Algoritmos Branch-and-Price para o Problema de Empacotamento em Recipientes com Restrições de Classe*

Yulle Glebbyo[1], Flávio K. Miyazawa[1], Rafael C. S. Schouery[1], Eduardo C. Xavier[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1251 – 13083-852 – Campinas – SP – Brazil

glebbyo@gmail.com, {fkm,schouery,eduardo}@ic.unicamp.br

Abstract. *In the Class Constrained Binpacking Problem, a set of items of various sizes and classes must be packed in bins, each with a capacity B and a limit C on the amount of items from different classes used, in a way that minimizes the number of bins used. We present Branch-and-Price algorithms for this problem, as well as algorithms for the Class Constrained Knapsack Problem, which appears as a pricing subproblem. In the experiments, the best algorithm proposed solved 93% of the instances in less than 30 seconds.*

Resumo. *No problema de Empacotamento em Recipientes Com Restrições de Classe, um conjunto de itens de variados tamanhos e classes deve ser empacotado em recipientes, cada um com capacidade B e limite C na quantidade de classes diferentes acomodadas, minimizando o número de recipientes usados. Apresentamos algoritmos de Branch-and-Price para este problema, assim como algoritmos para o Problema da Mochila Com Restrições de Classe, que aparece como subproblema de pricing. Nos experimentos executados, o melhor algoritmo proposto resolveu 93% das instâncias em menos de 30 segundos.*

1. Introdução

Problemas de empacotamento são frequentemente estudados na ciência da computação devido sua alta aplicabilidade na indústria. O problema de empacotamento em recipientes clássico foi abordado de diversas formas, incluindo algumas técnicas de algoritmos exatos. Uma variante do problema clássico, onde itens possuem classes, também foi estudada, porém os resultados obtidos se focaram em algoritmos de aproximação (e.g. [Shachnai and Tamir 2001], [Xavier and Miyazawa 2008] e [Epstein et al. 2010]). Neste trabalho, propomos atacar o Problema de Empacotamento em Recipientes com Restrições de Classe de maneira exata utilizando técnicas que se provaram eficazes para o problema clássico, como o *Branch-and-Price* [Vance et al. 1994], que utiliza do Problema da Mochila com Restrições de Classe como subproblema.

2. Problema de Empacotamento em Recipientes com Restrições de Classe

Dado um conjunto de recipientes de capacidade B, um limite C na quantidade de classes, e um conjunto de itens também $I = [n]$, onde cada item $i \in I$ possui um tamanho s_i e

*Este trabalho foi parcialmente financiado pelos processos nº 2013/21744-8 e 2014/25892-4, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e processos nº 311499/2014-7, 306358/2014-0 e 479070/2012-1, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq.)

uma classe $c_i \in [Q]$ sendo Q o número de classes diferentes na instância, o objetivo do CCBP é empacotar todos os itens utilizando a quantidade mínima de recipientes. Neste artigo, apresentamos algoritmos de *Branch-and-Price* para encontrar soluções exatas para o CCBP. Considerando P o conjunto de todos os padrões viáveis (subconjuntos de itens que podem ser empacotados em um mesmo recipiente) possíveis, e λ^p, uma variável binária que indica se um padrão p é usado na solução, podemos descrever o CCBP como o problema de minimizar a soma de λ^p, para todo $p \in P$, de forma que cada item esteja em pelo menos um dos padrões utilizados na solução [Gilmore and Gomory 1961].

Em nossos algoritmos utilizamos a regra de ramificação simples [Vance 1998] para o CCBP. Ela consiste em escolher uma coluna p que possua valor fracionário após a solução da relaxação do problema mestre, e forçar duas situações opostas: de um lado, a coluna p deve ser sempre incluída na solução ($\lambda^p = 1$); e do lado oposto, a coluna p não poderá fazer parte da solução ($\lambda^p = 0$), ou seja, será um padrão proibido. Como o número de padrões viáveis é exponencial, escolhemos um conjunto inicial de padrões e geramos novos padrões durante a execução do algoritmo conforme necessário. Este processo é chamado de *pricing*, e para o CCBP, recai no Problema da Mochila com Restrições de Classe (CCKP) com padrões proibidos onde, dados um conjunto de itens, onde um item i tem um valor v_i, classe c_i, e tamanho s_i, e uma lista de padrões proibidos, o objetivo é selecionar um subconjunto de valor máximo deste itens satisfazendo as restrições de capacidade e do limite de itens de classes diferentes na mochila, de forma que este subconjunto não forme uma solução presente na lista de padrões proibidos, dada como entrada.

Em nossos algoritmos, implementamos dois métodos de exploração da árvore de busca. O primeiro trata-se do *Best-Bound*, no qual cada nó é armazenado e indexado pelo seu valor de relaxação, e os nós com menor valor de relaxação são explorados primeiro, obedecendo uma fila de prioridades. No segundo método, *Diving*, escolhemos um lado para ser sempre explorado primeiro até que uma solução inteira seja encontrada. Uma vez encontrada a solução inteira, o nó com menor relaxação linear é escolhido para o início do próximo mergulho. Para a formulação proposta, escolhemos explorar primeiro o lado da sub-árvores no qual uma coluna é fixada na solução. Desta forma, postergamos a exploração de nós onde o *pricing* requer evitar padrões proibidos.

3. Problema da Mochila Com Restrições de Classe

Como mencionado na seção anterior, o problema de *pricing* que devemos resolver recai no CCKP com padrões proibidos. Porém, é interessante considerar também algoritmos para a versão sem padrões proibidos, já que por vezes não temos tais padrões (como na raiz da árvore de busca) e mesmo que tivermos, podemos executar um tal algoritmo com a esperança que o mesmo encontre um padrão não proibido e, caso o mesmo encontre um padrão proibido, podemos utilizar outro algoritmo como *fallback*. Nesta seção discutimos os algoritmos implementados para resolver este problemas. Além dos algoritmos discutidos nesta seção, consideramos também uma formulação natural do programa linear inteiro para este problema (com padrões proibidos), que foi resolvida utilizando o Gurobi.

DPI Quando não temos padrões proibidos, considere $U(b, c, q)$ como o valor máximo de uma solução de tamanho até b, usando até c classes de 1 até q. Considere também que para cada classe $i \in [Q]$, temos que n_i representa a quantidade de itens da classe i. Para encontrar $U(b, c, q)$, precisamos primeiro computar, para cada classe $i \in [Q]$, o

valor máximo de uma solução de tamanho s, utilizando apenas itens da classe i. Considere $U_i'(s, j)$ como o valor máximo de uma solução de tamanho até s, utilizando apenas os primeiros j itens da classe i. Este valor é calculado com a recorrência $U_i'(s, j) = \max\{U_i'(s, j-1), U_i'(s - s_j, j-1) + v_j\}$. Com isso, podemos calcular o valor máximo de uma solução de tamanho até s, utilizando apenas itens da classe i como $U_i'(s, n_i)$, para cada classe em $[Q]$. A partir destes valores, podemos finalmente computar $U(b, c, q)$ como

$$U(b,c,q) = \max \begin{cases} U(b,c,q-1) \\ \max\limits_{1 \le b' \le b} \{U(b-b', c-1, q-1) + U_q'(b', n_q)\}, \end{cases}$$

sendo que o valor de uma solução ótima é dado por $U(B, C, Q)$.

DPII Novamente, considerando que não temos padrões proibidos, considere agora os itens ordenados de acordo com as classes as quais eles pertencem. Isto nos garante que todos os itens pertencentes a uma mesma classe estarão em sequência. Considere $V(i, b, c, u)$ como o valor máximo de uma solução utilizando itens de 1 até i, tamanho máximo da mochila b, utilizando até c classes diferentes, e tendo u como a variável indicadora que mostra se a classe do item i é utilizada ou não na solução. Considere também, l_i como o índice do último item pertencente a classe anterior a de i. Com isso, podemos computar $V(i, b, c, u)$ de forma que

$$V(i,b,c,0) = \quad \max\left\{V(l_i,b,c,0), V(l_i,b,c,1)\right\}$$

$$V(i,b,c,1) = \begin{cases} \max\left\{V(i-1,b,c,1), V(i-1,b-s_i,c,1) + v_i\right\} & \text{caso } c_i = c_{i-1} \\ \max \begin{cases} V(i-1,b,c-1,1), \quad V(i-1,b,c-1,0) \\ V(i-1,b-s_i,c-1,1) + v_i, \quad V(i-1,b-s_i,c-1,0) + v_i \end{cases} & \text{caso } c_i \neq c_{i-1} \end{cases}$$

sendo que o valor de uma solução ótima é dado por $\max\{V(n, B, C, 0), V(n, B, C, 1)\}$.

Branch-and-Bound: Implementamos um algoritmo de *Branch-and-Bound* no qual exploramos os itens em ordem não crescente de eficiência, definida por v_i/s_i. Calculamos o limitante utilizando um algoritmo guloso. Durante o processo de atualização do valor incumbente, utilizamos uma estrutura de *hash* para checar se o padrão obtido nesta ramificação esta na lista de padrões proibidos e, em caso positivo, esta solução não é utilizada, e o algoritmo continua em busca de uma solução não proibida.

4. Experimentos e Resultados

Realizamos experimentos em 320 instâncias com diferentes valores de Q, C, n, B e intervalo de tamanho de cada item. Registramos o tempo necessário até a obtenção da solução ótima com um limite de 1800 segundos para cada instância. Utilizamos o Gurobi para resolver os limitantes e relaxações de programação linear inteira. Nestes experimentos, consideramos diferentes combinações de algoritmos para resolver o *pricing* assim como técnicas de exploração da árvore. Os algoritmos de Branch & Bound e o Programa Linear Inteiro (resolvido pelo *Gurobi*) são os únicos algoritmos que resolvem o problema de maneira definitiva, uma vez que os algoritmos de Programação Dinâmica podem retornar padrões proibidos. Além disso, o tempo de execução da DPI é $\Theta(BCQ + nB)$, enquanto que o tempo de execução da DPII é $\Theta(nBC)$ e, assim, dependendo da instância uma pode ser melhor do que a outra.

A Figura 1, apresenta a fração das instâncias resolvidas em função do tempo, considerando as combinações dos algoritmos de *pricing* básicos, *Branch-and-Bound* (B&B)

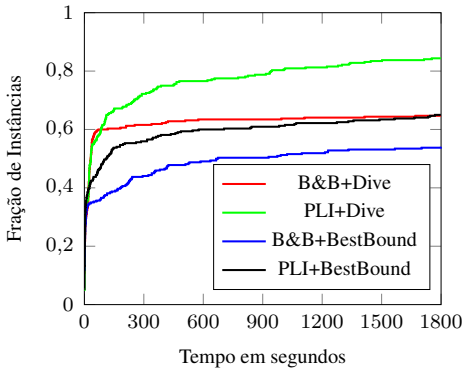

Figura 1. Básicos

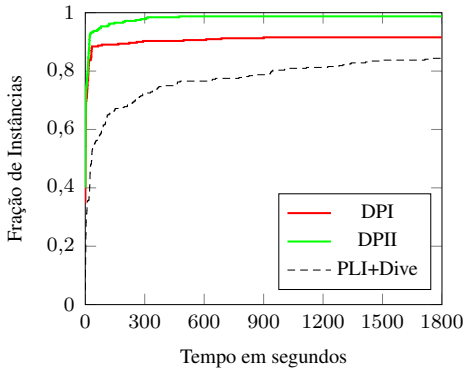

Figura 2. DPI / DPII

e Programação Linear Inteira (PLI), e os métodos de exploração *Diving* e *Best-Bound*. Podemos observar que o método *Diving* apresenta resultados melhores independente do algoritmo de *pricing* utilizado, e embora o algoritmo de B&B resolva até 63% das instâncias mais rápido do que sua contraparte em PLI, a diferença no tempo de solução para estas instâncias não é grande. Por outro lado, a combinação PLI e *Diving* consegue resolver cerca de 85% das instâncias antes do limite de tempo, enquanto B&B e *Diving* resolve em torno de 64%. A Figura 2, apresenta a melhor solução do Figura 1 e as incrementa com o uso dos algoritmos programação dinâmica para acelerar a geração de colunas. Tais algoritmos sempre são utilizados, mesmo na presença de padrões proibidos. Porém, caso a solução encontrada seja de fato um padrão proibido, nós resolvemos o problema utilizando o PLI. Podemos observar que o acréscimo das DPs melhora consideravelmente o tempo de solução das instâncias. Para a melhor combinação, *Diving* + PLI acrescentado de DPII, temos 93% das instâncias resolvidas em menos de 30 segundos, e totalizando em torno de 99% até o limite de tempo.

Estes resultados nos dão confiança para continuar atacando este problema com esta técnica, uma vez que não existem outros estudos experimentais publicados sobre este problema para comparação direta. Estamos realizando experimentos adicionais envolvendo outros algoritmos para resolução do *pricing*, assim como outro modelo de ramificação, sob um conjunto de instâncias ainda maior.

Referências

Epstein, L., Imreh, C., and Levin, A. (2010). Class constrained bin packing revisited. *Theoretical Computer Science*, 411(34):3073–3089.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Shachnai, H. and Tamir, T. (2001). Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, 4(6):313–338.

Vance, P. H. (1998). Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimimization and Applications*, 9(3):211–228.

Vance, P. H., Barnhart, C., Johnson, E. L., and Nemhauser, G. L. (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3(2):111–130.

Xavier, E. C. and Miyazawa, F. K. (2008). The class constrained bin packing problem with applications to video-on-demand. *Theoretical Computer Science*, 393(1):240–259.