe um aleatoriamente entre eles. O que faz a qualidade da solução variar.

As etapas de construção do GRASP são:

- *Avaliação dos candidatos*: associa-se um valor para cada um dos elementos que ainda não foram adicionados, considerando a influência deles na qualidade da solução. Estabelecendo assim, o máximo (g_{max}) e o mínimo (g_{min}), entres eles.

- *Lista restrita de candidatos (RCL)*: elabora-se a lista de possíveis elementos a serem adicionados, isso pode ser feito de duas maneiras:

  - Baseado na Cardinalidade: inclui os k melhores candidatos da lista, onde k é um número definido durante a programação do algoritmo.
  - Baseado em Valor: selecionamos os candidatos pertencentes ao intervalo $[g_{min}, \mu]$, onde μ é obtido pela fórmula:

    $$\mu = g_{min} + \alpha(g_{max} - g_{min})$$

    onde α é um valor entre 0 e 1 a ser definido pelo programador. Logo o tamanho da lista depende do valor de μ, que depende do parâmetro α adotado. Se escolhermos $\alpha = 0$, a construção se torna gulosa, ou seja, são escolhidos sempre os melhores elementos. Já se escolhemos $\alpha = 1$, a construção é simplesmente aleatória. O tamanho da lista tem papel fundamental na performance do algoritmo.

- *Escolha aleatória do elemento*: seleciona-se um elemento da RCL para ser adicionado na solução parcial.

Repetimos estes três passos até não ser mais possível adicionar elementos. Obtida a solução, usamos algum método específico para melhorar essa solução.

3.3.2 Algoritmo Colônia de Formigas

O Algoritmo foi baseado, como o próprio nome já diz, no comportamento das formigas. Nele observou-se que quando há um grande obstáculo no caminho das formigas entre um ninho e uma fonte de alimento, e que para contornar este obstáculo existe uma rota mais curta e outra mais longa, as formigas conseguem encontrar o caminho mais curto.

A explicação é que, no início do processo de busca as formigas tem probabilidade igual de escolher qualquer um dos caminhos, porém durante esse caminho elas depositam uma substância química chamada feromônio, que induz outras formigas a percorrerem o mesmo caminho. Quanto maior a taxa dessa substância, maior a probabilidade de elas escolherem o caminho. Como elas depositam em ambas as direções, essa taxa vai aumentando no menor caminho, pois as formigas que estão voltando dele depositam mais cedo do que as outras que seguiram pelo maior caminho. Porém esse feromônio se evapora com o tempo, diminuindo a quantidade presente no chão, o que favorece na escolha do menor caminho.

Baseado neste estudo, surge o Algoritmo Colônia de Formigas (ACO): Dado um grafo com N nós e arcos (i,j) associados a distância entre o nó i e j, o ACO tenta simular o comportamento descrito acima com formigas artificiais e atribuindo valores aos arcos (como se fossem o feromônio) a fim de encontrar o menor caminho.

Cada nó representa um possível elemento da solução e os arcos são responsáveis por ligar os elementos que farão parte da solução. Os valores atribuídos aos arcos que ligam um determinado nó ao demais, correspondem ao peso associado a qualidade que o próximo nó acrescentará na solução. O menor caminho encontrado é a melhor solução encontrada na iteração.

Inicialmente atribui-se as mesmas taxas de feromônio artificiais em todos arcos, e projeta a "formiga"em um nó para gerar um caminho. Quando há mais de uma aresta partindo de um nó, a escolha é feita baseada em uma probabilidade de escolher um certo caminho, através da seguinte fórmula:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{M}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}$$

Onde τ_{ij} é a taxa de feromônio, η_{ij} é um valor que envolve alguma informação heurística, α e β são parâmetros que definem o impacto do fe-

romônio e da informação heurística no cálculo da probabilidade e M são os índices dos nós não visitados.

Quando a formiga voltar ao nó de partida, teremos uma solução construída, e a cada iteração novas soluções são obtidas, ao passo que a taxa de feromônio é atualizada durante as iterações também. A escolha da etapa na qual a atualização será realizada durante a execução do algoritmo, pode ocorrer de diversas maneiras, por exemplo:

- a cada arco que a formiga percorre;
- ao final da construção de uma solução;
- nos arcos da melhor solução de cada iteração.

Após um conjunto de soluções obtidas, atualizamos a melhor solução. Além de adicionar a taxa de feromônio, também temos que descontar a taxa que se evapora, que pode ser feito pela fórmula:

$$\tau_{ij} = (1 - \rho)\tau_{ij}$$

onde ρ é a taxa de evaporação. Isso permite uma maior diversificação na busca pelo menor caminho. Esse procedimento de gerar novas soluções é feito até que a solução atualizada satisfaça algum critério.

Referências

[1] G. Zäpfel, R. Braune & M. Bögl, Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics, Springer, 2010.

[2] Chen Der-San, R. G. Batson & Dang Y.,Applied integer programming: modeling and solution, John Wiley & Sons, 2010.