Projeto de Otimização Inteira

Luiz Filipe Moraes, 112229
Lucas Eduardo Nogueira Gonçalves, 122055

02 de março de 2021

1 Introdução

Chamamos de Programação Matemática a área que estuda problemas de otimização, estes problemas estão relacionados com a maximização, ou minimização de valores objetivos de acordo com algumas restrições. De modo geral, ao tratar desta denominação, estamos nos referindo uma ampla classe de problemas. Além disso, o termo otimização, ou programação matemática, refere-se ao estudo de uma função através da escolha sistemática dos valores de variáveis reais ou inteiras dentro de um conjunto viável.

Como citado por Adriana Cristina e Andéa Carla no material disponibilizado para o minicurso realizado na $XXIV$ Semana de Licenciatura em Matemática em Bauru-SP: *Sempre que encurtamos um caminho para ganhar tempo, economizamos para comprar algo, tomamos decisão com base em investimentos, estamos interessados na melhor forma de aplicarmos nossos recursos. Resolver um problema de otimização, significa procurar a solução de um problema de forma a se maximizar algo ou a minimizar algo. Uma infinidade de problemas do cotidiano podem ser classificados como problemas de Otimização Combinatória.*

De acordo com a *NCSS Statistical Software*, temos que, enquanto a programação linear maximiza (ou minimiza) uma função objetivo linear sujeita a uma ou mais restrições, a programação inteira mista adiciona uma condição adicional de que pelo menos uma das variáveis só pode assumir valores inteiros.

2 Métodos

Como foco principal para este trabalho vamos trabalhar com dois métodos de otimização que estão fortemente conectados, o método de *Branch and Bound* e o *Cutting Plane*.

2.1 Branch and Bound

Introduzido por Land e Doig em *"An Automatic Method of Solving Discrete Programming Problem"*, o "*branch and bound*" se refere a ideia de uma enumeração de possíveis soluções candidatas a solução ótima para um problema que se espera como resultado coeficientes inteiros. De modo geral, é efetuado sucessivas partições da região factível, onde se encontra as possíveis soluções, cortando a árvore de pesquisa através da consideração de limites calculados ao longo da enumeração. Em um primeiro momento o algoritmo faz a partição da região viável do problema onde ocorreram as explorações. Para cada partição gera-se sub-problemas de tal forma que possibilite a exploração sistemática do algoritmo encontrando soluções (candidatas) que resolvem o problema e outras que não o favorecem (usualmente com soluções não viáveis), assim, ao momento de terminar todas as explorações tem-se entre as soluções candidatas a solução ótima do problema inicial.

De modo geral, como uma generalização para este algoritmo, se baseando no texto de Ronaldo Rust, Um algoritmo Branch and Bound para resolução de problemas em localização capacitados, temos as seguintes condições:

Considerando um problema do tipo

$$\begin{aligned} &\max_{x \in \mathbb{Z}^n} c^T x \\ &s.a.\ \ Ax \leq b \\ &x \geq 0. \end{aligned}$$

1. Resolvemos o relaxamento linear do problema. Se a solução for inteira, então terminamos. Caso contrário, criamos dois novos subproblemas ramificando em uma variável fracionária.
2. Um subproblema não está ativo quando ocorre um dos seguintes:
   - Utilizamos o subproblema para ramificar;
   - Todas as variáveis na solução são inteiras;
   - O subproblema é inviável;
   - Podemos compreender o subproblema por meio de um argumento delimitador.
3. Escolhemos um subproblema ativo e ramifique em uma variável fracionária. Repita até que não haja subproblemas ativos.

Para finalizar, podemos citar que como os cortes de custo estão relacionados a descoberta de um mapeamento válido para a instância executada, uma vez que realizam a comparação entre o custo mínimo e o custo da melhor solução já encontrada, temos que descartar aqueles mapeamentos cujo custo mínimo se igual ou exceda o custo da melhor solução.

2.2 Cutting Plane

No método Cutting Plane, o relaxamento do Problema de Otimização Inteira é resolvido e apenas as restrições que são violadas são inseridas. O modelo é reotimizado e a cada iteração uma formulação mais forte é obtida até que não sejam encontradas mais desigualdades violadas.

Por outro lado, quando nos referimos ao problema de descobrir quais são as restrições violadas ausentes, estamos falando também de um outro problema de otimização (encontrar a desigualdade mais violada), neste caso este é chamado de Problema de Separação. Os conhecidos planos de cortes foram propostos por Ralph Gomory na década de 1950 com o intuito de resolver problemas de programação inteira e programação inteira mista ou apenas de otimização inteira, nesse caso considerando como solução apenas valores inteiros. Por muito tempo este método foi considerado impraticável devido à instabilidade numérica, e como consequência ineficazes devido as varias rodadas de cortes que eram necessárias para avançar ou obter a solução ótima do problema. Porém na década de 1990, Gérard Cornuéjols e colegas de trabalho mostraram que eles eram muito eficazes em combinação com ramificação e limite maneiras de superar instabilidades numéricas.

Para ilustrar o que acontece neste método, suponhamos que temos um problema de otimização inteira dado por

$$\begin{aligned} &\max_{x \in \mathbb{Z}^n} c^T x \\ &s.a.\ \ Ax \leq b \\ &x \geq 0. \end{aligned}$$

Resolvemos a relaxação Problema de Otimização Inteira e obtemos uma solução fracionária x^*. Se queremos uma formulação melhorada deste programa inteiro, queremos gerar uma nova desigualdade $\alpha^T x \leq \beta$ tal que:

- É válido para o programa inteiro: todo ponto $x \in \mathbb{Z}^n$ que satisfaz $Ax \leq b$ e $x \geq 0$ também satisfaz a nova desigualdade. Afinal, não queremos mudar o problema.
- Consideremos a solução fracionária x^* obtivemos anteriormente: temos $\alpha^T x^* > \beta$. Se isso não acontecer, adicionar a nova desigualdade não ajudará; ainda teremos x^* como o solução ideal para o relaxamento PL.

Essa desigualdade é chamada de plano de corte para o programa de inteiros. Se pudermos chegar a um plano de corte, então podemos adicioná-lo como uma restrição adicional e resolver o relaxamento.

2.3 Comentários sobre Heurísticas

Os problemas de otimização inteira tem sido tratado na literatura por dois tipos de métodos: exatos e heurísticos. Os primeiros garantem que a solução ótima para o problema é encontrada, enquanto os métodos heurísticos alcançam soluções subótimas, porém possibilitam que instâncias maiores sejam tratadas em tempo menor.

Nos últimos anos, houve uma explosão de métodos heurísticos para problemas de otimização combinatória, com foco principalmente em aplicações específicas em áreas como programação, logística, roteamento, design e assim por diante. No entanto, talvez surpreendentemente, tem havido muito pouco feito para desenvolver métodos heurísticos para problemas gerais de programação inteira (PI), que incluem os domínios de programas de inteiros puros, onde todas as variáveis devem ser com valor inteiro e de programas inteiros mistos (MIPs).

3 Métodos

3.1 Algoritmos

Ambos os algoritmos foram desenvolvidos de maneira simples e baseados nas versões teóricas dos algoritmos apresentados. Foram desenvolvidos em python com auxílio do pacote MIP para a solução dos problemas relaxados. As lógicas utilizadas no desenvolvimento de cada algoritmo estão descritas a seguir:

3.2 Branch and Bound

Para resolver os problemas pelo método Branch and Bound, foi implementada uma função recursiva que retorna um conjunto de soluções factíveis e segue a lógica do pseudocódigo a seguir:

Algorithm 1: Branch and Bound

Entradas: Modelo(M), Conjunto(C)
(V, S) $\leftarrow$ *Solução*(*Relaxar*(M))
if *Norma(V)* $\leq$ *erro* **then**
  Retornar($C \cup \{S\}$)
else
  $r \leftarrow Escolher(V)$
  $N_e \leftarrow M$
  AdicionarRestrição($N_e, x_r <= Piso(V_r)$)
  $E \leftarrow BranchAndBound(N_e, C)$
  $N_d \leftarrow M$
  AdicionarRestrição($N_d, x_r >= Teto(V_r)$)
  $D \leftarrow BranchAndBound(N_d, C)$
  Retornar($C \cup \{D, E\}$)
end

Onde M é um modelo de otimização inteira, C é um conjunto que ao final da execução do programa contém as soluções encontradas pelo método, assim, a solução do modelo M é dada pelo menor ou maior elemento de C, dependendo do objetivo de M (minimização ou maximização, respectivamente). As funções que aparecem no pseudocódigo estão explicadas a seguir:

- $Relaxar(M)$: retorna a relaxação do modelo M, ou seja, retorna um modelo com mesma função objetivo e mesmas restrições do modelo M, mas com variáveis contínuas;
- $Solução(M)$: soluciona o modelo M e retorna uma dupla (V, S), onde V é o ponto que minimiza o modelo M e S é a solução do modelo obtida no ponto V;
- $Norma(V)$: calcula a norma euclideana do vetor V;
- $Retornar(X)$: retorna X e encerra a função atual;
- $Escolher(V)$: dado um vetor V, esta função retorna o índice da coordenada com maior parte fracionária no vetor;
- $AdicionarRestrição(M, L)$: dados um modelo com n-variáveis M e uma inequação linear L em n-variáveis, esta função adiciona a restrição L no modelo M;
- $Piso(V_r)$ e $Teto(V_r)$: retornam o piso e o teto do número V_r, respectivamente.

Temos que este algoritmo é iniciado com um modelo de otimização inteira M e um conjunto de soluções factíveis C inicialmente vazio e após calcular uma solução ótima para a relaxação de M, é verificado se esta solução tem todas coordenadas inteiras, em caso afirmativo, a solução obtida é colocada no conjunto C. Caso contrário, a coordenada com maior parte fracionária será excluída da região factível, para isso, cria-se dois novos modelos, um restringe a região factível a valores menores que o desta coordenada e o outro restringe a valores maiores. Assim, a função segue a busca até que seja encontrada uma solução para o problema com todas as coordenadas inteiras. Para garantir que a função retorne soluções aproximadamente inteiras, em cada novo ciclo calcula-se a norma do vetor que resolve o problema relaxado e a solução é aceita apenas caso esta norma seja menor que o erro admitido.

3.3 Cutting Plane

A função utilizada para resolver problemas pelo método Cutting Plane foi criada de acordo com a lógica do pseudocódigo a seguir:

Algorithm 2: Cutting Plane

```
Entradas: Modelo(M)
k ← 0
while k = 0 do
    (V, S) ← Solução(Relaxar(M))
    C ← S
    if Norma(V) ≤ erro then
        k ← 1
    else
        AdicionarRestrição(M, Gomory(M, V))
    end
end
```

Neste pseudocódigo, a solução do problema é dada pela variável C. Algumas das funções utilizadas neste pseudocódigo estão definidas em 3.2, a única função nova é a função $Gomory(M, V)$ que adiciona um corte de Gomory no modelo M para cortar o ponto V.

A função apresentada para este método é uma função iterativa que, dado um modelo M, a cada passo calcula uma solução para a relaxação de M. Caso esta solução seja inteira, a função é encerrada e a solução ótima foi encontrada, caso contrário, a função adiciona um corte de Gomory às restrições do modelo e inicia um novo passo.

3.4 Problemas

Os problemas de menor dimensão (até dez variáveis) utilizados para encontrar o algoritmo foram criados manualmente e suas soluções foram obtidas a partir do solver de problemas de otimização inteira do pacote MIP e PuLP. Já os problemas com maiores dimensões (a partir de vinte e oito variáveis) foram obtidos da biblioteca MIPLIB, uma biblioteca online de problemas de otimização inteira mista, mas os utilizados no presente trabalho apresentam apenas variáveis inteiras. No total, os algoritmos foram testados em vinte diferentes problemas, com 3 até 297 variáveis.

3.5 Testes

Todos os problemas considerados foram testados em ambos os algoritmos. Em todos os testes realizados, um ponto que fornecia uma solução ótima para um problema era considerado inteiro se ao arredondar cada coordenada deste ponto, então a diferença entre este novo ponto e o original tinha uma norma menor que o erro admitido. Em todos os testes este erro era de 0.01. Os testes foram realizados no ambiente do Google Colaboratory, que fornece um ambiente para desenvolvimentos de códigos em python, onde os códigos são desenvolvidos em python notebooks e podem ser rodados em uma máquina virtual com 12.72 GB de RAM e 107.77 GB em disco e processador com frequência base de 2.20 GHz.

Como alguns dos problemas era muito complexos e estes métodos são bastante custosos, então foi considerado um tempo computacional de 300 segundos. Pela seção 3.2, vemos que se o algoritmo Branch and Bound não encontrasse a solução ótima a tempo, então como este cria uma árvore com várias soluções inteiras para o problema, então ao alcanar o tempo limite, o algoritmo retornaria a melhor solução encontrada até o momento.

Pela seção 3.3, como sabemos que o algoritmo Cutting Plane não encontra várias soluções inteiras para o problema, mas apenas uma, então não foi possível seguir o mesmo raciocínio para a finalização do método Branch and Bound no caso de o algoritmo não ser finalizado antes do tempo limite. No caso do algoritmo Cutting Plane, caso o algoritmo não fosse finalizado antes do tempo limite, então a solução retornada pela função seria a solução atual, que não necessariamente é uma função inteira, mas que poderia estar próxima da solução ótima.

4 Resultados

Na tabela abaixo estão representados os testes realizados para esse trabalho, a quantidade de variáveis presente em cada um dos testes, a solução ótima e as soluções obtidas pelos métodos de Branch and Bound e Cutting Plane referentes aos algoritmos implementados na seção anterior.

Teste	Variáveis	Solução	Branch and Bound	Cutting Plane
M6	3	4,000	4,000	4,000
M7	4	100,000	100,000	100,000
M8	5	32,000	32,000	32,000
M4	8	38,000	38,000	38,000
M1	9	2.397,000	2.397,000	2.397,000
M3	9	541,000	541,000	541,000
M2	10	374,000	374,000	374,000
M5	10	292,000	292,000	292,000
gen-ip016	28	-9.476,155	-9.320,850	-9.505,352
gen-ip036	29	-4.606,680	-4.244,902	-4.631,156
gen-ip054	30	6.840,966	7.721,123	6.767,749
gen-ip021	35	2.361,454	2.726,650	2.328,674
gen-ip002	41	-4.783,733	-3.757,000	-4.837,688
v150d30-2hopcds	150	41,000	57,000	27,803
gt2	188	21.166,000	134.450,909	19.723,476
p0201	200	7.615,000	9.160,000	14,246
2club200v15p5scn	201	-70,000	-12,000	-124,027
glass-sc	214	23,000	40,000	14,246
iis-glass-cov	214	21,000	39,000	13,662
iis-hc-cov	297	17,000	46,000	9,709

Table 1: Resultados obtidos pelos métodos implementados.

Agora, na próxima tabela temos os erros obtidos através da diferença entre os valores ótimos dos problemas e os valores obtidos pelos métodos de Branch and Bound e Cutting Plane implementados.

Teste	Erro BnB	Erro CP
M6	0	0
M7	0	0
M8	0	0
M4	0	0
M1	0	0
M3	0	0
M2	0	0
M5	0	0
gen-ip016	0,016	0,003
gen-ip036	0,079	0,005
gen-ip054	0,129	0,011
gen-ip021	0,155	0,014
gen-ip002	0,215	0,011
v150d30-2hopcds	0,39	0,322
gt2	5,352	0,068
p0201	0,203	0,998
2club200v15p5scn	0,829	0,772
glass-sc	0,739	0,381
iis-glass-cov	0,857	0,349
iis-hc-cov	1,706	0,429
Total	10,669	3,363

Table 2: Erros obtidos pelos métodos implementados.

Ressaltamos aqui também que o tempo de execução de cada um dos algoritmos não foi calculado diretamente, uma vez que nos pequenos casos os dois métodos apresentados resolviam problemas em menos de um segundo, e nos casos com a maior quantidade de variáveis (28 ou mais) os dois métodos não convergiam antes do tempo limite proposto anteriormente.

5 Conclusão

De acordo com os resultados obtidos anteriormente podemos concluir que:

- Considerando o caso em que a dimensão do problema, seja a quantidade de variáveis ou a quantidade de restrições, temos que os dois métodos foram eficientes e encontraram as soluções ótimas em um tempo curto e utilizando poucos recursos da maquina virtual;
- A partir da Tabela de Erro apresentada na seção anterior é fácil ver que o erro entre a solução obtida pelos métodos e a solução ótima cresce de acordo com o número de variáveis do problema. Vale a pena destacar que o método de Branch and Bound sofre mais com o acréscimo da mesmas, isso tem relação direta com o tempo de execução do algoritmo e com a quantidade de passos que são executados no mesmo;
- Durante a execução dos testes foi possível notar que o Algoritmo de Branch and Bound apresentado na seção 3.2, além de ser mais lento, considerando o tempo para obter as soluções, utilizou uma quantidade maior de memória RAM da máquina virtual nos seus

testes. Por outro lado, como já citado anteriormente, as soluções obtidas por cada iteração do Algoritmo temos que as soluções obtidas são mais próximas de soluções inteiras.

6 Referências

- CAPRARA A. e FISCHETTI M. "0,1/2-Chvátal-Gomory cuts. Mathematical Programming", 74(3):221–235, 1996.

- CID. S. "Programação Linear Inteira". Instituto de Computação - UNICAMP, 2019. Material disponível em *https://www.ic.unicamp.br/ cid/cursos/MO420/201902/*

- CORNUÉJLS, GÉRARD (2007). "Renascimento dos cortes de Gomory na década de 1990". Annals of Operations Research , vol. 149 (2007), pp. 63–66. Método de plano de corte - *pt.qaz.wiki/wiki/Cutting-plane-method*

- CHERRI A. C. e VIANA A. C., "Introdução a Problemas de Otimização". Bauru-SP, 2012. Disponível em: *http://wwwp.fc.unesp.br/ adriana/curiosidades/Introducao.pdf.*

- GALLI, L. Algorithms for Integer Programming. Pisa PI, Itália: UNIPI, 2014.

- GOLDBARG, M. C. e LUNA, H. P. L. "Otimização Combinatória e Programação Linear: Modelos e Algoritmos", Editora Campus, Rio de Janeiro, 2000.

- NCSS Statistical Software, "Mixed Integer Programming, Chapter 482". Disponível em: *https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedure/NCSS/Mixed-Integer-Programming.pdf*

- NEMHAUSER, G. L.; WOLSEY, L. A. "Integer and combinatorial optimization". New York: John Wiley e Sons, 1998.

- LAND A. H.e DOIG A. G., "An Automatic Method of Solving Discrete Programming Problems". Vol. 28, No. 3 (Jul., 1960), pp. 497-520 (24 pages). Published By: The Econometric Society. Disponível em: *https://www.jstor.org/stable/1910129?seq=1.*

- RUST R., "Um algoritmo Branch and Bound para resolução de problemas em localização capacitados". Dissertação de Mestrado, UFRJ. Rio de Janeiro, 1983. Disponível em: *https://www.cos.ufrj.br/uploadfile/1368209564.pdf.*

- WOLSEY, L. A. "Integer programming". New York: John Wiley Sons, 1998.

Apêndice

No que segue abaixo temos os testes utilizados neste trabalho:

- Os testes M1, M2, M3, M4, M5, M6, M7 e M8 podem ser encontrados em: Testes;
- Enquantos os demais foram retirados do *The Mixed Integer Programming Library*, e podem ser encontrados em: MipLib.