Universidade Federal do Rio de Janeiro
Otimização Combinatória - COS890
Professor: Abilio Lucena

Leilões Combinatórios - uma aplicação do problema de empacotamento de conjuntos através de métodos exatos

Amanda Camacho Novaes de Oliveira

Diego Amaro Ferraz da Costa

Diego Athayde Monteiro

Resumo

Neste trabalho, descrevemos o problema dos leilões combinatórios como uma aplicação do Problema de Empacotamento de Conjuntos, primeiramente proposto por Karp [1]. Aplicamos três algoritmos exatos para a resolução do mesmo: **Relaxação Lagrangeana**, **Branch-and-Bound** e **Branch-and-Cut**. Os mesmos foram implementados através do solver *Gurobi*, utilizando a linguagem de programação *Julia* e biblioteca *JuMP*. Utilizamos diferentes instâncias para comparar o desempenho dos algoritmos, chegando à conclusão de que na maioria dos casos o *Branch-and-Cut* é o método mais eficiente, para as implementações realizadas.

1 Introdução

A otimização combinatória é a área que estuda problemas de otimização aplicados a conjuntos discretos. Os problemas de otimização são aqueles onde se procura determinar valores extremos de um conjunto de restrições, com respeito a uma função, esta chamada de função objetivo. As variáveis envolvidas nos problemas de otimização são denominadas variáveis de decisão. Pode-se classificar os problemas de otimização combinatória de acordo com os tipos de variáveis que os compõem: variáveis lineares ou não lineares; variáveis binárias, inteiras ou reais.

Neste trabalho, definimos e aplicamos técnicas de soluções exatas para um conhecido problema, o Problema de Empacotamento de Conjuntos (PEC). Ele consiste em encontrar uma combinação de subconjuntos de elementos de forma que se obtenha peso máximo e que nenhum elemento seja selecionado por mais de um conjunto. Pode ser descrito como um problema de programação inteira, onde as variáveis são lineares e podem assumir valores binários 0-1.

Este problema possui muita aplicação prática e uma das mais conhecidas é a de Leilões Combinatórios, que é o tema abordado nesse trabalho. Um leilão combinatório consiste em distribuir um número m de produtos através de n lances. Se um lance é escolhido, um determinado lucro é obtido, tendo como objetivo maximizar o lucro total, respeitando a condição de que cada produto só pode estar contido em no máximo um lance, considerando os lances escolhidos. Considerando esta aplicação do PEC, aplicamos três métodos exatos para resolver o problema: um algoritmo de **Relaxação Lagrangeana**, um algoritmo ***Branch-and-Bound*** e um segundo algoritmo *Branch-and-Bound* com algoritmos de planos de corte associados ao mesmo, denominado ***Branch-and-Cut***.

Esse texto está organizado da seguinte forma: na seção 2, detalhamos o Problema de Empacotamento de Conjuntos, e nas seções 3, 4 e 5 descrevemos um pouco cada método aplicado. Na seção 6 exibimos os resultados obtidos para cada umas das instâncias escolhidas, realizando uma comparação entre os métodos e, por fim, na seção 7 apresentamos nossas conclusões, contribuições de cada integrante do grupo e ideias para trabalhos futuros.

2 O Problema de Empacotamento de Conjuntos

O problema de empacotamento de conjuntos (PEC) (do inglês, *Set-Packing Problem*) foi um dos 21 problemas enunciados por Karp em [1]. Neste mesmo trabalho, Karp mostrou que se trata de um problema NP-completo através de uma redução feita do problema da clique máxima de um grafo. O PEC

também possui forte relação com o problema da cobertura de conjuntos, estes formando um par dual-forte.

A descrição formal do PEC, como um problema de otimização, é dada da seguinte forma:

- $I = \{1, ..., m\}$: conjunto finito de elementos
- $J = \{1, ..., n\}$: conjunto finito de índices
- $\{I_j \subseteq : j \in J\}$: conjuntos dos subconjuntos viáveis de I
- Existe um peso c_j à cada I_j, $j \in J$ $\{I_j \subseteq : j \in J\}$: conjuntos dos subconjuntos viáveis de I
- $\{I_j : j \in J_i\}$: subconjuntos dos subconjuntos viáveis de I que contém $i \in I$
- Variáveis binárias $x_j \in \{0,1\}$ que expressam a decisão de escolher ou não I_j, $j \in J$

Dadas estas definições, sua formulação como um problema de programação inteira é dada pela equação 1.

$$\begin{aligned} \max \quad & \sum_{j \in J} c_j x_j \\ \text{sujeito a} \quad & \sum_{j \in J_i} x_j \leq 1, \qquad i \in I \\ & x_j \in \{0,1\}, \quad j \in J \end{aligned} \tag{1}$$

Este problema possui uma ampla aplicação prática. Como um exemplo destas, temos o escalonamento de tripulações de companhias aéreas para aviões. Cada avião da frota precisa ter uma tripulação designada a ele, composta por um piloto, copiloto e navegador [2]. Como mencionado anteriormente, a abordagem que fazemos aqui ao PROBLEMA DE EMPACOTAMENTO DE CONJUNTOS é feita no contexto dos leilões combinatórios, que é objeto de estudo de muitos pesquisadores em diversos trabalhos ([3], [4], [5]). Nesta aplicação os conjuntos I e J podem ser enxergados como um conjunto de m produtos e um conjunto de n lances, respectivamente. Nosso objetivo então é maximizar o lucro total obtido sem violar a restrição de que um produto só pode ser escolhido por, no máximo, um lance, entre os lances selecionados. E este lucro é um valor associado a seleção de cada lance. Para a solução desse problema implementamos uma formulação alternativa, dada pela equação 2, onde a_{ij} vale 1 se o lance j seleciona o produto i, ou 0 caso contrário.

$$\begin{array}{lll} \max & \sum_{j=1}^{n} c_j x_j & \\ \text{sujeito a} & \sum_{j=1}^{n} a_{ij} x_j \leq 1, & i = 1, \cdots, m \\ & x_j \in \{0,1\}, & j = 1, \cdots, n \end{array} \tag{2}$$

Esta formulação, também observada em outros trabalhos como [6], por exemplo, foi utilizada por entendermos que ela é mais intuitiva, sendo mais fácil de visualizar e entender o problema dessa forma. Entretanto, foi observado ao longo da implementação que a formulação dada pela equação 1 é mais eficiente para instâncias muito grandes, uma vez que na maioria dos casos as matrizes A induzidas pela seleção de produtos por lances são esparsas.

3 Relaxação Lagrangeana

A relaxação lagrangeana consiste em um método de decomposição das restrições de um determinado problema em dois grupos: as restrições "convenientes" e as restrições "inconvenientes". Chamamos de restrições "inconvenientes" aquelas que fazem com que o problema se torne difícil de resolver. Dessa forma tais restrições são retiradas do problema de programação inteira e adicionadas à função objetivo em um termo $u(d - Dx)$. Estas variáveis u_i são denominadas variáveis duais ou multiplicadores de Lagrange.

Nem sempre a relaxação lagrangeana nos oferece uma solução ótima para o problema. Ela é limitada pelos resultados da relaxação linear. Entretanto mesmo quando não chega ao valor ótimo a relaxação traz informações úteis para a resolução do problema, e implementações utilizando heurísticas lagrangeanas, muitas vezes levam a algoritmos mais eficientes de otimização de um problema (como por exemplo é apresentado em [6]). O segundo grupo de restrições, as "convenientes" tornam o problema mais fácil de ser resolvido e eventualmente podemos obter a solução ótima para o problema original através da relaxação ou, se não for o caso, limitantes superiores e inferiores para fortalecer o problema.

O ponto de partida para a aplicação deste método é a elaboração do subproblema lagrangeano. Baseado no modelo de programação inteira que apresentamos na seção 2, o subproblema lagrangeano para o PEC é:

$$\begin{array}{ll} \max & \sum_{j=1}^{n} (c_j - \sum_{i=1}^{m} u_j a_{ij}) x_j + \sum_{i=1}^{m} u_i \\ \text{sujeito a} & x_j \in \{0,1\}, \quad \forall j = 1, \cdots, n \end{array}$$

Primeiramente, para obtenção de limitantes superiores, utilizamos a seguinte convenção: uma variável x_j é igual a 1, quando $(c_j - \sum_{i \in I_j} u_j) > 0$ e x_j é igual a 0, caso contrário. Além disso, caso todos os custos $(c_j - \sum_{i \in I_j} u_j)$ sejam negativos, forçamos o maior dentre esses custos negativos para receber valor 1.

O algoritmo 1 em conjunto com uma heurística gulosa para obtenção de limites inferiores resumem a abordagem adotada para o PEC, utilizando o método da relaxação Lagrangeana. A obtenção de limites inferiores consiste em ordenar o conjunto de lances por ordem decrescente de preços por número de produtos cobertos. Os resultados obtidos serão exibidos na seção 6.

Algoritmo 1 Método do Subgradiente para o PEC

Variáveis:
MaxIter = Número máximo de iterações definido
s = zeros(m), sqrSum = 0, $\pi_{min} = 0.0001$
$k = 0$ ▷ *Índice da iteração*

Começar loop
Definir $\pi = 2$ e multiplicadores de Lagrange $u = 0$
while $k <$ MaxIter **then**
    Resolver o subproblema Lagrangeano, obtendo: $\overline{x}$, $\overline{z}$
    Encontrar soluções viáveis a partir da solução relaxada: $\underline{x}$, $\underline{z}$
    $k = k + 1$

    Critérios de Parada:
    if $k \equiv$ MaxIter **or** $\pi < \pi_{min}$ **then**
        return $\underline{x}$

    if $(\overline{z} - \underline{z} < 1)$ **or** ($\underline{x}$ é viável e a função de otimização tem mesmo valor tanto na versão primal quanto na dual para os multiplicadores u e a solução $\underline{x}$) **then**
        return $\underline{x}$ ▷ *Encontrado o valor ótimo*

    Atualização de π
    if $\frac{\lfloor MaxIter \rfloor}{20}$ iterações consecutivas sem melhora **then**
        $\pi := \pi/2$

Atualização dos multiplicadores de Lagrange:

15: **for** cada produto i **do**

16: $s[i] = 1 - \sum_{j \in L} a[i,j] * \overline{x}[j]$

17: $u_i = \max\{0, (1+\epsilon)T \cdot s_i\}, \quad \forall i = 1, \ldots, m$

18: sqrSum = sqrSum + $s[i]^2$

19: $T = \dfrac{\pi * (\overline{z} - \underline{z})}{\text{sqrSum}}$

20: **if** $T < \pi_{min}$ **then**

21: **return** $\underline{x}$ ▷ *Parada por valor muito pequeno do passo*

22: **for** cada produto i **do**

23: $u[i] = max(0, u[i] - T * s[i])$

4 Branch and Bound

Uma técnica muito aplicada para resolução de problemas combinatórios é a técnica de dividir e conquistar que consiste em resolver subproblemas mais fáceis que o original e após este processo combinar as soluções obtidas até obtermos a solução do problema original. Porém, esta enumeração de subproblemas de forma explícita pode ser muito custosa, e justamente neste contexto, é que algoritmo *Branch-and-Bound* entra em ação.

O algoritmo *Branch-and-Bound* consiste em uma técnica de enumeração implícita de soluções, onde através de limites locais, obtemos limites globais. Toda a eficiência e ganho que o algoritmo *Branch-and-Bound* nos oferece é por conta das podas que acontecem na árvore de enumeração implícita, que podem ocorrer por: otimalidade, inviabilidade e limites duais. A técnica foi definida primeiramente em [7] por Ailsa Land e Alison Doig em 1960, e figura como um dos mais importantes métodos exatos no contexto de resolução de problemas de otimização até os dias atuais.

Para a implementação do método *Branch-and-Bound*, fizemos uso de uma abordagem mais direta, onde o próprio solver do *Gurobi* [8] já executa o método por padrão. Foram desabilitadas funções como cortes, múltiplas *threads*, métodos de pré-processamento e afins, de forma que apenas o método de *Branch-and-Bound* original fosse aplicado para a solução do problema. Resolvendo dessa forma, extraímos algumas informações importantes como o número de nós explorados pela árvore de enumeração e o tempo de execução.

5 Branch and Cut

Para fortalecermos o algoritmo *Branch-and-Bound*, é possível fazermos uso de algoritmos de planos de corte em cada nó de sua árvore de enumeração. Esses

algoritmos trazem informações adicionais ao processo de *Branching*, fazendo com que se gaste mais tempo em cada nó, na esperança de que isso diminua o número de nós explorados e o tempo total de utilização da CPU.

Como desigualdades válidas para o PEC, utilizamos desigualdades de cliques geradas a partir de um grafo de conflito. Este grafo é definido da seguinte forma: cada lance no problema de leilões combinatórios define um vértice do grafo, e dois vértices possuem aresta entre si se ambos os lances selecionam um mesmo produto. Como quaisquer lances conectados selecionam pelo menos um produto em comum, no máximo um lance de cada clique pode ser selecionado para uma solução viável.

O algoritmo 2 é resultante da estratégia descrita no parágrafo anterior. Utilizando o algoritmo de Bron-Kerbosch [9], enumeramos todas as cliques maximais do grafo de conflito e as adicionamos como desigualdades válidas fortes, quando essas são violadas pelas soluções parciais do *Branch-and-Bound*. Utilizando desse método, conseguimos diminuir bruscamente o número de nós explorados na árvore de enumeração implícita em diversos casos.

Algoritmo 2 Branch and Cut

```
    Variáveis:
 1: vizinhos = Array[]
    Definindo vizinhanças:
 2: for j em Lances do
 3:     viz = Array[]
 4:     for i em Produtos do
 5:         if a[i, j] ≡ 1 then
 6:             if sum(a[i, :]) > 1 then
 7:                 for j_2 em Lances do
 8:                     ifa[i, j_2] ≡ 1 and j_2 ≠ j and j_2 ∉ viz then
 9:                         Adiciona j_2 no Array viz
    Algoritmo de Bron-Kerbosch
10: function ACHARCLIQUE(clique, candidatos, excluidos, output)
11:     if candidatos and excluidos sejam vazios then
12:         Adiciona clique em output
13:     for v ∈ candidatos do
14:         R_2 = clique ∪ {v}
15:         P_2 = candidatos ∩ vizinhos[v]
16:         X_2 = excluidos ∩ vizinhos[v]
17:         ACHARCLIQUE(R_2, P_2, X_2, output)
18:         candidatos = candidatos \ {v}
19:         excluidos = excluidos ∪ {v}
```

```
    Algoritmo de Corte
20: function CORTE()
21:     ϵ = 0.0001
22:     x_val = Valores encontradas para as variáveis de decisão x no nó atual
23:     for clique maximal do grafo de conflito
24:         soma = 0
25:         for nó j ∈ clique
26:             soma = soma + x_val[j]
27:         if soma > 1 + eps then
28:             Restrição violada pela clique. Adicionar corte.
29:             Adicionar restrição correspondente à clique atual
```

6 Resultados

Ao longo desta seção são comparados os resultados obtidos pelos três métodos implementados – Relaxação Lagrangeana, *Branch-and-Bound* e *Branch-and-Cut* – aplicados a diferentes instâncias do problema de leilões combinatórios.

Inicialmente os três foram aplicados a um caso base simples, de 3 produtos e 4 lances, para verificar o funcionamento correto das implementações. Em seguida aplicamos os mesmos algoritmos em casos maiores, com o objetivo de avaliar o melhor valor possível encontrado para essas instâncias.

Abaixo detalharemos as informações relevantes de cada instância utilizada, considerando os resultados obtidos para cada um dos algoritmos executados.

É feita a ressalva de que os tempos de processamento apresentados foram obtidos considerando apenas uma execução de cada algoritmo, em máquinas próprias. Portanto, os mesmos não incluem incertezas e estão explicitados apenas para fins de comparação entre diferentes métodos para uma mesma instância.

6.1 Caso Base

Para fins comparativos com os métodos que foram aplicados para a resolução do PROBLEMA DE EMPACOTAMENTO DE CONJUNTOS, utilizamos como instância de teste a chamada *toy2*, com 3 produtos e 4 lances, que pode ser observada na tabela 1.

Esta é uma instância muito simples, criada especificamente para testar o funcionamento adequado dos métodos. É esperado que, para este caso, todos os métodos sempre encontrem o valor ótimo e que nenhum apresente problemas para chegar a essa solução. Neste exemplo, o valor ótimo é 32, obtido quando se seleciona os lances 1 e 4.

Produto \ Lance	l_1	l_2	l_3	l_4
p_1	1		1	
p_2		1		1
p_3			1	1
Valor do lance	10	12	18	22

Tabela 1: Instância de teste *toy2*. Na tabela principal, a ocorrência de um valor 1 entre o lance l_j e o produto p_i significa que p_i é selecionado por l_j. A linha de valores especifica o ganho que a seleção de cada lance l_j traz.

Na prática, todos os três métodos foram capazes de encontrar o valor ótimo. A relaxação lagrangeana o fez em 63 iterações, enquanto o *Branch-and-Bound* e o *Branch-and-Cut* encontraram a solução em apenas uma iteração do algoritmo simplex. Todos resolvem a instância em menos de 10 ms.

Como o problema é muito simples de resolver, ele não foi o suficiente para averiguar as vantagens do algoritmo *Branch-and-Cut*. Isso foi feito com instâncias mais complexas, apresentadas ao longo desta seção.

6.2 Leilão de energia elétrica (10 x 10)

Este é um exemplo baseado no cenário de leilão para a venda de geração de energia elétrica, adaptado de [10]. Foi extraída uma relação de 10 produtos e 10 lances, a qual pode ser observada na tabela 2.

Produto \ Lance	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
p_1						1	1			
p_2								1		
p_3		1				1				
p_4				1	1					
p_5										1
p_6			1				1	1	1	
p_7				1	1					1
p_8	1		1							
p_9						1				
p_{10}	1	1						1	1	
Valor do lance	97.00	95.00	94.30	93.00	98.90	98.75	90.00	99.00	95.00	98.90

Tabela 2: Leilão de energia elétrica. Na tabela principal, a ocorrência de um valor 1 entre o lance l_j e o produto p_i significa que p_i é selecionado por l_j. A linha de valores especifica o ganho que a seleção de cada lance l_j traz.

Este exemplo tem solução ótima com lucro de 296,65, obtido através da

seleção dos lances l_6, l_8 e l_{10}. Tal resultado foi alcançado pelas técnicas *Branch-and-Bound* e *Branch-and-Cut*. A relaxação lagrangeana, no melhor caso, chegou no valor 290,75 ao rodar o algoritmo por 1.000 iterações, com tempo de execução de 0,20 s.

Uma análise mais profunda mostrou que neste caso o algoritmo da relaxação lagrangeana não consegue solução melhor porque o limitante superior, por ela obtido, está limitado pelo valor da relaxação linear, independente da quantidade de iterações executadas no algoritmo, que neste caso é 342,8 (obteve-se 342,83 para o melhor limitante superior por relaxação lagrangeana). É interessante observar que existe uma grande diferença entre o resultado ótimo do problema com variáveis binárias e de sua relaxação linear, mesmo sendo essa uma instância relativamente pequena.

A resolução pelo método *Branch-and-Bound* clássico resultou na análise de 7 nós da árvore de enumeração, executando em 0,02 s, enquanto o *Branch-and-Cut*, com a adição dos planos de corte, só precisou analisar 3 nós para encontrar a solução ótima, durando 0,01 s.

O grafo de conflito desta instância, apresentado na figura 1, apresenta 6 cliques maximais: $\{1, 2, 8, 9\}$, $\{1, 3, 8, 9\}$, $\{2, 6\}$, $\{3, 7, 8, 9\}$, $\{4, 5, 10\}$ e $\{6, 7\}$. Durante a execução do algoritmo *Branch-and-Cut* foi aplicado um corte, dado pela clique $\{1, 3, 8, 9\}$, que foi violada na análise de um dos nós do *Branch-and-Bound*.

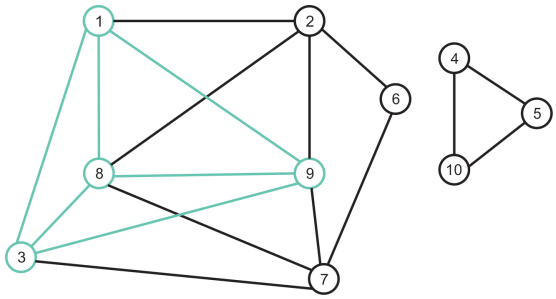

Figura 1: Grafo de conflito do leilão de energia elétrica. Em verde está assinalada a clique aplicada como plano de corte na execução do *Branch-and-Cut*.

6.3 Instâncias de referência

Para analisar melhor as propriedades dos métodos implementados, foram selecionadas instâncias do PEC com grande número de produtos e lances. Com muitas variáveis e restrições o problema se torna mais difícil e é necessário um maior esforço por parte dos métodos.

As instâncias aqui utilizadas foram extraídas da fonte [11], que contém um conjunto de *Benchmarks* para o PEC. Foram escolhidos três exemplos:

- "pb_100rnd0100.dat": 500 produtos, 100 lances e cada produto é coberto por dois lances. Os preços de cada lance variam entre 1 e 20.
- "pb_100rnd0200.dat": 500 produtos, 100 lances e cada produto é coberto por dois lances. Todo lance tem o valor unitário.
- "pb_100rnd0500.dat": 100 produtos, 100 lances e cada produto é coberto por dois lances. Os preços de cada lance variam entre 1 e 20.

Os resultados para cada uma das instâncias escolhidas podem ser observados a seguir.

6.3.1 Benchmark 500 x 100 - Pesos variáveis

Este primeiro *benchmark* corresponde à instância "pb_100rnd0100.dat", na qual tem-se 500 produtos e 100 lances. Os preços de cada lance são variados, podendo valer de 1 a 20, e cada produto é coberto por dois lances. O valor ótimo dessa instância é 372.

O *Branch-and-Bound* obteve valor máximo explorando 1036 nós (em 0,51 s), enquanto o *Branch-and-Cut* a obteve explorando apenas 95 nós (0,31 s). Foram adicionados 99 planos de corte no *Branch-and-Cut*, o que demonstrou uma clara melhora no desempenho do algoritmo.

A relaxação lagrangeana não converge para o ótimo, o limitante superior convergindo para 514,5. O algoritmo para com aproximadamente 40 s por estourar o limite de iterações. Esse resultado, embora não seja o ótimo, fornece o limitante oriundo da relaxação linear, o que é uma informação relevante, e a melhor solução viável encontrada é 347, o que está relativamente próximo do ótimo de 372, especialmente quando se considera a grande diferença com relação ao melhor limitante superior.

6.3.2 Benchmark 500 x 100 - Pesos unitários

Este segundo *benchmark*, correspondente à instância "pb_100rnd0200.dat", possui as mesmas características que a anterior (500 produtos por 100 lances, 2 lances por produto), com a diferença que os lances têm todos o mesmo peso, no caso um peso unitário. Isso significa que a escolha de um lance traz o mesmo benefício que a escolha de qualquer outro e a seleção de lances ótima vai ser aquela que permitir a escolha do número máximo de lances. O valor ótimo para esta instância é de 34.

O *Branch-and-Bound* obteve valor máximo explorando 1148 nós, realizando isso em 0,31 s, enquanto o *Branch-and-Cut* explorou apenas 35 nós, tendo aplicado 98 planos de corte e levando um total de 0,14 s. Ambos os métodos encontraram o valor ótimo, mas pode-se observar um ganho de desempenho quando se aplica o algoritmo de planos de corte.

Novamente, a relaxação lagrangeana não converge para o ótimo, encontrando o valor máximo de 32 (ao invés de 34), com melhor limitante superior equivalente a 50. O algoritmo rodou por 26 s e 684 iterações, quando parou, ao obter o valor do multiplicadores de lagrange muito pequeno, que é um dos critérios de parada do nosso algoritmo.

6.3.3 Benchmark 100 x 100 - Pesos Variáveis

O terceiro *benchmark* corresponde à instância "pb_100rnd0500.dat", na qual o número de produtos e lances é igual, ambos com cardinalidade 100. A variação dos preços de cada lance está contida na mesma faixa da instância anterior, podendo valer de 1 a 20, e cada produto também é coberto por dois lances. O valor ótimo dessa instância é 639.

Entretando, a resolução da instância pelos métodos *Branch-and-Bound* e *Branch-and Cut* se mostrou bem imediata. Ambos realizaram o procedimento em tempo muito pequeno e sem explorar nenhum nó da árvore de enumeração, levando em torno de 0,01 s para encontrar o ótimo. Porém, houve um tempo de pré-processamento utilizado para gerar o grafo de conflito e extrair as cliques maximais, que nesse caso não tiveram qualquer utilidade.

Nesta instância, a relaxação lagrangeana consegue atingir o ótimo, o valor 639. E encontra este valor em 412 iterações e leva 5,073 s para fazê-lo.

6.4 Instâncias PBP

As instâncias avaliadas a seguir foram criadas de acordo com o gerador de conjuntos PBP (*Price Proportional Bid*), descrito em [6][1]. Os conjuntos de teste PBP geram amostras nas quais o preço de um lance é relacionado à quantidade e qualidade dos produtos que ele cobre, e ao mesmo tempo o preço de um produto é aproximadamente proporcional ao número de lances pelos quais ele é coberto. Isso é um padrão observado em problemas reais de leilões combinatórios que o gerador de conjunto de testes tenta reproduzir.

São apresentadas a seguir duas instâncias:

[1] Código descrito foi reproduzido no arquivo "gerador_pbp.jl"

- "pbp_100-50_dens0.100000.txt": 100 produtos, 50 lances. Cada lance tem 10% de chance de selecionar um dado produto.
- "pbp_50-50_dens0.200000.txt": 50 produtos, 50 lances. Cada lance tem 20% de chance de selecionar um dado produto.

6.4.1 PBP 100 x 50 - Densidade 0,1

Nesta instância é apresentado um caso com 100 produtos e 50 lances, com densidade de 0,1. O parâmetro de densidade diz respeito à probabilidade de um produto ser coberto por um determinado lance (ou, similarmente, de um lance cobrir um produto). A instância gerada foi chamada "pbp_100-50_dens0.100000.txt".

Ambos *Branch-and-Bound* e *Branch-and-Cut* foram capazes de encontrar o valor ótimo de 285,811. O *Branch-and-Bound* encontrou a solução explorando 79 nós, e durou em torno de 0,06 s.

Nesta instância observou-se que a adição do algoritmo de planos de corte não gerou ganhos de tempo. Pelo contrário, embora acabe explorando apenas 18 nós, ele levou 2,41 s para rodar, sendo aproximadamente 40 vezes mais lento. Isso ocorre porque há a adição de planos de corte demais para a magnitude do problema, um total de 382 planos. Esta instância demonstra que possivelmente seria vantajoso para este problema alterar o algoritmo de planos de corte de forma que o mesmo adicione apenas a restrição mais violada ao invés de todas as restrições violadas toda vez que o mesmo é chamado.

Por fim, o algoritmo de relaxação lagrangeana ainda não é capaz, por si só, de encontrar o valor ótimo, obtendo melhor solução 256,996. O mesmo rodou por 1.000 iterações, obtendo melhor limite superior de 421,576.

6.4.2 PBP 50 x 50 - Densidade 0,2

A segunda instância apresenta um caso com 50 produtos e 50 lances, com densidade 0,2. O parâmetro de densidade indica que um determinado produto qualquer tem 20% de chance de ser coberto por um dado lance (e ao mesmo tempo um lance tem 20% de chance de cobrir um dado produto). A instância em questão foi chamada "pbp_50-50_dens0.200000.txt".

Nesta instância o *Branch-and-Bound* foi capaz de encontrar o valor ótimo de 313,575. Ele explorou 81 nós e levou 0,06 s.

A relaxação lagrangeana levou 5,15 s para terminar por ter atingido o número máximo de iterações. É interessante observar que este método foi capaz de obter o valor ótimo para esta instância, encontrando 313,575 como melhor limitante

inferior. No entanto ele não obtém garantia de otimalidade, e o limite dual lagrangeano mínimo obtido foi de 482,159.

Não conseguimos obter um resultado pelo método *Branch-and-Cut* para esta instância. O algoritmo de Bron-Kerbosch, para encontrar as cliques maximais no grafo de conflito, não finalizou a rotina após 3600 s de execução, desse modo, concluímos que o algoritmo desenvolvido não foi eficiente para resolver esta instância. Quando comparado com a ordem de grandeza que os outros algoritmos levam para resolver, esse tempo pode ser considerado infinito e, portanto, inviável.

Essa observação é um caso extremo, feita durante os testes de que, embora geralmente a adição dos planos de corte pelo *Branch-and-Cut* traga um aumento de eficiência para o *Branch-and-Bound*, esse aumento pode não compensar, uma vez que antes de rodar esse algoritmo mais eficiente, existe a necessidade de executar o algoritmo para encontrar os cliques maximais, o qual corresponde a uma perda de eficiência.

7 Conclusão

Neste trabalho, abordamos o Problema de Empacotamento de Conjuntos (PEC) e uma aplicação bem conhecida que são os leilões combinatórios. Resolvemos o problema pelo solver *Gurobi*, com a linguagem *Julia*, através da biblioteca *JuMP*. Utilizamos três métodos exatos e comparamos os resultados obtidos entre eles. Foram usadas diversos tipos de instâncias: algumas de elaboração própria, uma instância extraída de [10], outras retiradas de [11] e instâncias geradas a partir do algoritmo descrito em [6].

Observamos que a relaxação lagrangeana geralmente não obteve o valor ótimo – afinal, da maneira implementada, a mesma é limitada pela relaxação linear –, porém em diversos casos alcançamos soluções bem próximas do ótimo. Nesses casos, mesmo quando o ótimo não é obtido, as informações dos limites primal e dual são relevantes, e podem ser utilizados para auxiliar o processo de otimização como um todo. Durante os testes, verificamos que a relaxação não é o algoritmo mais rápido entre os três, mesmo quando alcança o valor ótimo, entretanto é importante ter em mente que o código desenvolvido para a relaxação lagrangeana não foi totalmente otimizado, contendo vários aspectos que diminuem seu desempenho, enquanto o *Branch-and-Bound* e o *Branch-and-Cut*, por usarem o solver *Gurobi*, provavelmente já estão implementados de maneira altamente eficiente.

Os métodos *Branch-and-Bound* e *Branch-and-Cut* obtiveram as soluções ótimas na maioria das instâncias testadas, o segundo se mostrando bem mais eficiente em termos de tempo computacional que o primeiro, por conta das

desigualdades válidas fortes que extraímos a partir de um grafo de conflito elaborado com base na instância em questão. Entretanto esse tempo não leva em conta o pré-processamento necessário para extrair o grafo de conflito e as cliques maximais, o que pode vir a ser altamente custoso.

No contexto das contribuições de cada integrante do grupo, a divisão foi realizada da seguinte forma: todo o trabalho de implementação dos algoritmos através da linguagem *Julia* e do solver *Gurobi* foi feito em total colaboração de todos os membros por meio de chamadas de vídeo onde uma pessoa compartilhava a tela enquanto todas davam palpites, ideias e faziam discussões a respeito da implementação. O elaboração do gerador de instâncias citado em [6] foi por conta da discente Amanda Camacho, o trabalho de pesquisa de fontes e esboço geral inicial do relatório foi feito pelo discente Diego Amaro, e a formalização dos algoritmos na forma teórica descritos neste documento foi feita pelo discente Diego Athayde. Por fim, toda a apresentação de slides e complementação do relatório também foi feita em conjunto através de ferramentas online onde todos os membros contribuíram de igual forma.

Como trabalhos futuros, pretendemos refinar os métodos implementados para obter soluções mais próximas da solução ótima nos casos em que não foram alcançadas. Um outro possível refinamento pode ser a combinação dos métodos aqui apresentados, a fim de englobarmos os pontos positivos de todos e eliminar – ou ao menos minimizar – os pontos negativos que cada um possui individualmente e através deste refinamento, obter as soluções ótimas em um tempo menor. E ainda, a aplicação de outros métodos para a resolução do problema aqui tratado, incluindo métodos não exatos, para tratar instâncias muito maiores.

Referências

[1] Richard Karp. Reducibility among combinatorial problems. volume 40, pages 85–103, 01 1972.

[2] A. Tajima and S. Misono. Using a set packing formulation to solve airline seat allocation/reallocation problems. *Journal of the Operations Research Society of Japan*, 42, 03 1999.

[3] Arne Andersson, Mattias Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. *ICMAS*, 05 2000.

[4] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. volume 1, pages 548–553, 01 1999.

[5] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Cabob: A fast optimal algorithm for combinatorial auctions. pages 1102–1108, 01 2001.

[6] Yunsong Guo, Andrew Lim, Brian Rodrigues, and Jiqing Tang. Using a lagrangian heuristic for a combinatorial auction problem. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 5–pp. IEEE, 2005.

[7] A. Land and A. Doig. An automatic method of solving discrete linear programming problems. *Econometrica*, 28:497–520, 07 1960.

[8] LLC Gurobi Optimization. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2020. Acessado em: 24/09/2020.

[9] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun ACM*, 16:575–577, 01 1973.

[10] E. Bastos Silva. Metodologia e simulação de leilão simultâneo-combinatório para novos empreendimentos de geração de energia elétrica, 02 2011.

[11] Benchmarks for the Set Packing Problem. `https://www.emse.fr/~delorme/SetPacking.html`. Acessado em: 24/09/2020.