Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Azevedo Moscoso Silva Cruz

Algoritmo Price-and-Cut com 3-SRCs e Enumeração de Colunas para o Problema de Alocação Generalizada

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC–Rio

Orientador: Prof. Marcus Vinicius Soledade Poggi de Aragão

Rio de Janeiro
Abril de 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Azevedo Moscoso Silva Cruz

Algoritmo Price-and-Cut com 3-SRCs e Enumeração de Colunas para o Problema de Alocação Generalizada

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC–Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Marcus Vinicius Soledade Poggi de Aragão
Orientador
Departamento de Informática — PUC–Rio

Prof. Thibaut Victor Gaston Vidal
Departamento de Informática — PUC–Rio

Prof. Rafael Martinelli Pinto
Departamento de Engenharia Industrial — PUC–Rio

Prof. Anand Subramanian
Departamento de Sistemas de Computação — UFPB

Rio de Janeiro, 23 de Abril de 2021

Rafael Azevedo Moscoso Silva Cruz
Graduou-se em Engenharia da Computação na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) em 2018. Durante a graduação, atuou como pesquisador do CNPq no departamento de Engenharia Elétrica e participou da primeira edição da Apple Academy na PUC-Rio. Estudou na Ecole Nationale Supérieure des Télecommunications (Télecom Paris) com o apoio da CAPES no programa de intercâmbio BRAFITEC e desenvolveu um projeto científico para a Enedis (Paris). Durante o Mestrado, como bolsista da CAPES do departamento de Informática, pesquisou e desenvolveu um método *price-and-cut* com 3-SRCs e enumeração de colunas para solucionar instâncias do PAG. Atualmente, trabalha como pesquisador no instituto Tecgraf para atender às demandas tecnológicas da indústria de óleo e gás.

Ficha Catalográfica

Cruz, Rafael Azevedo Moscoso Silva

Algoritmo Price-and-Cut com 3-SRCs e Enumeração de Colunas para o Problema de Alocação Generalizada / Rafael Azevedo Moscoso Silva Cruz; orientador: Prof. Marcus Vinicius Soledade Poggi de Aragão. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2021.

v., 104 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Tese. 2. Otimização. 3. PAG. 4. Decomposição. 5. Geração de Colunas. 6. Enumeração de Colunas. 7. Programação Inteira. 8. Desigualdades 3-SRC. 9. Geração de Colunas Estabilizada. I. Poggi de Aragão, Marcus. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Aos professores, familiares e amigos que foram inspiração para a minha formação
ao longo de anos de desenvolvimento pessoal, acadêmico e profissional.

Agradecimentos

Aos meus pais e demais familiares que sempre incentivaram os meus estudos com incansável apoio à minha formação acadêmica.

Ao meu orientador Marcus Poggi, sempre atencioso, que esteve prontamente disposto a prover esclarecimentos durante o desenvolvimento do trabalho e a apoiar a resolução de obstáculos encontrados ao longo da pesquisa.

À Daniela Mayumi pelo contínuo carinho e especialmente pela firme compreensão e pelo incessante apoio durante todo o mestrado.

À Pontifícia Universidade Católica do Rio de Janeiro que contribuiu fundamentalmente para o meu desenvolvimento acadêmico desde a graduação em engenharia da computação, oferecendo uma formação de excelência para que este trabalho de mestrado pudesse ser realizado.

Aos amigos e colegas do instituto Tecgraf, sobretudo à equipe J5, que estiveram presentes neste período de dedicação acadêmica do mestrado. Agradeço também a todos os colegas, professores e demais profissionais da pós-graduação do departamento de Informática da PUC-Rio.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo

Cruz, Rafael Azevedo Moscoso Silva; Poggi de Aragão, Marcus. **Algoritmo Price-and-Cut com 3-SRCs e Enumeração de Colunas para o Problema de Alocação Generalizada**. Rio de Janeiro, 2021. 104p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Esta dissertação estuda formulações, algoritmos e métodos exatos para resolver instâncias do Problema de Alocação Generalizada (PAG) com uma separação de desigualdades (3, 0.5)-SRC que viabilize a enumeração de colunas. Este trabalho é motivado pela perspectiva de alcançar o estado-da-arte com resultados competitivos comparáveis às melhores soluções encontradas na literatura por Avella (2010) e Michelon (2012). A pesquisa abrange métodos exatos e heurísticas, com ênfase no estudo que aborda a decomposição de Dantzig-Wolfe, o algoritmo de geração de colunas, a estabilização de colunas por meio da ponderação de duais proposto por Wentges (1997) e a enumeração de colunas habilitada pela minimização do *gap* decorrente do algoritmo de *price-and-cut*. O algoritmo de *price-and-cut* desenvolvido recorre à geração de colunas (*pricing*) aliada à separação de (3, 0.5)-SRCs para aumentar o *lower bound* gerado, assim minimizando o *gap*. A geração de colunas implementada é inspirada no algoritmo de Savelsbergh (1997); e a separação de (3, 0.5)-SRCs é motivada pelo trabalho de Jepsen (2008) e pelo algoritmo branch-cut-and-price proposto por Poggi e Uchoa (2016) para o CVRP. De acordo com os experimentos computacionais, as desigualdades adotadas são capazes de reduzir o *gap* suficientemente para viabilizar a enumeração de colunas em diversas instâncias do PAG com até 200 tarefas e 20 máquinas. O método utilizado obteve resultados compatíveis às melhores soluções conhecidas, enumerando todas as colunas necessárias para cobrir o *gap* determinado pelo *price-and-cut*. Esse resultado incentiva futuras pesquisas para estender a aplicação do algoritmo a instâncias maiores e mais difíceis.

Palavras–chave

Otimização; PAG; Decomposição; Geração de Colunas; Enumeração de Colunas; Programação Inteira; Desigualdades 3-SRC; Geração de Colunas Estabilizada;

Abstract

Cruz, Rafael Azevedo Moscoso Silva; Poggi de Aragão, Marcus (advisor). **Price-and-Cut Algorithm with 3-SRCs Cuts and Column Enumeration for the Generalized Assignment Problem**. Rio de Janeiro, 2021. 104p. MSc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This dissertation deals with formulations, algorithms and exact methods for solving the well-known Generalized Assignment Problem (GAP) through a price-and-cut approach with the separation of (3, 0.5)-SRC inequalities in order to improve column enumeration feasibility and efficiency. This work is motivated by the perspective of reaching state-of-the-art performance, attaining competitive results which are comparable with the best known solutions found in the literature by Avella (2010) and Michelon (2012). This research was build on exact methods and some heuristics with emphasis on the Dantzig-Wolfe decomposition, the column generation algorithm, the stabilization through weighted Dantzig-Wolfe decomposition proposed by Wentges (1997) and finally the column enumeration motivated by the gap minimization reached through the price-and-cut algorithm. The price-and-cut algorithm proposed here resort to column generation (pricing) combined with the separation of (3, 0.5)-SRC cuts in order to increase the generated lower bound, thus minimizing the attained gap. This column generation algorithm follows the work of Savelsbergh (1997); and the separation of (3, 0.5)-SRCs is formulated by Jepsen (2008) and motivated by the branch-cut-and-price algorithm proposed by Poggi and Uchoa (2016) for the CVRP. According to computational experiments, the adopted inequalities are capable of sufficiently reducing the gap, assuring the feasibility of column enumeration for several GAP instances with up to 200 tasks and 20 machines. This method achieved expressive results, compatible with the best known solutions, enumerating all the necessary columns to cover the gap found by the price-and-cut. Therefore, these results motivate future research towards the extension of the method's applicability to larger and more complex instances.

Keywords

Optimization; GAP; Decomposition; Column Generation; Column Enumeration; Integer Programming; 3-SRC Inequalities; Stabilized Column Generation;

Sumário

Lista de figuras

Lista de tabelas

Lista de *scripts*

1
Introdução

O conteúdo desta dissertação está integralmente ligado à pesquisa de algoritmos de otimização, abordando especificamente o problema de alocação generalizada (PAG) que possui aplicação prática muito direta em diversas operações e processos da atividade humana.

A alocação de recursos a limitados meios produtivos sob a penalização de custos é um problema recorrente não apenas na literatura acadêmica, como também no contexto de problemas reais. Em contextos reais, a minimização do custo de alocação de recursos é altamente desejável, sobretudo para a atender eficientemente a crescente demanda por bens e serviços e também para responder à produtividade competitiva em diversos segmentos da sociedade. Esse cenário que extrapola a pesquisa acadêmica motiva ainda mais a busca por técnicas mais eficientes de tomada de decisão para o Problema de Alocação Generalizada (PAG). Até mesmo em situações cotidianas, indivíduos e organizações são frequentemente desafiados a decidir alocações de tempo, de materiais e de tarefas que minimizem os custos, normalmente atribuições sujeitas a rígidas restrições logísticas, financeiras, legais e de tantas outras naturezas quanto à disposição e capacidade de recursos operacionais.

Na prática, os processos conduzidos por setores sociais, sobretudo empresariais, dependem de procedimentos para resolver problemas bem definidos por um conjunto de variáveis de decisão, visando alocar os seus recursos operacionais a um custo mínimo, sem violar as restrições de capacidade dos meios de operação e garantindo o melhor retorno possível.

Em geral, operações de alocação de recursos ocorrem em contextos descritos por variáveis de decisão características, seja a escolha de atribuição de tarefas a agentes (recurso humano), a seleção de rotas de entrega no problema de roteamento (recurso geográfico), a aplicação de capitais em fundos de investimento (recurso financeiro) ou o escalonamento de arquivos para um grupo de máquinas (recurso material).

Em problemas de natureza combinatória, a abordagem mais imediata e determinística para escolher o padrão de alocação perfeita, mais lucrativo ou com menos penalização, seria proceder à avaliação de cada combinação de decisões

para encontrar a solução ótima, de custo mínimo, seja a melhor rota de entrega ou a melhor atribuição de tarefas segundo os exemplos descritos anteriormente. Entretanto, uma decisão é um estado binário no contexto dos problemas analisados neste trabalho. Dado um recurso e um agente, a decisão é a escolha entre atribuir ou não atribuir esse recurso a esse agente. Apenas seguindo o método de enumeração, o crescimento exponencial do número de sequências de decisão em função do número de recursos e agentes (variáveis de decisão) do problema revela uma quantidade proibitiva de soluções inteiras viáveis.

Em vista disso, a enumeração de todos padrões de decisão deve ser evitada, cabendo a pertinência do estudo de métodos que reduzam o espaço de soluções viáveis candidatas à otimalidade, procurando a melhor configuração de atribuições dentro de menores subconjuntos de soluções viáveis. Entre os problemas de combinatória conhecidos, há o problema da alocação generalizada (PAG) que serve de objeto de estudo nesta dissertação, tratando a minimização de custo em contextos de alocação de recursos.

Em relação à análise de complexidade, o PAG pertence à classe NP-Difícil para a qual não são conhecidos métodos capazes de encontrar soluções ótimas em tempo polinomial. Embora o conjunto de possibilidades seja teoricamente finito, computacionalmente não há com avaliar explicitamente cada solução inteira a ponto de encontrar a solução ótima. É mais fácil encontrar soluções viáveis como *upper bounds*, sem comprovação de otimalidade. Esse caráter impeditivo da enumeração torna essa abordagem impraticável em contextos escaláveis, limitando a eficácia desse algoritmo. Então, essa dificuldade requer estratégias de otimização combinatória sofisticadas, abrangendo heurísticas, métodos exatos e algoritmos de Branch-Cut-and-Price (BCP).

Em meio aos algoritmos já pesquisados, a principal estratégia explorada tenta, de alguma forma, minimizar o espaço de soluções viáveis próximas do ótimo, analisando um enumerável e reduzido subconjunto de possibilidades à procura de soluções de qualidade em tempo polinomial, por vezes com otimalidade comprovada. O algoritmo disposto nesta dissertação investiga um método exato que tenta exatamente diminuir o espaço de enumeração a ponto de viabilizá-la.

Os avanços da comunidade acadêmica quanto aos métodos de otimização combinatória enriquecem a compreensão de técnicas computacionais para solucionar difíceis problemas operacionais. A contribuição deste trabalho pretende servir ao mesmo propósito.

Este trabalho disserta sobre um método de *price-and-cut* (PC) que adota desigualdades 3-SRCs, geração de colunas e enumeração de colunas para alcançar o estado-da-arte, melhorando a solução conhecida de algumas instâncias do problema de alocação generalizada.

1.1 Motivação

O Problema de Alocação Generalizada (PAG) é um problema de otimização combinatória com aplicações em diversos cenários de roteamento, escalonamento de recursos, localização de recursos e outras tantas operações de logística (Temel Oncan, 2007). Em termos de definição, o PAG é uma generalização do Problema de Alocação que, por sua vez, representa a subsestrutura de outros problemas como o Problema de Coloração de Grafos e o Problema de Partição de Grafos (Ferland, 1997). Assim sendo, a sua forte relevância se mostra consistente no meio acadêmico e também em aplicações reais.

Em relação à dissertação apresentada, o conteúdo disposto compreende o estudo e a implementação de um método de *price-and-cut* baseado na geração de colunas definida por Savelsbergh (1997) para o modelo de decomposição de Dantzig-Wolfe. Esse método inclui ainda a inserção de desigualdades $(C, p) - SRC$ (Chvátal-Gomory) aplicadas às restrições de exclusiva alocação de tarefa, sobretudo (3, 0.5)-SRCs que são *subset row cuts* de cardinalidade 3 com multiplicadores 0.5. Através da geração de colunas e da introdução de cortes, a diferença entre *lower* e *upper bounds* pode ser reduzida com a meta de facilitar a enumeração de colunas.

A ideia da adição de desigualdades é aumentar ainda mais o *lower bound* obtido pela geração de colunas para estreitar ao máximo o *gap* entre o *lower bound* e o *upper bound* conhecidos. Essa redução facilita a enumeração de colunas necessárias para compor um modelo mestre viável por programação inteira, considerando que o número de colunas na formulação de Dantzig-Wolfe torna o modelo mestre intratável, obrigando a adoção de técnicas para acrescentar apenas um número tratável de colunas na construção de um modelo mestre restrito. O *lower bound* e o *upper bound* serão denominados LB e UB, respectivamente, ao longo do texto desta dissertação.

O método proposto também considera uma estratégia de estabilização para convergência de algumas instâncias; a adaptação de uma eficiente implementação de programação dinâmica para o problema de mochila (*pricing*) que considere as SRCs; e ainda a fixação por custo reduzido de variáveis de atribuição nos subproblemas de mochila.

Em resumo, o objetivo fundamental desta dissertação é a apresentação de um algoritmo *price-and-cut* capaz de alcançar o estado-da-arte para o PAG através da enumeração de colunas a partir da resolução da decomposição de Dantzig-Wolfe, com adição de desigualdades (3, 0.5)-SRCs. A hipótese considerada é que o consequente aumento do LB conhecido reduzirá suficientemente o *gap* entre os limites inferior e superior a ponto de viabilizar e facilitar a enumeração de colunas.

A enumeração de colunas pretende construir um modelo de decomposição restrito suficiente que possa ser resolvido com programação inteira (IP). Desse modo, o método pretendido deve adicionar apenas colunas enumeradas com custo reduzido menor que o *gap* final ao modelo mestre restrito inteiro na busca pela solução inteira localizada entre os bounds encontrados pelo BCP e idealmente próxima da quase otimalidade.

Quanto à introdução de $(3, 0.5) - SRCs$, a estratégia consiste em aumentar mais o LB obtido pela geração de colunas para estreitar ao máximo o *gap* entre os *bounds*. Dado que a formulação de Dantzig-Wolfe conduz a um modelo de decomposição irrestrito com um número de colunas de tamanho exponencial, representando todas as possíveis soluções para cada problema de mochila, a estratégia de trabalhar com um mestre restrito é uma enorme vantagem tendo em vista que as colunas podem ser adicionadas iterativamente pela geração de colunas com base no custo reduzido computado pelo *pricing* de cada subproblema de máquina.

Portanto, a inserção de desigualdades contribui para a otimização da enumeração de colunas que pertençam à região convexa de viabilidade do mestre restrito inteiro.

As instâncias de PAG sobre as quais o algoritmo será testado pertecem à biblioteca OR-Library, abrangendo todas as classes de instância A, B, C, D e E. A lenta convergência da geração de colunas observada para algumas instâncias está sendo acelerada com base no emprego da técnica de estabilização proposta por Pigatti *et al.* (2005).

1.2 Literatura

O Problema de Alocação Generalizada (PAG), introduzido em Ross and Soland (1975), aparece na literatura como um problema de otimização combinatória com uma extensa aplicabilidade, servindo para modelar outros problemas como o roteamento de transportes (VRP), o escalonamento de recursos (Cattrysse and Van Wassenhove, 1994), alguns desafios de logística, alguns desafios de supply chain e até mesmo problemas na área de telecomunicações.

Em relação aos problemas de escalonamento de recursos, o PAG serve de modelo para otimizar atribuições pertinentes como a atribuição de tarefas a máquinas, conforme mencionado em Temel Oncan (2007). O PAG também aparece como subproblema de instâncias do VRP, segundo estudo para a resolução de instâncias do TSP em Fisher and Jaikumar (1981). Em telecomunicações, a configuração ótima do Border Gateway protocol (BGP) também pode ser modelado como um PAG, segundo demonstração de Bressound (2003).

Em logística e supply chain, o Problema de Partição de Demanda (DPP) também é mencionado como uma generalização do PAG por Benjaafar (2004). De acordo com Freling and Wagelmans (2003), outro problema de logística que também pode ser modelado e resolvido como um PAG é o *Multi-Period Single Sourcing Problem* (MPSSP) que atribui grupos de clientes a depósitos de modo que cada cliente esteja associado a exatamente um único depósito em cada período de tempo. A extensão das aplicações do PAG e suas formulações variantes está detalhadamente documentada no trabalho de Temel Oncan (2007) que apresenta uma análise da presença do PAG na literaturaem anos antecedentes a 2007.

A intratabilidade de instâncias do PAG decorre do fato desse problema ser classificado como NP-Difícil, como demonstrado em Sahni and Gonzalez (1976). Contudo, o PAG é reconhecido inúmeras vezes como uma subestrutura na formulação de muitos outros problemas de otimização combinatória (Temel Oncan, 2007), como a coloração de grafos. Consequentemente, isso revela a importância de métodos eficientes dedicados ao PAG. Segundo Fisher and Jaikumar (1981), encontrar soluções viáveis para o PAG é um problema NP-Complete. Em verdade, a literatura propõe sobretudo algoritmos de *branch-and-price* (Savelsbergh, 1997), algoritmos de branch-and-cut-and-pricebranch-and-cut-and-price (Pigatti *et al.*, 2005) e um conjunto de heurísticas e metaheurísticas como as descritas em Yagiura and Ibaraki (2004). Esses algoritmos se esforçam para resolver o maior número possível de instâncias difíceis do PAG, chegando mesmo a alcançar soluções que se aproximam da quase otimalidade, em alguns casos com otimalidade comprovada.

O algoritmo *branch-and-price* é introduzido por Savelsbergh (1997) com o recurso à geração de colunas asssociado ao subproblema de atribuição de tarefas em cada máquina. Mais tarde, o mecanismo de estabilização implementado em Pigatti *et al.* (2005) acelera a convergência da geração de colunas do algoritmo de *branch-and-price*, adicionando variáveis de penalidade à formulação para corrigir flutuações das soluções duais durante a geração de colunas. Em 1997, outro método de estabilização apresentado por Wentges (1997) propõe uma decomposição de Dantzig-Wolfe ponderada cuja geração de colunas avalia a relaxação Lagrangiana do PAG, evitando a degenerescência de instâncias durante a geração de colunas. Essa técnica de estabilização foi adotada pelo algoritmo *price-and-cut* desta dissertação, resolvendo instâncias degeneradas mais eficientemente.

Entre os artigos mais recentes, Avella (2010) descreve uma implementação eficiente de um método exato para encontrar hiperplanos (cortes) através de um problema de otimização definido sobre a solução conhecida para cada polítopo convexo dos subproblemas de mochila. Avella (2010) classifica as instâncias do PAG da biblioteca OR em categorias de dificuldades, mostrando resultados interessantes que alcançam solução ótima para algumas instâncias difíceis. Em

2012, Michelon (2010) implementa um método exato branch-and-bound com fixação de variáveis usando relaxação Lagrangiana para resolver instâncias do PAG, obtendo também resultados interessantes para diversas instâncias. Os resultados computacionais desses artigos são usados como referência para avaliar os resultados computacionais do algoritmo dessa dissertação.

Diante desta literatura, apesar dos resultados de algoritmos *branch-and-price* conduzirem a reduções significativas do *gap* entre o LB e o UB, outros trabalhos, como Fukasawa and Goycooles (2007), sugerem a separação de cortes do subproblema de mochila da geração de colunas para reduzir ainda mais o *gap* entre os *bounds*. Portanto, há evidências de que a introdução de cortes adequados pode vir a estreitar a distâncias entre *bounds*. O algoritmo desta dissertação investe nessa estratégia de adicionar cortes ao mestre restrito formado pela decomposição e pela geração de colunas, sendo realizada a inserção de desigualdades 3-SRCs, aumentando o LB do mestre restrito relaxado para tornar a enumeração de colunas mais tratável.

Entretanto, a separação e a adição de desigualdades ao modelo de decomposição Dantzig-Wolfe eventualmente torna o *pricing* do subproblema da mochila mais custoso, podendo inviabilizar a resolução do subproblema em tempo hábil. Isso ocorre porque as desigualdades alteram a formulação da decomposição de Dantzig-Wolfe e consequentemente a separação de cortes altera a estrutura do subproblema.

O trabalho de Poggi and Uchoa (2003) aponta uma formulação robusta designada como o mestre explícito, também uma formulação de decomposição, para a qual o *branching* e a adição de desigualdades são mais facilmente tratados. Embora não tenha como alvo o PAG, o artigo Poggi and Uchoa (2016) deve ser referido por apresentar a definição de algumas $(C, p) - SRCs$ no problema de roteamento (VRP), inclusive projetando uma adaptação otimizada para adição e utilização de SRCs que diminui o impacto computacional negativo, em termos de eficiência, sobre o *pricing*. Essas desigualdades podem funcionar para o PAG, seguindo a mesma definição.

Nesta dissertação, os valores de referência para avaliar o estado da arte são retirados tanto de Avella (2010) quanto de Michelon (2010).

1.3 Objetivos e Contribuições

Esta dissertação estuda formulações, algoritmos e métodos exatos para resolver instâncias do Problema de Alocação Generalizada (PAG) com uma separação de desigualdades (3, 0.5)-SRC que viabilize a enumeração de colunas.

A contribuição desta pesquisa é o desenvolvimento de um algoritmo *price-and-cut* com desigualdades (3, 0.5)-SRC que maximiza o LB do modelo de formulação de Dantzig-Wolfe para o PAG. Essa maximização do LB equivale à minimização do *gap*. Essa minimização restringe as colunas visitadas pela enumeração de colunas. Portanto, o algoritmo proposto facilita a enumeração de colunas com o objetivo de encontrar uma solução inteira para instâncias do PAG, traçando os objetivos indicados abaixo.

1. Reduzir o *gap* em relação à formulação padrão do PAG a partir da convexificação das restrições de capacidade de máquina (decomposição de Dantzig-Wolfe);

2. Otimizar a enumeração de colunas a partir do *gap* mínimo encontrado, restringindo a quantidade de colunas consideradas pela programação inteira do modelo de decomposição;

3. Provar a otimalidade da solução inteira encontrada para instâncias do PAG;

4. Encontrar uma solução inteira que seja melhor ou equivalente à melhor solução inteira da literatura para instâncias do PAG.

O algoritmo *price-and-cut* com separação de desigualdades (3, 0.5)-SRC e enumeração de colunas é um método que implementa tanto métodos exatos quanto algumas heurísticas para otimizar a enumeração de colunas, gerando uma base de colunas suficiente para viabilizar uma solução inteira, idealmente ótima, para o modelo inteiro da formulação de Dantzig-Wolfe, restrito a essa base de colunas, para instâncias do PAG.

O LB maximizado do *price-and-cut* é uma condição fundamental para viabilizar a enumeração de colunas, dado que o número de colunas enumeráveis pode ser reduzido para um subconjunto composto apenas por colunas com custo reduzido $\bar{r}_k \leq UB - LB$, onde LB é o LB do *price-and-cut* e o UB é o melhor UB conhecido para o PAG, segundo outros algoritmos já consolidados para o PAG, como Avella (2010) e Michelon (2010).

Portanto, o método defendido neste texto para solucionar instâncias do PAG pretende alcançar o estado-da-arte, recorrendo às técnicas e aos conceitos listados abaixo, exatamente nesta ordem de execução do fluxo implementado.

1. Decomposição de Dantzig-Wolfe, estabelecendo uma convexificação de restrições de capacidade de máquina que reduza o *gap* em relação à formulação padrão, utilizando as soluções inteiras da região decomposta;

2. Geração de colunas para resolver a relaxação linear do modelo de formulação Dantzig-Wolfe através do *pricing*;

3. Modelo de programação inteira para resolver subproblemas de mochila (*pricing*), otimizando a atribuição de tarefas a cada máquina;

4. Técnica de estabilização para tratar a degenerescência de instâncias na geração de colunas, fundamentada na formulação de Dantzig-Wolfe ponderada definida em Wentges (1997).

Em sequência, o algoritmo recorre à implementação de outras técnicas que são contribuições particulares desta dissertação para instâncias PAG. Essas contribuições estão listadas abaixo.

1. Separação de desigualdades do tipo (3, 0.5)-SRC para aumentar o LB da relaxação linear do modelo de decomposição de Dantzig-Wolfe;

2. Modelo de programação inteira para resolver problemas de mochila com restrições adicionais associadas às desigualdades (3, 0.5)-SRC;

3. Algoritmo de programação dinâmica para resolver problemas de mochila com restrições adicionais associadas às desigualdades (3, 0.5)-SRC;

4. Enumeração de colunas, sujeita ao *gap* minimizado, com potencial para encontrar uma solução inteira melhor ou equiparável ao resultado estabelecido pelo estado-da-arte, provando eventualmente a sua otimalidade.

1.4 Organização do Texto

O texto desta dissertação está estruturado de modo a apresentar inicialmente uma definição do Problema de Alocacação Generalizada (PAG) e prosseguir para a exposição da fundamentação teórica de algoritmos de geração de colunas, descrevendo técnicas e conceitos fundamentais para o desenvolvimento deste trabalho. Em seguida, o texto aborda os algoritmos, as formulações e as técnicas que caracterizam contribuições particulares desta dissertação.

1. O capítulo intitulado "Introdução" introduz os contextos de formulação e aplicação do PAG, explicitando a motivação da pesquisa desenvolvida nesta dissertação, apresentando uma revisão da literatura sobre métodos exatos e heurísticas desenvolvidos para o PAG e identificando os objetivos e contribuições deste trabalho;

2. O capítulo intitulado "Problema de Alocação Generalizada (PAG)" apresenta a definição do problema otimizado neste texto, introduzindo a formulação clássica do PAG;

3. O capítulo intitulado "Algoritmos de Geração de Colunas para o PAG" aborda as técnicas, os métodos e os conceitos que servem de fundamentação teórica, discorrendo sobre tópicos pertinentes aos algoritmos de geração de colunas, especificamente a decomposição de Dantzig-Wolfe, a estabilização de Wentges e os modelos de programação inteira e algoritmos de programação dinâmica para resolver subproblemas de mochila do *pricing*;

4. O capítulo intitulado "Introdução de Desigualdades (3, 0.5)-SRC" é a primeira contribuição, apresentando um algoritmo para realizar a separação de desigualdades do tipo (3, 0.5)-SRC capazes de aumentar o LB da relaxação linear do modelo de formulação Dantzig-Wolfe; esse capítulo apresenta a definição das (3, 0.5)-SRCs, propõe um algoritmo de separação desses cortes para o PAG e adapta o modelo de programação inteira e o algoritmo de programação dinâmica para resolver os subproblemas de mochila do *pricing* com a existência de desigualdades (3, 0.5);

5. O capítulo intitulado "Algoritmo de Enumeração de Colunas" também é uma contribuição desta dissertação que determina um algoritmo adaptado para enumerar colunas implicitamente, restringindo a extensão da enumeração de modo a torná-la significativamente mais eficiente a partir do *gap* minimizado anteriormente no *price-and-cut*;

6. O capítulo intitulado "Resultados Computacionais" analisa a eficácia e a eficiência do algoritmo *price-and-cut* com a separação de desigualdades (3, 0.5)-SRC e a enumeração de colunas, comparando as soluções obtidas com as soluções do estado-da-arte encontradas por Avella (2010) e Michelon (2010);

7. O capítulo intitulado "Trabalhos Futuros" discorre sobre possíveis direções de pesquisa para melhorar o desempenho e os resultados encontrados pelo algoritmo proposto nesta dissertação;

8. O capítulo intitulado "Conclusão" finaliza o texto, descrevendo brevemente os efeitos das contribuições deste trabalho.

2
Problema de Alocação Generalizada (PAG)

O Problema de Alocação Generalizada (PAG) determina a busca pela melhor atribuição de um conjunto finito de n tarefas a um conjunto finito e restrito de m máquinas de modo que cada tarefa j seja exclusivamente atribuída a uma única máquina i, tendo em vista a otimização do custo c_{ij} de execução das tarefas pelas máquinas, respeitando imperativamente a capacidade máxima de cada máquina i em relação aos pesos de atribuição w_{ij}.

2.1
Definição do Problema

Suponha que exista um conjunto de n tarefas para serem alocadas exclusivamente a m máquinas, de maneira que cada tarefa i precise ser alocada exclusivamente a uma máquina j. O objetivo é minimizar o custo total de alocação de todas tarefas às máquinas, sendo conhecidos os custos c_{ij} e os pesos w_{ij} de cada alocação (j, i). A especificação do PAG está estruturada conforme listado a seguir.

1. Seja c_{ij} o custo de atribuir uma tarefa j à máquina i, aumentando o custo total de alocação avaliado pela função objetivo.

2. Seja w_{ij} o peso depositado na máquina i pela tarefa j caso essa atribuição seja concretizada, subtraindo w_{ij} da capacidade limite disponível b_i da máquina i.

3. Considere que qualquer atribuição de tarefas à máquina i não pode exceder a capacidade máxima b_i.

4. Considere que cada tarefa j deve ser atribuída a exatamente uma máquina i (restrição de atribuição).

5. Seja $x_{ij} \in \{0, 1\}$ uma variável de decisão do PAG de modo que x_{ij} será 1 se e somente se a tarefa j estiver alocada à máquina i; caso contrário, x_{ij} terá valor 0.

Portanto, a otimização do PAG equivale à determinação de um conjunto de atribuições x_{ij} fixadas a 1 que não violem as restrições estabelecidas para o problema. Assim sendo, a formulação deste problema é naturalmente inteira, pela

própria definição de escolha binária de selecionar exclusivamente cada possível atribuição de tarefas às máquinas, atribuindo ou não uma dada tarefa j a uma dada máquina i. Em termos de nomenclatura, considere que máquinas e agentes são termos equivalentes, assim como tarefas e itens.

Em atenção à interpretação daquilo que significa um resultado viável da otimização do PAG, perceba que uma solução viável é uma sequência de decisões entre selecionar ou ignorar cada possível atribuição de tarefa j à máquina i, sempre respeitando as restrições. Naturalmente, o princípio de exclusividade da decisão de atribuição implica necessariamente que uma tarefa já alocada não poderá ser entregue a nenhuma outra máquina.

A partir da especificação do PAG apresentada, considere que a concessão de uma tarefa *j* a uma máquina *i* será representada pela variável de decisão x_{ij} de valoração binária. Caso a tarefa *j* seja processada pela máquina *i*, então x_{ij} assume obrigatoriamente valor 1; caso contrário, x_{ij} assume valor 0. Ademais, a integralidade dessas variáveis deve ser garantida.

$$x_{ij} \in \{0, 1\} \quad \forall\, i = 1, \ldots, m\ ;\ j = 1, \ldots, n \tag{2-1}$$

Embora o PAG apareça em alguns trabalhos da literatura com o objetivo de maximizar o lucro de associação de tarefas a agentes (Savelsbergh, 1997), outra definição comumente encontrada na literatura, descrita na equação (2-2), considera uma função objetivo de minimização dos custos da associação Temel Oncan (2007). Apesar de abordagens distintas, ambas definições são equivalentes no sentido de aceitar que minimizar custos corresponde a maximizar lucros.

$$Z_{IP} = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2-2}$$

Quanto à nomenclatura adotada, a genericidade do PAG confere ao problema uma certa flexibilidade quanto aos termos usados para designar os elementos que são atribuídos aos encarregados. Então, a definição às vezes recorre à terminologia de problemas de mochila, tratando da atribuição de itens a mochilas de capacidade limitada; às vezes recorre à terminologia de problemas de *scheduling*, tratando da atribuição de tarefas a agentes ou máquinas, segundo o estudo de Temel Oncan (2007).

O objetivo do PAG se resume a encontrar uma distribuição exclusiva das *n* tarefas às *m* máquinas, mantida a restrição de capacidades máximas permitidas para cada máquina e ainda ponderando que cada atribuição (j, i) acarreta um custo c_{ij} pela função objetivo Z_{IP} e contabiliza um peso w_{ij} de processamento da tarefa *j* subtraído da capacidade máxima da máquina i.

Desta maneira, a formulação padrão do PAG compreende duas classes de restrições. Primeiro, o conjunto de restrições que obrigam a atribuição exclusiva de cada tarefa j a uma máquina i disponível, como garantido pelas desigualdades (2-3).

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall \; j = 1, \ldots, n \tag{2-3}$$

Em segundo, há o conjunto de restrições de capacidade de máquina que estabelece um limite máximo de peso b_i suportado por cada máquina i. Esse limite de peso é garantido pelo conjunto de restrições, características de problemas de mochila, que asseguram o respeito pela capacidade máxima de peso atribuído a cada máquina dado um subconjunto de tarefas atribuídas, conforme as desigualdades (2-4).

$$\sum_{j=1}^{n} w_{ij} x_{ij} \; \leq \; b_i \quad \forall \; i = 1, \ldots, m \tag{2-4}$$

Essas restrições de capacidade possuem a característica peculiar de formarem uma estrutura de blocos independentes que podem ser tratados como subproblemas de mochila desagregados da formulação clássica.

A decomposição de Dantzig-Wolfe aproveita exatamente essa particularidade estrutural para transformar essa formulação do PAG em uma formulação mais fácil sujeita a um conjunto de subproblemas de mochila que regulam o custo reduzido (*pricing*) de colunas. Esse assunto é uma peça fundamental abordada no Capítulo 3 para o algoritmo de *price-and-cut*.

2.2 Formulação Padrão

A formulação clássica do Problema de Alocação Generalizada (PAG) admite um conjunto de variáveis de decisão X, definido em (2-5), havendo uma variável para cada possível atribuição de tarefa a máquina, com custos $c_{ij} \in C$ e pesos $w_{ij} \in W$ de alocação, sendo conhecidas as capacidades máximas $b_i \in B$ de cada máquina i em conformidade com a especificação do PAG já conhecida.

$$X \; = \; \{ \, x_{ij} \in \{0, 1\} \mid i \; = \; 1, \ldots, m, \; j \; = \; 1, \ldots, n \, \} \tag{2-5}$$

$$C \; = \; \{ \, c_{ij} \; \in \{0, \; 1\} \; \mid \; i \; = \; 1, \ldots, m, \; j \; = \; 1, \ldots, n \, \} \tag{2-6}$$

$$W \; = \; \{ \, w_{ij} \; \in \mathbb{Z}^{+} \; \mid \; i \; = \; 1, \ldots, m, \; j \; = \; 1, \ldots, n \, \} \tag{2-7}$$

$$B \;=\; \{\, b_i \;\in \mathbb{Z}^+ \;\mid\; i \;=\; 1, \ldots, m\} \tag{2-8}$$

O objetivo do PAG abordado neste trabalho será a minimização de custos. Logo, a formulação adotada é um problema convexo de otimização combinatória com restrições convexas e que impõe a obrigatoriedade de que todas as tarefas sejam executadas por alguma máquina.

Essas considerações conduzem naturalmente à formulação padrão inteira descrita abaixo em (2-9).

$$\begin{aligned}
Z_{IP} \;=\; & \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\
& s.\,t. \\
& \sum_{i=1}^{m} x_{ij} = 1 && \forall\; j = 1, \ldots, n \\
& \sum_{j=1}^{n} w_{ij} x_{ij} \;\leq\; b_i && \forall\; i = 1, \ldots, m \\
& x_{ij} \in \{0,\, 1\} && \forall\; i = 1, \ldots, m\,;\; j = 1, \ldots, n
\end{aligned} \tag{2-9}$$

2.3
Relaxação Linear da Formulação Padrão

A formulação padrão inteira é computacionalmente intratável para algumas instâncias do PAG devido à natureza NP-Difícil da generalização de problemas de mochila que caracteriza o PAG. Essa intratabilidade significa que há instâncias do PAG que são difíceis de resolver por programação inteira.

Em contrapartida, um LB pode ser encontrado, com complexidade polinomial, para uma dada instância do PAG através da programação linear. Então, a relaxação linear (LP) da formulação clássica definida em (2-9) mostra-se relevante para a determinação do LB de instâncias do PAG e esse LB pode servir como uma solução inicial (linear), uma referência útil, para outros algoritmos e heurísticas.

Essa relaxação linear abandona a integralidade das variáveis x_{ij} e, assim, permite atribuições fracionárias por meio de variáveis $x_{ij} \in [0.0,\; 1.0]$ definidas para toda máquina $i = 1, \ldots, m$ e toda tarefa $j = 1, \ldots, n$.

Entretanto, a solução linear encontrada não corresponde necessariamente a uma solução inteira válida e, portanto, outros métodos precisam ser designados para buscar soluções inteiras potencialmente ótimas.

3
Algoritmos de Geração de Colunas para o PAG

3.1
Decomposição de Dantzig-Wolfe

O conjunto de possíveis combinações de tarefas atribuíveis a cada máquina define uma quantidade muito grande, exponencial, de soluções inteiras para instâncias do PAG. Não apenas há um número exponencial de soluções inteiras para o PAG, como também a alocação generalizada é uma generalização de problemas de mochilas que, portanto, configura um problema NP-Difícil, não sendo sempre possível inspecionar explicitamente todas as soluções inteiras viáveis em tempo polinomial. Essa dificuldade típica de problemas NP-Difíceis exige a adoção de técnicas mais sofisticadas para solucionar instâncias de forma eficaz e eficiente.

Em meio aos esforços e às contribuições encontrados na literatura, Dantzig e Wolfe (1961) propõem uma técnica de decomposição para programas lineares que possibilita adotar uma estratégia de geração de colunas (*pricing*) para resolver a relaxação linear de um modelo decomposto. Essa decomposição ocorre pela desagregação de restrições independentes que são as restrições de capacidade máxima de máquina no caso do PAG. Essa desagregação permite resolver um modelo clássico de PAG a partir da formulação de um modelo mestre sujeito ao respectivo conjunto de subproblemas com soluções inteiras nas regiões convexas definidas pelas restrições de capacidade de máquina desagregadas. Portanto, resolver uma instância do PAG torna-se mais fácil com essa separação da formulação clássica em formulações mais simples, nomeadamente a formulação mestre com atribuições de tarefas (colunas) sujeitas ao conjunto de subproblemas de mochila de cada máquina.

A motivação primária é evitar a exploração explícita de todas as possíveis soluções que existem para uma instância do problema. Então, um método eficiente sustentado pela decomposição de Dantzig-Wolfe é implementado para selecionar somente um subconjunto de atribuições de tarefas às máquinas (colunas), construindo um modelo mestre restrito. Esse modelo mestre restrito considera explicitamente uma base de colunas que constitui um subconjunto de soluções significativamente menor. Em outras palavras, uma solução linear ótima pode ser

determinada para uma instância PAG através da geração de colunas para um modelo de decomposição de Dantzig-Wolfe, resolvendo os subproblemas de mochila que precificam (*pricing*) padrões de alocação de tarefas às máquinas.

O algoritmo *price-and-cut* proposto depende intrinsecamente desse procedimento de desagregação das restrições de capacidade de máquina da formulação clássica (2-9), previamente descrita no Capítulo 2. Essa desagregação transforma as variáveis de decisão x_{ij} em blocos de decisão λ_i^k sujeitos às restrições desagregadas (problemas de mochila).

Esse processo discretiza as soluções de atribuição de tarefas a máquinas através do conceito de coluna k para expressar cada bloco de atribuição viável λ_i^k de n tarefas à máquina i. Essas colunas devem respeitar sempre as restrições desagregadas caracterizadas como regiões convexas de problemas de mochila clássicos. Consequentemente, as colunas viáveis para esse modelo de decomposição são somente aquelas que não infringem as restrições de capacidade das m máquinas.

Durante a geração de colunas, posteriormente abordada, a inclusão de qualquer coluna ao mestre restrito depende sempre do seu custo reduzido computado pela otimização do respectivo subproblema de mochila (*pricing*) definido pela restrição desagregada correspondente (máquina i). O subproblema de *pricing* para avaliar o custo reduzido de cada coluna é um problema de mochila para cada máquina, tendo como função objetivo a expressão do custo reduzido da alocação de um conjunto de tarefas T a uma determinada máquina i. Portanto, o conceito de coluna da decomposição obedece a formalização abaixo.

1. A definição da coluna representada vetorialmente pela variável $\lambda_i^k \in \{0,1\}^{|T|}$ é uma atribuição de T tarefas à máquina i.

2. A coluna λ_i^k está sujeita à restrição de capacidade $\sum_{j=1}^{n} w_{ij} x_{ij}^k \leq b_i$.

3. O *pricing* é a minimização do custo reduzido, computada pelo respectivo subproblema de mochila, que se dispõe a encontrar as melhores colunas para o mestre restrito.

Essa técnica de decomposição com geração de colunas eficientemente evita a análise explícita de todas as possíveis soluções viáveis da formulação clássica. Entretanto, a quantidade de colunas avaliadas na programação inteira dessa formulação de Dantzig-Wolfe também é muito grande, intratável.

O diferencial dessa desagregação é a disposição de regiões convexas bem definidas pelos subproblemas de mochila que certificam o procedimento de *pricing* para avaliação de colunas, permitindo que apenas colunas de melhor custo reduzido sejam consideradas na construção de um modelo mestre restrito.

O *pricing* consiste em analisar as soluções de problemas de mochila para cada máquina individualmente, selecionando colunas λ_i^k com custo reduzido mínimo negativo para compor o modelo mestre restrito da decomposição. Esse procedimento é indispensável para habilitar o método de geração de colunas, descrita posteriormente nesta dissertação.

Em consequência disso, compete ao algoritmo principal coordenar a geração de colunas com a decomposição de Dantzig-Wolfe para determinar subconjuntos de tarefas j atribuídos às máquinas i que encaminhem o modelo a convergir para uma solução ótima. O *pricing* para buscar esses blocos de atribuição por máquina é efetuado pelo algoritmo de geração de colunas.

Decorrente da teoria exposta, a formulação da decomposição de Dantzig-Wolfe (3-6) substitui o conjunto convexo de restrições (2-4) definidas para cada máquina por conjuntos de vértices que compõem esse mesmo poliedro, de acordo com o teorema 3.1 que demonstra que todo poliedro possui uma representação vetorial.

Teorema 3.1 *(Teorema de Minkowski-Weyl) Dado um subconjunto X definido em $\mathbb{R}^d$, as seguintes afirmações são equivalentes.*

1. *X é um poliedro dada uma matriz real finita A e dado um vetor real b, tal que:*

   $X = \{x \mid Ax \leq b\}$.

2. *Existe um conjunto finito de vetores P e raios extremos R que definem X como especificado abaixo.*

   $$X = \left\{ x \mid x = \sum_{p \in P} \lambda_p \cdot \bar{x}_p + \sum_{r \in R} \delta_r \cdot \bar{x}_r \right\}$$

O teorema de Minkowski-Weyl demonstra que todo poliedro possui uma representação vetorial, oferecendo fundamento para a decomposição de Dantzig-Wolfe.

Portanto, a decomposição de Dantzig-Wolfe (3-6) representa a região convexa de cada restrição de máquina (2-4) por um conjunto de pontos extremos. Esse conjunto de pontos define a região convexa descrita pela respectiva restrição desagregada (2-4), considerando colunas λ_i^k para combinações convexas desses pontos.

De acordo com a formulação clássica do PAG apresentada em (2-9), existe uma região convexa para cada máquina i. Deste modo, considere as regiões convexas definidas em (3-1) para as restrições de capacidade das máquinas i.

$$Q_i = \left\{ x_{ij} \in \{0,1\} \;\middle|\; \sum_{j=1}^{n} w_{ij} x_{ij} \leq b_i \right\} \quad \forall \quad i = 1, \ldots, m \tag{3-1}$$

Pelo teorema 3.1, assuma que uma coluna λ_i^k representa uma alocação de tarefas $j \in T$ à respectiva máquina i dentro da região convexa correspondente. O conjunto P_i de colunas λ_i^k definido em (3-3) descreve todas as possíveis combinações convexas de pontos descritos por d_{ij}^k, onde d_{ij}^k é uma atribuição de tarefa j para a máquina i na coluna k.

O poliedro Q_i passa a ser caracterizado pelos respectivos vetores combinados pelas colunas P_i referentes à máquina i. Decorre disso que, pelo teorema 3.1, a região convexa de Q_i pode ser escrita como uma combinação linear de pontos descritos por d_{ij}^k definida para cada coluna λ_i^k, conforme Equação 3-2.

$$Q_i = \left\{ x_{ij} \in \{0,1\} \;\middle|\; x_{ij} = \sum_{k=1}^{|P_i|} d_{ij}^k \lambda_i^k \right\} \quad \forall \quad i = 1, \ldots, m \tag{3-2}$$

$$P_i = \left\{ \lambda_i^1, \lambda_i^2, \ldots, \lambda_i^k \right\} \quad \forall \quad i = 1, \ldots, m \tag{3-3}$$

$$d_{ij}^k = \begin{cases} 1, & \text{se atribuição } (j,\, i) \text{ ocorre na coluna } k \\ 0, & \text{caso contrário} \end{cases}$$
$$\forall \quad k = 1, \ldots, |P_i|,\ i = 1, \ldots, m,\ j = 1, \ldots, n \tag{3-4}$$

Os coeficientes d_{ij}^k, definidos em (3-4), são multiplicadores da combinação linear definida pela coluna λ_i^k, igualmente advindos da desagregação, logo determinados pelo subproblema de mochila da região convexa correspondente. Os subproblemas associados às restrições independentes do PAG e necessários ao *pricing* de colunas são abordados na Seção 3.3.

Em decorrência dessa decomposição, o conjunto de variáveis de decisão x_{ij} relacionadas à mesma máquina i são reescritas em função de $d_i^k \cdot \lambda_i^k$, como mostrado abaixo em (3-5). A variável de decisão é escrita em função de colunas tal que $x_{ij} = \sum_{k \in P_i} d_{ij}^k \lambda_i^k$ onde d_{ij}^k é o coeficiente de λ_{ij}^k.

$$x_{ij} = \sum_{k=1}^{|P_i|} \left(d_{ij}^k \, \lambda_i^k \right)$$
$$s.\,t.$$
$$\sum_{k=1}^{|P_i|} \lambda_i^k = 1$$

$$\lambda_i^k \in \{0,\ 1\} \qquad \forall \quad k = 1, \ldots, |P_i| \tag{3-5}$$

As restrições de atribuição de tarefa j (2-3) também são reescritas em função dos termos $d_{ij}^k\ \lambda_i^k$, trocando as variáveis x_{ij} pelas colunas da decomposição.

Essa desagregação de Dantzig-Wolfe resulta na discretização das m restrições de máquina (2-4) pelas representações vetoriais das respectivas regiões convexas, levando à formulação do mestre irrestrito em (3-6), com a substituição designada em (3-5) de variáveis x_{ij} pelas respectivas colunas.

Nesta formulação de decomposição, as colunas são as variáveis de decisão. Qualquer coluna que exceda a capacidade máxima da respectiva máquina é imperativamente rejeitada, segundo a definição apresentada em (3-1).

$$
\begin{aligned}
Z_{DW_{IP}} \ = \ & \min \sum_{i=1}^{m} \sum_{k=1}^{|P_i|} \sum_{j=1}^{n} \left(c_{ij}\ d_{ij}^k\ \lambda_i^k \right) \\
& s.\,t. \\
& \sum_{i=1}^{m} \sum_{k=1}^{|P_i|} \left(d_{ij}^k\ \lambda_i^k \right) = 1 \qquad \forall \quad j = 1, \ldots, n \\
& \sum_{k=1}^{|P_i|} \lambda_i^k \ = 1 \qquad \forall \quad i = 1, \ldots, m \\
& \lambda_i^k \in \{0,\ 1\} \qquad \forall \quad k = 1,\ ...,\ |P_i|;\ i = 1,\ ...,\ m
\end{aligned}
\tag{3-6}
$$

O algoritmo de *price-and-cut* proposto adota a geração de colunas para resolver a relaxação linear da formulação de Dantzig-Wolfe (3-6), removendo a integralidade das colunas λ_i^k de modo que $0 \leq \lambda_i^k \leq 1\ \forall\ k = 1,\ \ldots,\ |P_i|;\ i = 1,\ \ldots,\ m$.

Essa reformulação relaxada com desagregação e representação de colunas viabiliza a geração de colunas para localizar colunas iterativamente, compondo o modelo com um número reduzido de colunas, suficiente para resolver esse modelo de relaxação linear até a otimalidade.

Esse método não precisa lidar com todas as colunas existentes, apenas aquelas que possuem um melhor custo reduzido pelo *pricing* do subproblema de mochila, definido para cada conjunto convexo (3-1). Os coeficientes d_{ij}^k também são determinados pelos subproblemas, assumindo valores binários $\{0, 1\}$ que indicam a inclusão da alocação da tarefa j à máquina i na solução do subproblema que gera a coluna λ_i^k.

Embora a solução ótima dessa relaxação linear não seja necessariamente uma solução ótima inteira, já indica um LB útil para reduzir o espaço de busca da enumeração de colunas tratada no Capítulo 5. Dado que o algoritmo *price-and-cut* precede a enumeração de colunas otimizada pretendida, essa abordagem é uma parte essencial para que se obtenha uma base de colunas capaz de caracterizar um LB maior (redução do *gap*). Esse *gap* reduzido é o resultado que permite delimitar e reduzir o espaço da enumeração de colunas subsequente. Essa é a finalidade do *price-and-cut* proposto nesta dissertação.

3.2 Definição do Mestre Restrito

A formulação da decomposição Dantzig-Wolfe $Z_{DW_{IP}}$ assume o conjunto de todas as colunas viáveis, portanto os pontos das regiões convexas de cada restrição de máquina. Entretanto, os poliedros inteiros de cada restrição desagregada contêm uma quantidade proibitiva de colunas, tornando necessário restringir as colunas usadas para resolver a programação inteira dessa formulação. Essa restrição de colunas do modelo de decomposição consiste em selecionar somente um subconjunto de colunas em cada poliedro para adicionar à programação inteira do modelo.

Então, o mestre restrito é conceitualmente o modelo mestre da formulação de decomposição sujeito a um subconjunto de colunas, assim considerando apenas uma reduzida base de colunas selecionadas que viabilizam uma solução viável e, idealmente, ótima. Quando a integralidade do modelo é abandonada, a geração de colunas é um método ideal para localizar as colunas que devem ser adicionadas ao mestre restrito tendo em vista alcançar a solução linear ótima. Existindo uma base de colunas suficiente formada pela geração de colunas, o modelo mestre restrito pode ser construído e resolvido considerando apenas as colunas dessa base.

É significativamente difícil formar uma boa base de colunas inteiras sem algoritmos heurísticos e outros métodos sofisticados. É muito mais fácil resolver a relaxação linear do mestre restrito, acrescentando colunas iterativamente pela geração de colunas.

Em virtude disso, embora sejam conceitualmente independentes, a decomposição de Dantzig-Wolfe aparece quase sempre associada à geração de colunas. Nesta situação, sendo a geração de colunas desejada, o mestre restrito deve ter a formulação de decomposição relaxada, abandonando a integralidade das colunas λ_i^k para que colunas possam ser geradas através do *pricing* (custo reduzido). O detalhamento desse algoritmo está disponível a partir da Seção 3.5.

O conceito de modelo mestre restrito implica que restringimos o conjunto de colunas existentes no modelo de decomposição, definido em (3-6), para torná-lo

computacionalmente tratável. Perante o exposto, conhecida a formulação do modelo mestre da decomposição de Dantzig-Wolfe, o algoritmo de *price-and-cut* deste trabalho realiza a geração de colunas sobre a relaxação linear desta formulação (3-6), removendo a integralidade das colunas.

Em condições favoráveis, a quantidade de colunas pode ser significativamente reduzida no mestre restrito, permitindo resolver a programação inteira do modelo mestre da decomposição apenas com uma base de colunas encontrada para a sua relaxação linear pela geração de colunas. Em muitos casos, não haverá uma solução viável, nem ótima, para o modelo de Dantzig-Wolfe apenas com a base de colunas encontrada para a relaxação linear.

Entretanto, essa base de colunas pode ser incrementada, complementada, através da enumeração de colunas, considerando explicitamente as colunas que tenham custo reduzido $\bar{r}_k \leq UB - LB$, onde $UB - LB$ é o mínimo *gap* entre o UB e o LB conhecidos. Portanto, a meta do algoritmo *price-and-cut* é encontrar o maior LB possível na geração de colunas para que o *gap* seja minimizado, diminuindo o número de colunas no escopo da enumeração posterior.

3.3 Pricing da Decomposição Dantzig-Wolfe

O *pricing* de colunas é um componente fundamental da geração de colunas e o tempo para computar o custo reduzido de colunas em cada máquina é importante para garantir um algoritmo eficiente capaz de resolver a formulação relaxada de Dantzig-Wolfe para o PAG. O subproblema da geração de colunas do PAG é precisamente o problema de mochila, já abordado e contextualizado em Seção 3.1.

Suponha um conjunto de n itens com pesos $W_i = \{q_j \in \mathbb{Z}^+ \; \forall \; j = 1, \ldots, n\}$ e lucros $G_i = \{p_j \in \mathbb{Z}^+ \; \forall \; j = 1, \ldots, n\}$ para serem colocados dentro de uma mochila i, respeitando o peso máximo b_i permitido para essa mochila. Assuma também as variáveis de decisão $y_j \in \{0, 1\} \; \forall \; j = 1, \ldots, n$, onde $y_j = 1$ se e somente se o item j é atribuído à mochila i, senão $y_j = 0$.

$$
\begin{aligned}
Z_i = \max & \sum_{j=1}^{n} p_j y_j \\
s.\,t. & \\
& \sum_{j=1}^{n} q_j y_j \leq b_i \\
& y_j \in \{0,\ 1\} \qquad\qquad \forall \quad j = 1, \ldots,\ n
\end{aligned}
\tag{3-7}
$$

O objetivo da inserção de itens na mochila é a maximização do lucro total carregado dentro da mochila, considerando o valor p_j de cada item j carregado pela mochila i. O problema de mochila é um clássico problema combinatório que possui a formulação (3-7), assumindo as variáveis de decisão y_j binárias (0 ou 1).

1. O lucro de introdução do item j na mochila i é representado por p_j, conhecido o conjunto de lucros $G_i = \{p_j \in \mathbb{Z}^+ \; \forall \; j = 1, \ldots, n\}$ para a mochila i.

2. O peso do item j dentro da mochila i é conhecido como q_j, definido pelo conjunto de pesos $W_i = \{q_j \in \mathbb{Z}^+ \; \forall \; j = 1, \ldots, n\}$ para a mochila i.

3. A inserção do item j na mochila i é uma decisão do problema de otimização que se traduz na variável $y_j \in \{0, 1\}$ tal que $y_j = 1$ se e somente se o item j está posto dentro da mochila i; e caso contrário $y_j = 0$.

4. Toda inserção de item j na mochila i incorre um acréscimo de lucro p_j na função objetivo e também consome uma cota de peso q_j do peso máximo b_i suportado pela mochila i.

Existe método exato para solucionar mochilas 0-1, a própria programação inteira da formulação clássica. Entretanto, há outros métodos com melhor complexidade que a programação inteira, como implementações eficientes de programação dinâmica, já considerando instâncias mais difíceis e maiores. A redução do tempo de otimização para encontrar a solução ótima para problemas de mochila é extremamente desejável para melhorar a eficiência da geração de colunas e da enumeração de colunas.

Em respeito ao contexto de *pricing* da geração de colunas para o PAG, a mochila i representa uma máquina i e os itens j são tarefas j. Logo, ao invés de selecionar um subconjunto de itens, a formulação de mochila do *pricing* determina um subconjunto de tarefas e a mochila é substituída pela representação de uma máquina. Resumidamente, assuma as seguintes adaptações da mochila para atender ao *pricing* de colunas do PAG, conforme listado.

1. A função objetivo da mochila passa a ser uma minimização do custo reduzido de colunas definidas pelo subproblema associado à máquina i do mestre restrito da decomposição de Dantzig-Wolfe aplicada ao PAG.

2. A mudança semântica implica que os itens da mochila são as n tarefas do PAG e a mochila é representada por uma máquina i do PAG.

O subproblema de *pricing* associado à máquina i é um problema de mochila que determina um padrão de atribuição de tarefas à respectiva máquina i. O padrão correspondente à solução da mochila define uma coluna de custo reduzido

mínimo, coluna candidata para ser adicionada ao modelo restrito da formulação de decomposição. O *pricing* pode ser resolvido tanto pela programação inteira quanto pela programação dinâmica. O processo de geração de colunas está detalhado na Seção 3.5.

A avaliação de colunas para compor o modelo mestre restrito é pautada pelos subproblemas de *pricing* decorrentes da decomposição da formulação clássica. Essas colunas λ_i^k localizadas em cada região convexa (3-1) estão sujeitas às respectivas restrições de máquina i (2-4) desagregadas. A viabilidade de qualquer coluna do modelo deve, consequentemente, atender à restrição de capacidade da máquina correspondente.

Deste modo, o conjunto de colunas P_i admite somente as atribuições de tarefas j à máquina i que respeitam a capacidade máxima b_i da máquina i. Então, uma coluna viável de qualquer região convexa Q_i obrigatoriamente exige alocações de tarefas j à respectiva máquina i que mantenham a coluna λ_i^k dentro do invólucro convexo de Q_i.

O custo reduzido $\bar{r}_k$ da coluna k, definido em (3-8), serve de medida para avaliar a inclusão da coluna no mestre restrito. Portanto, a função objetivo de cada subproblema de *pricing* é formulada para encontrar a alocação de tarefas j à máquina i que minimiza o custo reduzido da respectiva coluna na sua região convexa (3-1), de maneira que a alocação não ultrapasse a capacidade limite da máquina. Logo, o *pricing* é equivalente a um problema de mochila. Logo, os lucros p_j de cada tarefa j na formulação de mochila são escritos em função do custo reduzido na formulação do subproblema (*pricing*).

A função objetivo de cada subproblema de mochila é uma expressão de cálculo do custo reduzido em cada região convexa das restrições de capacidade de máquina. Dado que o mestre restrito privilegia sobretudo as colunas cuja contribuição é capaz de fazer o valor corrente da sua função objetivo convergir para o ótimo, os subproblemas de *pricing* são problemas de formulação inteira com objetivo de minimização.

De acordo com a formulação de Dantzig-Wolfe para o PAG, o custo reduzido é calculado com base na dualidade das restrições do problema mestre, tanto os valores duais μ_j das restrições associadas às tarefas j (2-3) quanto as duais π_i das restrições ligadas às máquinas i (2-4). Essas duais contribuem para o custo reduzido $\bar{r}_k$ definido na Equação 3-8, onde N_k é o conjunto de tarefas j alocadas à máquina i na coluna k. Logo, a coluna de custo reduzido mínimo da máquina i é encontrada através da otimização do subproblema de *pricing* com função objetivo definida pela Equação 3-9, onde y_j é uma variável de decisão que aloca a tarefa j à máquina i no subproblma de mochila.

$$\bar{r}_k = \sum_{j \in N_k} \left(c_{ij} - \mu_j \right) - \pi_i \tag{3-8}$$

$$Z^i_{pricing} = \min \left(\sum_{j=1}^{n} \left(c_{ij} - \mu_j \right) y_j \right) - \pi_i \tag{3-9}$$

A estrutura do subproblema revela-se semelhante à estrutura do clássico problema de mochila (*knapsack*). Em contrapartida, a formulação do problema de mochila é uma maximização e um subproblema do *pricing* é uma minimização do custo reduzido da atribuição de tarefas j à máquina i que compõe uma coluna de custo reduzido mínimo. Logo, a alteração do objetivo da mochila supõe a transformação da minimização em uma maximização, invertendo os sinais para maximizar o negativo do custo reduzido. Assim sendo, a função objetivo (3-9) é reescrita como uma maximização na Equação 3-10 cuja solução reflete o custo reduzido mínimo.

$$Z^i_{pricing} = \max \left(\sum_{j=1}^{n} \left(\mu_j - c_{ij} \right) y_j \right) + \pi_i \tag{3-10}$$

Portanto, dado que cada subproblema de *pricing* depende apenas de uma restrição de máquina e a máquina i é fixa como representação da mochila, o valor da variável dual π_i é fixo durante a otimização da mochila i. Decorre dessa observação que a minimização descrita em (3-9) não depende da variável dual π_i, podendo ser reescrita ainda mais simplificadamente como definido em (3-11).

$$Z^i_{knapsack} = \max \left(\sum_{j=1}^{n} \left(\mu_j - c_{ij} \right) y_j \right) \tag{3-11}$$

A partir da definição da função objetivo para o *pricing* de colunas e das restrições de capacidade (2-4), a formulação do subproblema de mochila encontra uma definição plena como indicado em (3-12).

$$\begin{aligned} Z^i_{knapsack} = \max & \left(\sum_{j=1}^{n} \left(\mu_j - c_{ij} \right) y_j \right) \\ s.\,t. & \\ & \sum_{j=1}^{n} w_{ij}\, y_j \leq b_i \\ & y_j \in \{0,\, 1\} \qquad\qquad \forall \quad j = 1, \ldots,\, n \end{aligned} \tag{3-12}$$

Deste modo, a geração de colunas consegue buscar as colunas de menor custo reduzido, respeitando a capacidade máxima b_i de cada máquina i.

Quando uma solução é encontrada para a mochila i, a partir da formulação (3-10), o custo reduzido da coluna candidata relacionada a essa solução de mochila é calculada subtraindo o valor da função objetivo do valor negativo da variável dual π_i, assim como descrito na equação (3-13). É preciso inverter o sinal da solução $Z^i_{knapsack}$ uma vez que houve inversão do sinal do custo reduzido na função objetivo de maximização do subproblema.

$$\bar{r}_k = -Z^i_{knapsack} \; - \; \pi_i \tag{3-13}$$

Em suma, o conjunto das colunas de menor custo reduzido associado à máquina i é encontrado através do *pricing* determinado pela formulação de mochila descrita em (3-14).

$$\begin{aligned} Z^i_{pricing} = \max & \left(\sum_{j=1}^{n} (\mu_j - c_{ij}) y_j \right) \\ s.\,t. & \\ & \sum_{j=1}^{n} w_{ij} y_j \; \leq b_i \\ & y_j \in \{0,\, 1\} \qquad\qquad \forall \quad j = 1, \ldots,\, n \end{aligned} \tag{3-14}$$

No contexto da geração de colunas, uma solução da mochila gera uma coluna k com variável de decisão λ^k_i. Os respectivos multiplicadores d^k_{ij} são os coeficientes combinados pela variável de decisão λ^k_i da coluna k do problema mestre. A variável λ^k_i representa a atribuição de tarefas j à máquina i, tarefas selecionadas pela solução ótima da mochila i resolvida pelo *pricing*. Logo, essa atribuição de tarefas j à máquina i pela coluna k é estabelecida pela respectiva solução do subproblema de mochila do *pricing*.

1. O coeficiente d^k_{ij} considerado pela coluna λ^k_i possui valor 1 se e somente se a variável y_j da solução ótima da mochila i for igual a 1, senão d^k_{ij} assume valor 0;

2. A variável λ^k_i da coluna k avaliada pelo *pricing* representa a atribuição, à máquina i do mestre restrito, das tarefas selecionadas pela solução ótima da mochila i.

Em qualquer iteração da geração de colunas para o modelo restrito, a solução de cada subproblema de mochila indica um conjunto de tarefas alocadas à respectiva

máquina com custo reduzido mínimo. Então, uma solução da mochila i corresponde a uma potencial nova coluna λ_i^k a ser explicitamente considerada pelo mestre restrito.

Embora os subproblemas de mochila tenham sido resolvidos com programação inteira, a programação dinâmica clássica para problemas de mochila também é uma alternativa para executar esse *pricing*. Nesta dissertação, a programação inteira foi inicialmente adotada porque o a programação dinâmica clássica para mochila não atendia à computação de custos reduzidos com desigualdades (3, 0.5)-SRC.

Entretanto, o *pricing* da geração de colunas pode ser resolvido com a programação dinâmica clássica para mochila quando não há separação de desigualdades (3, 0.5)-SRCs. O algoritmo dessa programação dinâmica recorre à função objetivo da formulação da mochila já definida na Equação 3-11, minimizando o custo reduzido das colunas. Neste caso, a função objetivo da mochila i depende somente dos custos c_{ij} de atribuição de tarefa j à máquina i e das duais μ_j das restrições de atribuição de tarefa j, sendo que esses custos e duais possuem valor constante para cada atribuição de tarefa j à máquina i.

Esse algoritmo de programação dinâmica padrão percorre todas as n tarefas j segundo uma ordenação fixa J, construindo uma árvore de *labels* limitada pela restrição de capacidade máxima da maquina i. Então um nível da árvore é construído para cada possível atribuição de cada tarefa j aos *labels* existentes. O *label* é uma possível atribuição de tarefas j já visitadas pela programação dinâmica à máquina i, armazenando o custo total e o peso ocupado da respectiva atribuição. Esse custo total reflete um possível valor de função objetivo do problema de mochila, portanto uma solução viável.

Em cada nível t da árvore, a inserção da tarefa j deste nível é avaliada individualmente para cada *label* k do nível anterior $t-1$, segundo os critérios discriminados abaixo.

1. Existem duas possibilidades de ramificação do *label* k, podendo ocorrer a criação de um *label* $\bar{k}$ que herda tarefas de k tanto pela adição quanto pela não adição da tarefa $j = J[t]$ do nível t;

2. Considere que um *label* k representa uma possível solução viável com a atribuição de um subconjunto de tarefas $j \in N_k$ já visitadas pela programação dinâmica;

3. N_k é o conjunto de tarefas j alocadas à máquina i no *label* k;

4. OPT é um mapa que organiza *labels* k por tarefa j (nível t) e peso da *label* k;

5. Considere $l_k = \sum_{j \in N_k} w_{ij}$ como a capacidade ocupada pelas tarefas do *label* k;

6. Considere que o somatório dos custos implicados por todas as tarefas j atribuídas à máquina i pelo *label* k, onde p_j é o custo da atribuição da tarefa j na formulação clássica de mochila e $\sum_{j \in N_k} p_j$ representa o custo da *label* k;

***Script* 1** Programação Dinâmica Padrão para Mochila (PAG)

Data: conjunto de tarefas T; máquina i

Result: mapa OPT com as soluções da mochila para a máquina i e tarefas T

- Define uma ordenação J de tarefas, uma para cada nível da árvore de *labeling*;
- Declara o mapa OPT para armazenar *labels* para cada nível da árvore;
- Inicializa o *label* raíz no nível 0 da árvore;

for $t = 1$; $t \leq |J|$; $t = t + 1$ **do**
- Reconhece a tarefa corrente $J[t]$;
- **for** $y = 0$; $y \leq OPT[t-1]$; $y = y + 1$ **do**
  - Acessa o *label* k em $OPT[t-1][y]$;
  - **if** *label* x ***existe*** *no nível* $OPT[t]$ *com peso* $\sum_{j \in N_k} w_{ij}$ *da label* k **then**
    - **if** $\sum_{j \in N_k} p_j > \sum_{j \in N_x} p_j$ **then**
      - Cria *label* $\bar{k}$ no nível $OPT[t]$ com ($\sum_{j \in N_k} p_j$, $\sum_{j \in N_k} w_{ij}$);
      - Deleta o *label* x;
  - **else**
    - Cria *label* $\bar{k}$ no nível $OPT[t]$ com ($\sum_{j \in N_k} p_j$, $\sum_{j \in N_k} w_{ij}$);
  - **if** $\sum_{j \in N_k} w_{ij} + w_{iJ[t]} > b_i$ **then**
    - **continue**;
  - **if** *label* x ***existe*** *no nível* $OPT[t]$ *com peso* $\sum_{j \in N_k} w_{ij} + w_{iJ[t]}$ **then**
    - **if** $\sum_{j \in N_k} p_j + p_{J[t]} > \sum_{j \in N_x} p_j$ **then**
      - Cria *label* $\bar{k}$ em $OPT[t]$ com ($\sum_{j \in N_k} p_j + p_{J[t]}$, $\sum_{j \in N_k} w_{ij} + w_{iJ[t]}$);
      - Deleta o *label* x;
  - **else**
    - Cria *label* $\bar{k}$ em $OPT[t]$ com ($\sum_{j \in N_k} p_j + p_{J[t]}$, $\sum_{j \in N_k} w_{ij} + w_{iJ[t]}$);

- **Retorna** OPT;

7. Não ocorre a inserção da tarefa j no filho $\bar{k}$ do *label* k caso a tarefa j tenha peso w_{ij} tal que $l_k + w_{ij} > b_i$, onde b_i é a capacidade máxima da máquina i;

8. É permitida a inserção da tarefa j no *label* filho $\bar{k}$ se e somente se essa inserção não exceder a capacidade máxima b_i da máquina e ainda não existir um *label* no nível t ocupando a mesma capacidade $l_k + w_{ij}$ da mochila com um valor de solução igual ou maior que $\sum_{g \in N_k} p_g + p_j$;

9. Caso a tarefa j seja inserida na mochila, o novo *label* $\bar{k}$ terá solução com valor $\sum_{g \in N_{\bar{k}}} p_g = \sum_{g \in N_k} p_g + p_j$ e peso $l_{\bar{k}} = l_k + w_{ij}$;

10. Caso a tarefa j não seja inserida na mochila, o novo *label* $\bar{k}$ terá solução com valor $\sum_{g \in N_k} p_g$ e peso l_k;

Desta forma, o algoritmo 1 constrói uma árvore binária de *labels* com uma solução ótima para cada valor de capacidade menor que b_i em cada nível t até visitar a última tarefa ou até que mais nenhuma tarefa caiba nas *labels* do último nível gerado $t - 1$.

Embora funcione bem para mochilas cuja função objetivo depende apenas de valores constantes c_{ij} e μ_{ij}, esse algoritmo desconhece a regra de ativação das duais das (3, 0.5)-SRCs e, caso não haja uma adaptação adequada, falha no *pricing* do método com separação de desigualdades (3, 0.5)-SRC proposto nesta dissertação. Essa adaptação da programação dinâmica clássica de mochila é um assunto tratado no Capítulo 4.

3.4 Decomposição Dantzig-Wolfe Ponderada de Wentges

Em condições específicas onde a formulação do problema dispõe de um conjunto de restrições independentes, há muitos benefícios na aplicação da decomposição de Dantzig-Wolfe para problemas MIP. A estratégia consiste em simplificar a formulação clássica, explorando a estrutura independente de restrições dificultadoras para que elas possam ser removidas e atendidas separadamente. Em decorrência da remoção dessas restrições, a formulação do MIP passa a estar sujeita a menos restrições, logo mais fácil de ser resolvida.

Entretanto, o *pricing* de colunas para o mestre restrito, durante a geração de colunas, não prevê as oscilações dos valores das variáveis duais $\pi_i \ \forall \ i = 1, \ \ldots, \ m$ e $\mu_j \ \forall \ j = 1, \ \ldots, \ n$ das restrições do modelo mestre, assim desconsiderando a degenerescência de instâncias que retarda significativamente a convergência da geração de colunas. Logo, técnicas de estabilização são usadas para corrigir essa indesejável degenerescência, diminuindo a magnitude dessas

oscilações e, consequentemente, melhorando a eficiência da geração de colunas. Quando existe uma oscilação que retarda a convergência da geração de colunas, convém haver um ajuste dos valores duais das restrições para mantê-los próximos do valor corresponde à melhor solução conhecida para o modelo. A decomposição de Dantzig-Wolfe ponderada propõe uma estratégia para lidar com isso a partir do *bound* determinado pela relaxação Lagrangeana.

A relaxação Lagrangeana é uma alternativa para resolver a relaxação linear de problemas MIP, sobretudo para computar eficientemente bons *bounds* para o problema. Explorando a estrutura de dualidade das restrições que caracterizam a alta dificuldade do MIP, essa técnica permite expressar essas restrições dentro da função objetivo do MIP através das suas duais. Essa transformação que relaxa as restrições conduz à formulação da relaxação Lagrangeana do MIP.

$$
\begin{aligned}
Z_{MIP} = \min & c\,x \;+\; d\,y \\
& s.\,t. \\
& \quad A\,x \;+\; B\,y \leq q_1 \\
& \quad C\,x \;+\; D\,y \leq q_2 \\
& \quad x \geq 0,\; y \;\in\; P
\end{aligned}
\tag{3-15}
$$

Considere a formulação generalizada de MIP acima em (3-15). A ideia da relaxação Lagrangeana é escrever as restrições $A\ x + B\ y\ \leq\ q_1$, difíceis na formulação, direto na função objetivo em função das suas duais u, conforme descrito abaixo em (3-16).

Logo, a reformulação pela relaxação Lagrangeana ainda considera essas restrições através da dualidade e não mais na forma de desigualdades.

$$
\begin{aligned}
Z_{LR} = \min & \; c\,x \;+\; d\,y \;+\; (q_1 \;-\; A\,x \;-\; B\,y)\,u \\
& s.\,t. \\
& \quad C\,x \;+\; D\,y \leq q_2 \\
& \quad x \geq 0,\; y \;\in\; P
\end{aligned}
\tag{3-16}
$$

Caso os valores das duais u sejam fixados em dado instante, a formulação da relaxação Lagrangeana passa a ser denominada como subproblema Lagrangeano. A partir da fixação das duais u da relaxação Lagrangeana, a solução desse

subproblema Lagrangeano serve heuristicamente para determinar um LB para o MIP de origem.

Neste contexto, diante da necessidade de buscar o melhor LB, uma opção válida para obter esse valor é otimizar a relaxação Lagrangeana, resolvendo a dual Lagrangeana (LD). Essa dual Lagrangeana corresponde à maximização da relaxação Lagrangeana em relação às duais u. Essa maximização se reduz trivialmente ao subproblema Lagrangeano em situações nas quais as duais u são fixas, algo que ocorre em toda iteração da geração de colunas.

Nesta dissertação, a estabilização da geração de colunas é inspirada nessa estratégia apresentada por Wentges (1997), que recorre à dual Lagrangeana para preservar sempre o melhor LB encontrado ao longo da geração de colunas.

Esse procedimento diminui as oscilações de duais do mestre restrito do PAG, sem recorrer às penalizações da estabilização implementada em Pigatti *et al.* (2005).

Esse método de estabilização por decomposição ponderada utiliza apenas dois parâmetros, mais fáceis de ajustar em comparação às variáveis de penalização. Esses parâmetros são o coeficiente $0 < \alpha \leq 1$ para ponderação de duais e o coeficiente de tolerância $\gamma > 0$ usado como condição de parada da estabilização.

No contexto particular da formulação de Dantzig-Wolfe do PAG, a relaxação Lagrangeana dualiza as restrições de atribuição de tarefas, transformando a formulação clássica (2-9) em um problema de mochila, como desenvolvido em (3-17), (3-18) e (3-19), onde μ_j é a variável dual da restrição de tarefa j do modelo de decomposição.

$$Z_{RL} = \min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n}\left(1 - \sum_{i=1}^{m} x_{ij}\right)\mu_j \tag{3-17}$$

$$Z_{RL} = \min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n}\mu_j - \sum_{i=1}^{m}\sum_{j=1}^{n}\mu_j x_{ij} \tag{3-18}$$

$$Z_{RL} = \min \sum_{i=1}^{m}\sum_{j=1}^{n}\left(c_{ij} - \mu_j\right)x_{ij} + \sum_{j=1}^{n}\mu_j \tag{3-19}$$

Desta maneira, o subproblema Lagrangeano para o PAG adquire a forma de um problema de mochila, decorrente da adição da dualidade das restrições de atribuição do modelo padrão à função objetivo.

Dada a fixação das duais μ_j referentes às restrições de atribuição, a dual Lagrangeana do PAG assume a formulação (3-20) cuja resolução define o melhor LB para o mestre restrito do PAG em relação às duais $\mu_j \ \forall \ j = 1, \ldots, n$ consideradas pela relaxação Lagrangeana.

$$
\begin{aligned}
Z_{RL} = \min \; & \sum_{i=1}^{m}\sum_{j=1}^{n} \big(c_{ij} \; - \; \mu_j\big)x_{ij} \; + \; \sum_{j=1}^{n}\mu_j \\
& s.\,t. \\
& \sum_{j=1}^{n} w_{ij}x_{ij} \; \leq \; b_i \qquad\qquad \forall \; i = 1, \ldots, m \\
& x_{ij} \in \{0,\; 1\} \qquad\qquad \forall \; i = 1, \ldots, m \; ; \; j = 1, \ldots, n
\end{aligned}
\tag{3-20}
$$

Esse método de estabilização proposto por Wentges (1997) para a geração de colunas se beneficia exatamente dessa relaxação Lagrangeana de problemas MIP. A formulação do PAG é classicamente um problema IP, tendo apenas variáveis de decisão inteiras e a relaxação Lagrangeana pode ser aplicada da mesma forma, seguindo o mesmo conceito exposto.

Essa estabilização visa amortecer as oscilações das duais para estabilizar os custos reduzidos encontrados pelo *pricing* de colunas. Portanto, a relaxação Lagrangeana do PAG deve servir para preservar o melhor LB conhecido durante a geração de colunas do mestre restrito.

Em posse do melhor LB conhecido em cada iteração do *pricing*, as duais correntes π_i^{RM} e μ_j^{RM} do mestre restrito para tarefas j e máquinas i podem ser ponderadas através de uma combinação linear com as respectivas duais π_i^{LB} e μ_j^{LB} da melhor solução LB conhecida, como escrito abaixo em (3-21) e (3-22), onde π_i^{ST} e μ_j^{ST} são as respectivas duais estabilizadas.

$$\pi_i^{ST} \; = \; \alpha \, \pi_i^{RM} \; + \; (1 \; - \; \alpha) \, \pi_i^{LB} \quad \forall \quad i = 1, \; \ldots, \; m \tag{3-21}$$

$$\mu_j^{ST} \; = \; \alpha \, \mu_j^{RM} \; + \; (1 \; - \; \alpha) \, \mu_j^{LB} \quad \forall \quad j = 1, \; \ldots, \; n \tag{3-22}$$

Essa ponderação estabelece duais controladas para as restrições de atribuição (índice j) e as restrições de máquina (índice i), respectivamente μ_j^{ST} e π_i^{ST}, para que oscilação de duais possam ser evitadas no *pricing* da geração de coluna do PAG. Assuma que μ_j^{RM} e π_i^{RM} são as duais de restrições de atribuição e de restrições de máquina do mestre restrito, respectivamente. Desta maneira, a oscilação das duais do mestre restrito nos subproblemas de mochila são controladas, sendo mantidas próximas das duais de referência μ_j^{LB} e π_i^{LB} correspondentes ao melhor LB corrente.

Conforme o mestre restrito é construído, com a inserção de colunas, um melhor LB pode ser encontrado, exigindo que tanto o LB quanto as respectivas duais sejam atualizados. Essa atualização é avaliada através da relaxação Lagrangeana,

computando novamente a dual Lagrangeana para as duais estabilizadas que são candidatas a substituir as duais de referência na eventual confirmação de um novo LB.

3.5 Algoritmo de Geração de Colunas Clássico

De acordo com a decomposição de Dantzig-Wolfe aplicada ao PAG, o mestre restrito é conceitualmente inviável quando estão presentes todas as colunas λ_i^k possíveis no modelo, totalizando um número exponencial de variáveis de decisão.

Em contrapartida, não há garantia de solução ótima para o mestre restrito inicializado com colunas artificiais que apenas garantem a viabilidade do modelo. Então, ainda há a necessidade imperativa de adoção de um método que localize uma base mínima contendo as melhores colunas a compor o mestre restrito para que a solução ótima do modelo de decomposição seja acessível. A geração de colunas é um método iterativo que serve exatamente a esse propósito, construindo uma base de colunas λ_i^k com custo reduzido computado pelo *pricing*.

O método de geração de colunas (algorithm 2) é um algoritmo que iterativamente avalia o *pricing* de colunas para cada máquina i do problema. Em geral, a geração de colunas é utilizada para construir o mestre restrito com bases cada vez melhores, mantendo o modelo viável e buscando uma solução ótima através da dualidade do mestre e o *pricing* de novas colunas obtido pelos subproblemas de mochila.

O respeito pelas restrições de capacidade de máquina impostas pelo *pricing* é condição crítica para a viabilidade de qualquer coluna. Senão, uma coluna fora da região convexa Q_i da restrição desagregada seria elegível, sendo uma coluna λ_i^k com atribuição $\{i,\ T\}$ de T tarefas à máquina i totalizando um peso maior que b_i. Entre as colunas λ_i^k que atendem às limitações de capacidade de máquina, as colunas de menor custo reduzido são aquelas que melhor contribuem para a solução ótima da função objetivo do mestre restrito.

Em termos simplificados, cada coluna é composta por um conjunto de tarefas T atribuídas a uma máquina i, respeitando os limites de capacidade. Essa limitação do PAG significa que as colunas λ_i^k são soluções dos subproblemas de mochila que minimizam o custo reduzido dessa atribuição de tarefas.

Recorrendo à otimização dos subproblemas do *pricing* do PAG, a geração de colunas consegue encontrar colunas de custo reduzido $-Z^i_{knapsack} - \pi_i$ mínimo a cada iteração. Eventualmente, o custo reduzido dessas colunas precisa ser avaliado para determinar se a coluna deve ser adicionada ao mestre restrito. Quando a geração de colunas atinge a convergência, todas as colunas geradas pelo *pricing*

possuem custo reduzido não negativo e, portanto, nenhuma coluna mais precisa ser adicionada ao mestre restrito.

Embora esse método seja muito eficaz, ela apresenta pelo menos duas dificuldades, a primeira relacionada à lenta convergência e a segundo relacionada à integralidade da solução do mestre restrito. O *pricing* do custo reduzido existe para a relaxação linear da formulação de Dantzig-Wolfe. Então, ocorrida a convergência da geração de colunas, o mestre restrito terá atingido uma solução linear ótima para a relaxação linear do PAG e essa solução constitui um LB válido mas que não assegura uma solução inteira para o problema.

Assim sendo, essa técnica não garante soluções inteiras ótimas para instâncias do PAG. Não obstante, o mestre restrito relaxado sempre encontra um LB que pode ser utilizado para estabelecer um *gap* $UB - LB$, já tendo um UB conhecido como referência para a instância em questão.

Mesmo que a finalidade seja encontrar a solução inteira ótima para instâncias do PAG, o LB encontrado pela geração de colunas pode ser usado como instrumento para atingir esse objetivo posteriormente com a técnica de enumeração de colunas. Afinal, quanto maior o LB encontrado e quanto menor o UB de referência, menor o *gap*. Quanto menor o *gap*, menor o espaço de enumeração de colunas que completam o mestre restrito inteiro a ponto do modelo alcançar uma solução inteira ótima.

Esse conceito de redução do *gap* combinado com o auxílio de desigualdades do tipo 3-SRC para aumentar o LB é a direção investigada para o algoritmo de *price-and-cut* implementado e estudado nesta dissertação. Em vista disso, a geração de colunas é uma primeira parte fundamental para o entendimento do *pricing* que precede a inserção de cortes. A separação de desigualdades do tipo 3-SRC é outro assunto, tratado em outro capítulo exclusivamente dedicado a isso.

No momento, considere novamente a formulação de mochila para minimização do custo reduzido de colunas λ_i^k. Todas essas colunas estão dentro da região convexa de alguma restrição de capacidade de máquina. O termo $Z^i_{knapsack}$ indica a solução ótima da mochila para a máquina i, ignorando a dual π_i de restrições de máquina, como especificado anteriormente em (3-11). O termo $Z^i_{pricing}$ indica a solução ótima do *pricing* associado à máquina i, já especificado em (3-10). Logo, o custo reduzido de colunas encontradas pelas mochilas é calculado pela expressão (3-13).

Embora o custo reduzido mínimo seja a solução que configura a coluna que mais reduz a função objetivo do mestre restrito, todas outras soluções viáveis são aceitas para geração de coluna desde que o custo reduzido $\bar{r}_k$ seja negativo, ou seja, igual ou menor que um parâmetro $-\epsilon$ (cujo valor costuma ser bem próximo de zero, i.e. 0.0001).

Há muitas colunas viáveis que não melhoram a solução do mestre restrito ou que retardam a convergência dos custos reduzidos nas regiões convexas da decomposição. Se as melhores colunas forem selecionadas, a relaxação linear do mestre restrito alcançará a solução ótima com um conjunto mínimo de colunas mais rapidamente. Em caso de instâncias com muita degenerescência, técnicas de estabilização são indispensáveis para viabilizar a geração de colunas.

É conveniente poder escolher as colunas que podem ser gradativamente adicionadas ao mestre restrito para que seja construída uma base cada vez melhor para que a solução ótima do mestre melhore iterativamente da forma mais eficiente possível. No escopo deste trabalho, apenas as colunas de custo reduzido mínimo são avaliadas, gerando no máximo uma coluna por máquina a cada iteração.

Há muitas colunas disponíveis à geração, nem todas pertencentes ao conjunto mínimo para convergência. Contudo, a geração de colunas é um algoritmo finito que gradualmente constrói o mestre restrito, adicionando novas colunas, aumentando o custo reduzido $\bar{r}_k$ das colunas e melhorando a função objetivo de minimização do PAG, até que o *pricing* em cada máquina não seja mais negativo. No contexto teórico, a degenerescência retarda a convergência mas não torna a geração de colunas um processo infinito.

Essa geração iterativa de colunas é realizada até que o custo reduzido em todos os subproblemas das regiões convexas Q_i se torne nulo. Quando os custos reduzidos são nulos (normalmente maior que um parâmetro $-\epsilon$) para todas as máquinas, nenhuma nova coluna é gerada e a convergência do algoritmo é aceita. A convergência indica que a solução corrente do mestre restrito já sustenta um LB para a formulação do PAG.

1. Inicializar o mestre restrito com um conjunto mínimo de colunas articiais com custos muito altos para assegurar a viabilidade inicial do modelo.

2. Inicializar os m modelos inteiros dos subproblemas de mochila para cada máquina i.

3. Resolver a relaxação linear do mestre restrito.

4. Atualizar os subproblemas, sobretudo as funções objetivo com os valores correntes das duais de restrições do mestre restrito.

5. Resolver cada subproblema de mochila, seja por programação inteira ou por programação dinâmica, analisando as soluções inteiras encontradas de forma a criar novas colunas no mestre restrito que tenham custo reduzido aceitável.

6. Se alguma coluna houver sido adicionada ao mestre, repetir desde o passo 3.

7. Encerrar a geração de colunas porque nenhuma nova coluna foi encontrada para o mestre restrito.

***Script* 2** Geração de colunas para instâncias PAG

Data: tolerância $\epsilon > 0$ para avaliar o custo reduzido de colunas na geração.

Result: modelo de Dantzig-Wolfe DWM-R restrito à base de colunas geradas.

- **Entrada**: conjuntos $M = \{1, \ldots, m\}$ (máquinas) e $N = \{1, \ldots, n\}$ (tarefas);
- **Entrada**: valores de capacidade de máquina b_i para cada máquina $i = 1, \ldots, m$;
- **Entrada**: custos de atribuição c_{ij} para cada máquina $i \in M$ e tarefa $j \in N$;
- Constrói modelo mestre restrito (Dantzig-Wolfe) DWM-R com zero colunas;
- Inicializa o modelo DWM-R (mestre restrito) com variáveis artificiais;
- Inicializa o conjunto K de colunas geradas com zero colunas (vazio);

repeat

- Resolve o mestre restrito DWM-R, computando a solução linear Z_{RM};
- Atualiza $\mu_j^{RM}\ \forall\ j \in N$ com as duais das restrições de tarefa do DWM-R;
- Atualiza $\pi_i^{RM}\ \forall\ i \in M$ com as duais das restrições de máquina do DWM-R;
- Esvazia o conjunto K de colunas geradas (iteração anterior);

for $i \in M$ **do**

- Resolve o *pricing* (mochila) para máquina (i), usando as duais μ_j^{RM} e π_i^{RM};
- Computa o custo reduzido $\bar{r}_k$ da coluna k (máquina i) candidata à geração;

if $\bar{r}_k \leq -\epsilon$ **then**

- Insere a coluna $k \in K$ no modelo mestre restrito DWM-R;
- Insere a coluna k gerada no conjunto K;

until $K \neq \varnothing$;

- **Retorna** modelo resolvido DWM-R;

A técnica de estabilização, seja por penalização ou ponderação de duais, é recomendada para instâncias do PAG para as quais a geração de colunas demora um longo tempo para convergir.

Desta forma, essa estratégia permite a formação de uma base reduzida de colunas adequadas ao modelo de mestre restrito. Essa base reduzida, seja mínima ou não, torna o mestre restrito viável e confere acesso à solução ótima da relaxação linear da formulação de decomposição.

O maior desafio dessa abordagem é garantir uma solução inteira a partir da base de colunas encontrada. Logo, expandir essa base de colunas geradas, se necessário, para tornar o mestre restrito inteiro (MIP) viável com solução ótima para a instância do PAG.

No início do algoritmo de geração de colunas (algorithm 2), o mestre restrito é inicializado com um conjunto artificial de colunas para que seja viável e tenha uma solução inicial. A partir dessa base inicial de colunas e da solução inicial correspondente, colunas são adicionadas ao mestre restrito, melhorando a base gradativamente. O *pricing* é realizado através dos problemas de mochila, avaliando os custos reduzidos de novas colunas candidatas.

Em cada iteração, o algoritmo atualiza a função objetivo de cada subproblema de mochila com as atuais duais das restrições de tarefa e das restrições de máquina, baseado especificamente no custo reduzido enunciado em (3-13). Seguindo esse procedimento, os subproblemas são resolvidos e novas colunas de custo reduzido são geradas e introduzidas no mestre restrito.

Em relação ao *pricing*, a solução ótima do subproblema da máquina i indica uma nova coluna λ_i^k com uma atribuição de tarefas j à máquina i representada pelos coeficientes d_{ij}^k, fornecendo ao mestre restrito a mesma alocação de tarefas encontrada na solução da mochila para a máquina i. Portanto, se o item j está alocado à mochila i na solução do subproblema, então o coeficiente d_{ij}^k da variável de coluna λ_i^k assume valor 1; caso contrário, esse coeficiente assume valor 0.

Quando uma coluna λ_i^k é gerada para o mestre restrito, ela é adicionada à função objetivo do mestre restrito com o coeficiente linear igual à soma dos custos c_{ij} de todas as atribuições $(i,\ j)$ desta coluna, atualizando a função objetivo como descrito na Equação 3-23. Assuma que T_i^k é o conjunto de todas as tarefas atribuídas à máquina i pela coluna k.

$$Z_{mestre_{LP}}^{t\,+\,1} = Z_{mestre_{LP}}^{t} + \left(\sum_{j \in T_i^k} c_{ij}\ d_{ij}^k\ \lambda_i^k \right) \quad \forall \quad i\ =\ 1,\ \ldots,\ m \qquad \text{(3-23)}$$

O incremento da função objetivo $Z_{mestre_{LP}}^{t}$ da iteração corrente ocorre através do *pricing* de colunas até que a convergência seja detectada. No término do algoritmo, o mestre restrito apresenta o melhor LB para a instância do PAG.

Em seguida, o algoritmo de *price-and-cut* estudado prossegue para a adição de desigualdades 3-SRCs, tentando aumentar o LB encontrado, reduzindo o *gap* entre *bounds* e, assim, aproximando a base de colunas do mestre restrito à base de colunas que abarca a solução ótima da instância do PAG.

3.6 Algoritmo de Geração de Colunas com Estabilização de Wentges

Em caso de degenerescência da instância, a geração de colunas tende a ser demasiadamente lenta devido à ineficiência da geração de cortes na formulação mestre dual. Quando o *pricing* não separa colunas adequadas para uma

rápida convergência, técnicas de estabilização devem ser adotadas para melhorar a sequência de soluções duais encontradas durante a geração de colunas. A estabilização é um diferencial para resolver algumas instâncias retiradas da OR-Library que são muito degeneradas.

A sucessão de soluções duais iterativamente geradas para o mestre restrito implica em duais de atribuição $\mu_j \in \Pi^A$ para toda tarefa $j = 1, \ldots, n$ e duais de máquina $\pi_i \in \Pi^M$ para toda máquina $i = 1, \ldots, m$. Essas duais definem a função objetivo dos subproblemas de mochila, influenciando diretamente a qualidade das soluções encontradas pelo *pricing* em cada máquina do PAG. Consequentemente, a qualidade das colunas geradas é afetada visto que o custo reduzido das colunas separadas a cada iteração depende do *pricing*.

Em situações de lenta convergência das soluções primais do mestre restrito, pelo menos uma causa desse comportamento precisa ser identificada para que a degenerescência possa ser devidamente corrigida. Existe pelo menos dois fatores que retardam a convergência da geração de colunas, uma dualidade erraticamente oscilante e o efeito de *Tailing-off*.

O efeito de *tailing-off* ocorre quando as colunas adicionadas ao mestre restrito contribuem apenas com melhoria marginal da solução, diminuindo a velocidade de convergência do método. Senão, o retardo de convergência pode também ser ocasionado pela oscilação acentuada das soluções duais. Isso acontece por causa da instabilidade dual, até mesmo não monotônica, que conduz o *pricing* a encontrar colunas com custo reduzidos que aumentam e diminuem, pulando entre extremos, ora se distanciando ora se aproximando da melhor solução dual conhecida. Essa oscilação dual precisa ser amortecida para que a convergência seja acelerada, gerando melhores cortes duais, logo uma melhor base de colunas em um período de tempo mais curto (menos iterações de *pricing*).

A técnica de estabilização proposta por Pigatti *et al.* (2005) é um exemplo de estratégia com resultados positivos para reduzir a degenerescência de algumas instâncias do PAG. Essa estratégia propõe a introdução de penalidades através da adoção de variáveis de folga e de excesso na restrição de capacidade do subproblema de mochila. Desta forma, as oscilações são penalizadas e as variações drásticas das duais de restrição no *pricing* são suavizadas, estabilizando assim os custos reduzidos calculados e acelerando a convergência da geração de colunas. Embora essa prática de penalização seja um método eficiente, o ajuste de parâmetros exige bastante cuidado. Entretanto, o algoritmo apresentado neste texto difere na forma como corrige as oscilações de duais durante a geração de colunas.

O algoritmo de *price-and-cut* defendido nesta dissertação escolhe uma forma de estabilização diferente, usando a decomposição de Dantzig-Wolfe ponderada, defendida por Wentges (1997) e tratada na Seção 3.1. Essa abordagem mantém uma

solução dual de referência que corresponde ao melhor LB conhecido e computa um amortecimento das duais correntes por ponderação com as duais de referência ao invés de penalizar as oscilações duais, conforme detalhado na Seção 3.4.

$$
\begin{aligned}
Z_{LD} = \min \; & \sum_{i=1}^{m} \sum_{j=1}^{n} \left(c_{ij} \; - \; \mu_j^{ST}\right) y_{ij} \; + \; \sum_{j=1}^{n} \mu_j^{ST} \\
s.\,t. \; & \\
& \sum_{j=1}^{n} w_{ij} y_{ij} \; \leq \; b_i \quad \forall \; i = 1, \ldots, m \\
& y_{ij} \in \{0,\, 1\} \quad \forall \; i = 1, \ldots, m \; ; \; j = 1, \ldots, n
\end{aligned}
\tag{3-24}
$$

Esse método recorre à relaxação Lagrangeana da formulação do PAG para computar eficientemente um LB pela dual Lagrangeana definida em (3-24), a partir de duais de referência. Em qualquer iteração, quando a dual Lagrangeana encontra um LB melhor do que o *bound* de referência, a melhor solução dual guardada é substituída pelas duais estabilizadas desta iteração.

Inicialmente, uma solução linear precisa ser determinada como solução de referência para a estabilização e variáveis artificiais são adicionadas ao mestre restrito para que ele seja viável.

Em vista disso, o método instancia e soluciona um modelo de relaxação linear da formulação clássica do PAG com o Gurobi. A solução obtida para esse modelo inicializa o algoritmo de geração de colunas com a estabilização de Wentges, guardando as duais de restrição de atribuição μ_j^{LB} e as duais para restrição de máquina π_i^{LB} como referência para as primeiras ponderações de duais.

O algoritmo seguirá resolvendo o *pricing* com valores de duais ponderadas μ_j^{ST} e π_i^{ST}, guardando as melhores duais conhecidas (referência), representadas por μ_j^{LB} e π_i^{LB}. Em cada iteração da geração de colunas, as duais de referência guardadas μ_j^{LB} e π_i^{LB} e as duais correntes do mestre restrito μ_j^{RM} e π_i^{RM} são ponderadas segundo um parâmetro $0 \; < \; \alpha \; \leq \; 1$, segundo Equação 3-25 e Equação 3-26, estabilizando o custo reduzido de colunas encontradas para cada máquina. Então, considere o parâmetro α da ponderação e também um parâmetro de tolerância $\gamma > 0$ usado como condição de parada da estabilização (convergência).

$$
\pi_i^{ST} \; = \; \alpha \, \pi_i^{RM} \; + \; (1 \; - \; \alpha) \, \pi_i^{LB} \quad \forall \quad i = 1, \; \ldots, \; m \tag{3-25}
$$

$$
\mu_j^{ST} \; = \; \alpha \, \mu_j^{RM} \; + \; (1 \; - \; \alpha) \, \mu_j^{LB} \quad \forall \quad j = 1, \; \ldots, \; n \tag{3-26}
$$

Portanto, o algoritmo 2 de geração de colunas é adaptado para considerar as duais estabilizadas μ_j^{ST} e π_i^{ST} na formulação do *pricing*, conforme escrito em (3-27), para o algoritmo 3 de geração de colunas com estabilização Wentges.

$$
\begin{aligned}
Z^i_{knapsack} &= \max\left(\sum_{j=1}^{n} \left(\left(\alpha\, \mu_j^{RM} \;+\; (1 \;-\; \alpha)\, \mu_j^{LB} \right) \;-\; c_{ij} \right) y_{ij} \right) \\
Z^i_{knapsack} &= \max\left(\sum_{j=1}^{n} \left(\mu_j^{ST} \;-\; c_{ij} \right) y_{ij} \right) \\
& \qquad s.\,t. \\
& \qquad\quad \sum_{j=1}^{n} w_{ij}\, y_{ij} \leq b_i \\
& \qquad\quad y_{ij} \in \{0,\, 1\} \quad \forall \quad j = 1, \ldots,\, n
\end{aligned}
\tag{3-27}
$$

Algumas outras alterações são realizadas, principalmente alterações para avaliar o melhor LB conhecido a cada iteração e identificar a adequação do valor de α rumo à convergência da solução do modelo.

- Considere que Π_A^{RM} seja o vetor de duais de atribuição $\mu_j^{RM}\ \forall\ j = 1,\ \ldots,\ n$;
- Considere que Π_M^{RM} seja o vetor de duais de máquina $\pi_i^{RM}\ \forall\ i = 1,\ \ldots,\ m$;
- Considere que Π_A^{LB} seja o vetor de duais de atribuição $\mu_j^{LB}\ \forall\ j = 1,\ \ldots,\ n$;
- Considere que Π_M^{LB} seja o vetor de duais de máquina $\pi_i^{LB}\ \forall\ i = 1,\ \ldots,\ m$;
- Considere que Π_A^{ST} seja o vetor de duais de atribuição $\mu_j^{ST}\ \forall\ j = 1,\ \ldots,\ n$.
- Considere que Π_M^{ST} seja o vetor de duais de máquina $\pi_i^{ST}\ \forall\ i = 1,\ \ldots,\ m$.

O parâmetro α pode ser constante ou dinâmico. Qualquer ajuste de α no decorrer da geração de colunas implicará na alteração de ponderação das duais conforme a Equação 3-25 e a Equação 3-26, aumentando ou diminuindo o grau de alteração das duais μ_j^{RM} e π_i^{RM} do mestre restrito pelas duais de referência μ_j^{LB} e π_i^{LB}. Para um valor muito baixo de α, as duais estabilizadas utilizada pelo *pricing* se manterão muito mais próximas das duais de referência μ_j^{LB} e π_i^{LB}. Para um valor muito alto de α, as duais estabilizadas utilizada pelo *pricing* se manterão muito mais próximas das duais correntes do mestre restrito μ_j^{RM} e π_i^{RM}.

No início da geração de colunas, um valor mais baixo de α favorece uma aceleração da convergência da solução do mestre restrito para o valor do LB de referência. A partir do momento que as soluções primais e duais se aproximam da convergência, o valor baixo de α pode impedir a geração de colunas benéficas para o mestre restrito, de modo que a convergência não avança mais.

***Script* 3** Geração de colunas com estabilização de Wentges (PAG)

Data: parâmetro $0 < \alpha \leq 1$ para a estabilização; $\epsilon > 0$ para avaliar a geração de colunas; $\gamma > 0$ para avaliar a convergência da estabilização.

Result: modelo de Dantzig-Wolfe DWM-R restrito à base de colunas geradas.

- **Entrada**: conjuntos $M = \{1, \ldots, m\}$ (máquinas) e $N = \{1, \ldots, n\}$ (tarefas);
- **Entrada**: valores de capacidade de máquina b_i para cada máquina $i = 1, \ldots, m$;
- **Entrada**: custos de atribuição c_{ij} para cada máquina $i \in M$ e tarefa $j \in N$;
- Constrói modelo mestre restrito (Dantzig-Wolfe) DWM-R com zero colunas;
- Inicializa o modelo DWM-R (mestre restrito) com variáveis artificiais;
- Inicializa o conjunto K de colunas geradas com zero colunas (vazio);
- Resolve a relaxação linear clássica do PAG com solução Z_{LR} (*warm start*);
- Inicializa duais de referência μ_j^{LB} com duais μ_j^{LR} da solução Z_{LR} para todo $j \in N$;
- Inicializa duais de referência π_i^{LB} com duais π_i^{LR} da solução Z_{LR} para todo $i \in M$;
- Computa a relaxação Lagrangeana inicial $L(\Pi_A^{LB})$ com base nas duais μ_j^{LB};

repeat

- Resolve o mestre restrito DWM-R, computando a solução linear Z_{RM};
- Atualiza vetor Π_A^{RM} com as duais das restrições de tarefa do DWM-R;
- Atualiza vetor Π_M^{RM} com as duais das restrições de máquina do DWM-R;
- Atualiza duais estabilizadas $\mu_j^{ST} = \alpha\ \mu_j^{RM} + \ (1 - \alpha)\ \mu_j^{LB} \quad \forall \quad j \in N$;
- Atualiza duais estabilizadas $\pi_i^{ST} = \alpha\ \pi_i^{RM} + \ (1 - \alpha)\ \pi_i^{LB} \quad \forall \quad i \in M$;
- Computa a relaxação Lagrangeana $L(\Pi_A^{ST})$;
- Esvazia o conjunto K de colunas geradas (iteração anterior);

for $i \in M$ **do**

- Resolve o *pricing* (mochila) para máquina (i), usando as duais π_j^{ST} e π_i^{ST};
- Computa o custo reduzido $\bar{r}_k$ da coluna k (máquina i) candidata à geração;
- **if** $\bar{r}_k \leq -\epsilon$ **then**
  - Insere a coluna k no mestre restrito DWM-R e no conjunto K;

if $L(\Pi_A^{ST}) > L(\Pi_A^{LB})$ **then**

- Atualiza as duais de referências $\mu_j^{LB} \leftarrow \pi_j^{ST} \quad \forall \quad j \in N$;
- Atualiza as duais de referências $\pi_i^{LB} \leftarrow \pi_i^{ST} \quad \forall \quad i \in M$;
- Atualiza a relaxação Lagrangeana $L(\Pi_A^{LB})$;

if $Z_{RM} - L(\Pi_A^{LB}) \leq \gamma$ **then**

- Atualiza $\alpha \leftarrow 1.0$ para interromper a estabilização;

if Z_{RM} ***não muda*** *em 100 iterações consecutivas* **then**

- Atualiza $\alpha = \alpha + 0.2$;
- **if** $\alpha > 1.0$ **then**
  - Atualiza $\alpha \leftarrow 1.0$ para interromper a estabilização;

until $K \neq \varnothing$;

- **Retorna** modelo resolvido DWM-R;

Nesse caso, o valor de α deve ser aumentado heuristicamente, desbloqueando o avanço da convergência até que α alcance o valor 1. Quando α assume o valor 1, ocorre a parada da estabilização porque apenas as duais μ_j^{RM} e π_i^{RM} do mestre restrito passam a ser consideradas pelo *pricing*, sem ponderação.

O algoritmo de *price-and-cut* implementado começa com $\alpha = 0.1$ e aumenta a incrementos de 0.2 toda vez que a estabilização bloqueia o avanço da convergência na geração de colunas. A justificativa para esses ajustes do parâmetro α reside no fato de que esse valor inicial de 0.1 e os valores de incrementos de 0.2 tiveram melhor eficiência na geração de colunas para instâncias degeneradas do PAG.

Após uma iteração, o mestre restrito do PAG é alterado com a inserção de novas colunas. Então, a dual Lagrangeana $L(\Pi_A^{ST})$ é novamente computada para que o LB, duais μ_j^{LB} correspondentes, seja atualizado se e somente se $L(\Pi_A^{ST}) > L(\Pi_A^{LB})$. Não há necessidade de calcular novamente $L(\Pi_A^{LB})$ já que, havendo a atualização $\Pi_A^{LB} = \Pi_A^{ST}$, consequentemente $L(\Pi_A^{LB}) = L(\Pi_A^{ST})$.

A condição de parada da estabilização é pautada pela convergência, sendo interrompida quando a distância entre a solução primal do mestre restrito e a dual Lagrangeana $L(\Pi_A^{LB})$ se torna suficientemente estreita.

O parâmetro $\gamma > 0$ serve para avaliar essa condição de parada, verificando se $Z_{RM} - L(\Pi_A^{LB}) \leq \gamma$. Caso seja, α é atualizado para 1, parando a ponderação das duais e prosseguindo com a geração de colunas padrão. De acordo com os experimentos realizados, o valor de γ escolhido foi 0.01 para as instâncias do PAG.

Posteriormente, ao término da geração de colunas, o *price-and-cut* separa e introduz desigualdades do tipo 3-SRC no mestre restrito. Quando há 3-SRCs no modelo, a estabilização passa a depender de modificações que reconheçam a existência desses cortes na formulação. Essa adaptação da estabilização não foi implementada nem estudada, sobretudo porque a degenerescência crítica aparece no início do algoritmo, somente com uma base de colunas inicial no mestre restrito.

4
Introdução de Desigualdades (3, 0.5)-SRC

A finalidade do algoritmo *price-and-cut* é orientada pela perspectiva de que um LB maior pode ser obtido pela combinação do *pricing* e da introdução de cortes adequados, tendo em vista viabilizar a enumeração de colunas para a maioria das instâncias do PAG.

A primeira parte dessa estratégia compete ao *pricing*, realizado pela geração de colunas sobre a relaxação linear do mestre restrito da formulação de decomposição de Dantzig-Wolfe (3-6).

Essa geração de colunas resulta no mestre restrito composto por uma reduzida base de colunas que confere ao mestre restrito uma solução linear ótima. A solução primal desse mestre restrito serve como um LB para a instância do PAG. Assim, o *gap* mínimo $UB - LB$ para o mestre restrito gerado é definido, onde (UB) é o menor UB conhecido.

O objetivo da segunda parte, a separação de desigualdades (cortes do tipo (3, 0.5)-SRC), é aumentar o LB para otimizar a viabilização da enumeração de colunas. Quanto menor o *gap* $UB - LB$, menor o espaço de busca por colunas percorrido pela enumeração de colunas, favorecendo a adição de todas as colunas que faltam à base de colunas do mestre restrito para que o modelo tenha uma solução inteira ótima para a instância do PAG. Esse *gap* é relevante porque a enumeração de colunas avalia somente colunas com custo reduzido $\bar{r}_k \leq UB - LB$.

Em vista disso, a separação de desigualdades do tipo (3, 0.5)-SRC é motivada pela possibilidade de otimização da enumeração de colunas. Portanto, a segunda parte do *price-and-cut* deve ser capaz de gerar essas desigualdades afim de que aumente o LB do mestre restrito relaxado resolvido por geração de colunas.

O algoritmo permanece executando um ciclo alternado entre *pricing* (geração de colunas) e *cut* (separação de desigualdades (3, 0.5)-SRC) até que nenhum nova desigualdade adequada seja identificada para o mestre restrito. Nesse momento, o algoritmo encerra o *price-and-cut* com um LB maximizado.

Logo, o algoritmo de *price-and-cut* atua como um maximizador de LB e, consequentemente, um minimizador do *gap*.

4.1 Separação de Desigualdades

A introdução de desigualdades é uma abordagem escolhida para suprir a necessidade de aumento do LB do mestre restrito gerado pelo *pricing*. Esse aumento é relevante para que a distância entre o LB e o UB seja reduzida ao máximo, aproximando o conjunto de colunas geradas ao conjunto de colunas que formam uma base suficiente para uma solução inteira potencialmente ótima. Caso uma enumeração de colunas completa seja viável para o *gap* obtido, a solução inteira desse mestre restrito tem otimalidade comprovada.

O método implementado dispõe-se a separar um conjunto de desigualdades válidas para o mestre restrito, conhecidas como *subset row cuts (SRC)* (Jepsen *et al*, 2008) e obtidas pelo procedimento de arredondamento de Chvátal-Gomory aplicado sobre as restrições (3-5) de atribuição de tarefa a uma máquina na formulação do mestre restrito. Quanto à nomenclatura, assuma que o índice i de máquina da coluna λ_k é identificado pela coluna de índice k dado que que cada coluna é definida para uma única máquina i.

$$\sum_{k \in P} \left(\lfloor q \sum_{j \in L} d_j^k \rfloor \lambda_k \right) \leq \lfloor q \, |L| \rfloor \tag{4-1}$$

Essas desigualdades, escritas em (4-1), são definidas como uma combinação linear de restrições L, denominadas *rows*, e por um mulplicador $q \in [0, 1]$. Em razão das *rows* L serem restrições de atribuição de tarefas, o lado esquerdo da inequação (4-1) será, por definição, uma combinação linear de todas as colunas $\lambda_k \in P_i$ que estão contidas nesses *rows*.

Em perspectiva de exemplificação, considere uma (3, 0.5)-SRC v definida pela tupla de restrições de tarefas $L = 1, 5, 9$. Assuma que existe uma coluna λ_1 que atribui as tarefas $j = 1$ e $j = 5$ à máquina i. Assuma também que existe outra coluna λ_2 que atribui as tarefas $j = 1$, $j = 5$ e $j = 9$ à máquina i. Ainda admita a existência de uma terceira coluna λ_3 que atribui tarefas $j = 1$ e $j = 4$ à máquina i. Pela definição (4-1), admitindo que existem apenas essas três colunas, o lado esquerdo da restrição de (3, q)-SRC é escrita como indicado em (4-2).

$$\lfloor q \, (d_1^1 + d_5^1) \rfloor \, \lambda_1 \, + \, \lfloor q \, (d_1^2 \, + \, d_5^2 \, + \, d_9^2) \rfloor \, \lambda_2 \, + \, \lfloor q \, (d_1^3) \rfloor \, \lambda_3 \, \leq \, \lfloor 3 \, q \rfloor \tag{4-2}$$

Então, a desigualdade (3, 0.5)-SRC (4-1) analisada, uma vez que seja adicionada à formulação do mestre restrito, passa a ser uma nova restrição do mestre restrito, assumindo a forma da inequação (4-1). Dado que a separação de cortes visa o aumento do LB, somente são adicionadas as desigualdades violadas que

aumentam o valor da função objetivo do mestre restrito. As demais desigualdades analisadas, quando não aumentam a função objetivo do mestre, são descartadas.

Em síntese, a restrição de qualquer (L, q)-SRC estará sendo violada sempre que o lado esquerdo for maior que o lado direito da inequação (4-1). Assim que a desigualdade for adicionada à formulação do mestre restrito, o modelo será resolvido e um novo valor de função objetivo será encontrado, um novo LB. Apenas compensa adicionar a desigualdade em caso de aumento desse LB, estreitando assim o *gap* relativo ao UB de referência.

Portanto, uma (3, 0.5)-SRC candidata será adicionada ao modelo mestre restrito se e somente se todas as seguintes condições listadas abaixo forem verdadeiras.

1. O valor do lado esquerdo da inequação (4-1) da desigualdade (3, 0.5)-SRC está sendo violada pela atual solução do mestre restrito.

2. O valor da função objetivo do mestre restrito aumentar em decorrência da inserção da restrição (4-1) da desigualdade (3, 0.5)-SRC.

Assumindo o exemplo enunciado anteriormente em (4-2), há alguns cenários para avaliar essa restrição de (3, 0.5)-SRC em relação à violação em função das colunas.

☐ Cenário A: $\lambda_1 = 1, \lambda_2 = 0, \lambda_1 = 0$

$$\lfloor 0.5\,(d_1^1 + d_5^1)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^2 \;+\; d_5^2 \;+\; d_9^2)\rfloor\,0 \;+\; \lfloor 0.5\,(d_1^3)\rfloor\,0 \;\leq\; \lfloor 1.5\rfloor$$
$$\lfloor 0.5\,(1+1)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$1 \;\leq\; 1$$

☐ Cenário B: $\lambda_1 = 0, \lambda_2 = 1, \lambda_1 = 0$

$$\lfloor 0.5\,(d_1^1 + d_5^1)\rfloor\,0 \;+\; \lfloor 0.5\,(d_1^2 \;+\; d_5^2 \;+\; d_9^2)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^3)\rfloor\,0 \;\leq\; \lfloor 1.5\rfloor$$
$$\lfloor 0.5\,(1+1+1)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$1 \;\leq\; 1$$

☐ Cenário C: $\lambda_1 = 0, \lambda_2 = 0, \lambda_1 = 1$

$$\lfloor 0.5\,(d_1^1 + d_5^1)\rfloor\,0 \;+\; \lfloor 0.5\,(d_1^2 \;+\; d_5^2 \;+\; d_9^2)\rfloor\,0 \;+\; \lfloor 0.5\,(d_1^3)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$\lfloor 0.5\,(1)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$0 \leq\; 1$$

☐ Cenário D: $\lambda_1 = 1, \lambda_2 = 1, \lambda_1 = 0$

$$\lfloor 0.5\,(d_1^1 + d_5^1)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^2 \;+\; d_5^2 \;+\; d_9^2)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^3)\rfloor\,0 \;\leq\; \lfloor 1.5\rfloor$$
$$\lfloor 0.5\,(1+1)\rfloor\,1 \;+\; \lfloor 0.5\,(1+1+1)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$2 \leq\; 1$$

☐ Cenário E: $\lambda_1 = 1, \lambda_2 = 0, \lambda_1 = 1$

$$\lfloor 0.5\,(d_1^1 + d_5^1)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^2 \;+\; d_5^2 \;+\; d_9^2)\rfloor\,0 \;+\; \lfloor 0.5\,(d_1^3)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$\lfloor 0.5\,(1+1)\rfloor\,1 \;+\; \lfloor 0.5\,(1)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$1 \leq\; 1$$

☐ Cenário F: $\lambda_1 = 0, \lambda_2 = 1, \lambda_1 = 1$

$$\lfloor 0.5\,(d_1^1 + d_5^1)\rfloor\,0 \;+\; \lfloor 0.5\,(d_1^2 \;+\; d_5^2 \;+\; d_9^2)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^3)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$\lfloor 0.5\,(1+1+1)\rfloor\,1 \;+\; \lfloor 0.5\,(1)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$1 \leq\; 1$$

☐ Cenário G: $\lambda_1 = 1, \lambda_2 = 1, \lambda_1 = 1$

$$\lfloor 0.5\,(d_1^1 + d_5^1)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^2 \;+\; d_5^2 \;+\; d_9^2)\rfloor\,1 \;+\; \lfloor 0.5\,(d_1^3)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$\lfloor 0.5\,(1+1)\rfloor\,1 \;+\; \lfloor 0.5\,(1+1+1)\rfloor\,1 \;+\; \lfloor 0.5\,(1)\rfloor\,1 \;\leq\; \lfloor 1.5\rfloor$$
$$2 \leq\; 1$$

A restrição é violada em dois cenários (D e G) onde as duas colunas ativadas λ_1 e λ_2 assumem valor 1. A coluna λ_3 não está ativada por ter apenas uma tarefa da (3, 0.5)-SRC avaliada. Então, esse exemplo também mostra que uma coluna ativa é toda coluna que contribui para aumentar o valor do lado esquerdo da restrição da (3, 0.5)-SRC. A coluna apenas contribui dessa forma quando ela atribui pelo menos duas tarefas da tupla L da desigualdade à máquina i onde a coluna está definida. Esse conceito de ativação da coluna é importante para a análise das (3, 0.5)-SRC nos subproblemas de mochila do *pricing*.

O algoritmo de separação de (3, 0.5)-SRCs, descrito em (4), começa pela enumeração das combinações L de três distintas restrições de atribuição de tarefa. Em instâncias com uma quantidade muito grande de tarefas, o número dessas combinações pode ser proibitivo. Logo, o algoritmo proposto analisa apenas um subconjunto das s restrições de atribuição de tarefas mais promissoras, onde s é um parâmetro ajustável. Essas combinações definem as restrições que participam da combinação linear definida para a desigualdade. Essa combinação linear conduz ao somatório de todos os termos $d_j^k \lambda_k$ (variáveis de coluna) de cada restrição selecionada.

Então, o algoritmo identifica todas as colunas implicadas por essas restrições e computa o valor atual l do lado esquerdo da inequação (4-1). Em seguida, esse valor l é utilizado para avaliar a violação da restrição da (3, 0.5)-SRC definida pela combinação L. Caso a restrição esteja violada, isso constitui uma indicação positiva para a pertinência da sua inserção na formulação do mestre restrito.

Em razão do resultado dessa violação, o algoritmo verifica se a inserção dessa restrição aumenta o LB encontrado pela solução do mestre restrito. Caso o aumento seja constatado, essa desigualdade (3, 0.5)-SRC é adicionada ao mestre restrito.

***Script* 4** Separação de desigualdades do tipo (3, 0.5)-SRC (PAG)

Data: modelo mestre restrito (Dantzig-Wolfe) DWM-R; número máximo $s > 0$ de linhas (*rows*) combinadas para análise de cortes (3, 0.5)-SRCs.

Result: modelo mestre restrito DWM-R com (3, 0.5)-SRCs separadas.

if *modelo DWM-R* ***não*** *viável* **then**
- **Retorna** modelo DWM-R;

- Ordena as colunas do modelo DWM-R (valor de variável decrescente) no vetor D;
- Inicializa o conjunto T de tarefas selecionadas;

for $k = 1; k \leq |D|; k = k + 1$ **do**
- Insere no conjunto T todas as tarefas atribuídas na coluna $D[k]$;
- **if** $|T| \geq s$ **then**
  - ***break***;

- Enumera o conjunto **R** de todas as combinações 3 a 3 de tarefas distintas de T;

repeat
- Remove de **R** uma combinação de 3 tarefas (possível corte 3-SRC **w**);
- Declara o conjunto **U** com todas as colunas das 3 restrições de tarefas de **w**;
- Computa o valor do lado esquerdo l da restrição de (3, 0.5)-SRC do corte **w**;
- **if** $l > 1.0 \rightarrow$ *(restrição de (3, 0.5)-SRC do corte w* ***violada*** *no modelo DWM-R)* **then**
  - Cria a restrição de (3, 0.5)-SRC do corte **w** no modelo DWM-R;
  - Resolve o mestre restrito com a nova (3, 0.5)-SRC **w** separada;
  - **if** *valor da função objetivo do modelo DWM-R* ***aumentou*** **then**
    - **for** $i \in$ *{subproblemas de mochila associados às colunas de U}* **do**
      - Insere o corte **w** no subproblema de mochila (máquina i);
        - ☐ Cria variável de decisão do corte **w** no subproblema de mochila (máquina i);
        - ☐ Cria as restrições da (3, 0.5)-SRC **w** no subproblema de mochila (máquina i);
  - **else**
    - Remove o corte (3, 0.5)-SRC **w** do modelo DWM-R;

until $R \neq \varnothing$;

- Resolve o modelo mestre restrito (Dantzig-Wolfe) DWM-R;
- **Retorna** modelo DWM-R;

A introdução de uma desigualdade (3, 0.5)-SRC no mestre restrito exige uma adaptação da formulação dos subproblemas de *pricing* para que as duais das novas restrições (3, 0.5)-SRC sejam consideradas no cálculo de custo reduzido. Essa adaptação do *pricing* está detalhada na Seção 4.2.

A implementação dessa geração de desigualdades (3, 0.5)-SRC apresenta algumas limitações pela característica combinatória de análise das possíveis (3, 0.5)-SRCs geráveis.

Entretanto, estabelecendo um valor adequado de s, a geração de desigualdades pode ser executada de forma eficiente. O aumento de s leva ao crescimento exponencial do número de combinação de restrição, logo um número exponencial de (3, 0.5)-SRCs. Portanto, esse parâmetro deve ser sempre devidamente ajustado para não tornar inviável essa geração de cortes. Em instância com até 200 tarefas, o algoritmo atua com uma ótima eficiência e consumo de memória, analisando todas as possíveis (3, 0.5)-SRCs. Em instâncias com mais de 200 tarefas, o parâmetro s precisa ser ajustado cuidadosamente. Os experimentos computacionais realizados nesta dissertação não avaliou instâncias muito maiores que 200 tarefas.

Quando o parâmetro s recebe um valor menor que o número de tarefas da instância, apenas um subconjunto de (3, 0.5)-SRCs é avaliado. A seleção desse subconjunto é feita heuristicamente, ordenando as colunas do mestre restrito em ordem decrescente de valor de $d_j^k \lambda_k$ e percorrendo as colunas nessa ordem, adicionando as tarefas j encontradas para as colunas até que s tarefas tenham sido adicionadas. Essa heurística favorece a análise de desigualdades (3, 0.5)-SRC mais prováveis de constituir restrições (3, 0.5)-SRC violadas na formulação do mestre restrito.

No contexto do PAG, as diversas possíveis combinações de tamanho 3 das restrições de atribuição de tarefas são percorridas e analisadas, efetuando a tentativa de localizar desigualdades (3, 0.5)-SRCs que possam ser separadas para obter algum acréscimo do LB do mestre restrito relaxado. Considere que as (3, 0.5)-SRCs são definidas como *subset row cuts* de cardinalidade $|L| = 3$ e multiplicador $q = 0.5$. Então, essas desigualdades são compostas pela combinação linear de 3 restrições de tarefas (3-5), somando as respectivas colunas λ_k de cada uma dessas restrições.

Naturalmente, pela definição da restrição (4-1), uma (3, 0.5)-SRC produz uma restrição (4-3) dentro do mestre restrito. Essas restrições de corte são combinações lineares de colunas associadas a cada uma das três restrições de tarefa (3-5) que compõem a combinação da respectiva (3, 0.5)-SRC. Assim sendo, o coeficiente das colunas d_j^k na combinação linear de três restrições de tarefas j_1, j_2 e j_3 da (3, 0.5)-SRC será determinado pela expressão (4-3).

$$\sum_{k \in P_i} \left(\lfloor 0.5 \sum_{j \in w_v} d_j^k \rfloor \, \lambda_k \right) \leq 1.0 \quad \forall \quad v = 1, \ldots, |S| \tag{4-3}$$

A combinação linear de restrições reunidas por uma (3, 0.5)-SRC é a soma das colunas $d_j^k \lambda_k$, contabilizando a quantidade de ocorrência de cada coluna nas três restrições. Se a tarefa j está contemplada pela (3, 0.5)-SRC, então os termos $d_j^k \lambda_k$ são somados com o multiplicador 0.5 na combinação linear que resulta na restrição da desigualdade para toda coluna $k = 0, \ldots, P$.

- Caso a variável d_j^k seja 1, então a coluna λ_k é um bloco da máquina i que processa a tarefa j na coluna de índice k;
- Caso a variável d_j^k seja 0, então a coluna λ_k é um bloco da máquina i que não aloca a tarefa j na coluna de índice k;
- Caso uma coluna de índice k apareça em apenas uma restrição da (3, 0.5)-SRC, então a coluna λ_k terá coeficiente igual a $\lfloor 0.5\lambda_k \rfloor$;
- Caso uma coluna de índice k apareça em exatamente duas restrições da (3, 0.5)-SRC, então a coluna λ_k terá coeficiente igual a $\lfloor 1\lambda_k \rfloor$;
- Caso uma coluna de índice k apareça em exatamente três restrições da (3, 0.5)-SRC, então a coluna λ_k terá coeficiente igual a $\lfloor 1.5\lambda_k \rfloor$;

A separação de desigualdades (3, 0.5)-SRCs inicia pela enumeração de todas as combinações de 3 restrições de atribuição de tarefas, semelhante a listar todas as combinaçõees das n tarefas, três a três. Dada uma (3, 0.5)-SRC enumerada, somente será inserida no mestre restrito caso seja um corte com restrição violada no mestre restrito e cuja inserção aumente o LB corrente.

Após a introdução de todas as (3, 0.5)-SRCs separadas, a geração de colunas é novamente executada até que todas as colunas geradas pelo *pricing* tenham custo reduzido nulo.

Durante a geração de colunas, conforme novas colunas são geradas para o mestre restrito, as restrições de atribuição de tarefa são alteradas (colunas adicionadas). Essa modificação exige que as restrições de (3, 0.5)-SRCs existentes sejam atualizadas, tendo em consideração as novas colunas. Qualquer alteração nas restrições de (3, 0.5)-SRCs requer a atualização dos subproblemas de mochila do *pricing*, obrigatoriamente, para que o *pricing* seja compatível com o mestre restrito durante a geração de colunas.

Essa versão da geração de colunas com desigualdades (3, 0.5)-SRC está descrita abaixo em 5. A estabilização de Wentges implementada não está adaptada para a geração de colunas com essas desigualdades. Esse ciclo de *price-and-cut* é repetido até que nenhuma nova desigualdade (3, 0.5)-SRC seja encontrada.

***Script* 5** Geração de colunas com (3, 0.5)-SRCs (PAG)

Data: tolerância $\epsilon > 0$ para avaliar o custo reduzido de colunas na geração.

Result: modelo de Dantzig-Wolfe DWM-R restrito à base de colunas geradas.

- **Entrada**: conjuntos $M = \{1, \ldots, m\}$ (máquinas) e $N = \{1, \ldots, n\}$ (tarefas);
- **Entrada**: valores de capacidade de máquina b_i para cada máquina $i = 1, \ldots, m$;
- **Entrada**: custos de atribuição c_{ij} para cada máquina $i \in M$ e tarefa $j \in N$;
- Constrói modelo mestre restrito (Dantzig-Wolfe) DWM-R com zero colunas;
- Inicializa o modelo DWM-R (mestre restrito) com variáveis artificiais;
- Inicializa o conjunto K de colunas geradas com zero colunas (vazio);

repeat

- Resolve o mestre restrito DWM-R, computando a solução linear Z_{RM};
- Atualiza $\mu_j^{RM}\ \forall\ j \in N$ com as duais das restrições de tarefa do DWM-R;
- Atualiza $\pi_i^{RM}\ \forall\ i \in M$ com as duais das restrições de máquina do DWM-R;
- Esvazia o conjunto K de colunas geradas (iteração anterior);

for $i \in M$ **do**

- Resolve o *pricing* (mochila) para máquina (i), usando as duais μ_j^{RM} e π_i^{RM};
- Computa o custo reduzido $\bar{r}_k$ da coluna k (máquina i) candidata à geração;

if $\bar{r}_k \leq -\epsilon$ **then**

- Insere a coluna $k \in K$ no modelo mestre restrito DWM-R;
- Insere a coluna k gerada no conjunto K;

- Esvazia o conjunto D de (3, 0.5)-SRCs implicadas pelas colunas geradas (K);

for $k \in K$ **do**

- Identifica todas (3, 0.5)-SRCs implicadas pela coluna gerada k;
- Armazena no conjunto D todas as (3, 0.5)-SRCs identificadas;

for $v \in D$ **do**

- Atualiza a restrição da (3, 0.5)-SRC v do modelo DWM-R;

- Resolve o mestre restrito DWM-R, computando a solução linear Z_{RM};

for $v \in D$ **do**

- Atualiza as restrições da (3, 0.5)-SRC v nos subproblemas de mochila;

until $K \neq \varnothing$;

- **Retorna** modelo resolvido DWM-R;

4.2 Pricing da Decomposicão com (3, 0.5)-SRCs

Em situações de inserção de uma desigualdade (3, 0.5), a modificação da formulação do mestre restrito altera a estrutura de subproblemas do *pricing*. Essas restrições introduzidas pelas desigualdades impoẽm uma adaptação da expressão de cálculo do custo reduzido das colunas para que as duais associadas às (3, 0.5)-SRCs sejam consideradas pelo *pricing*.

Logo, a função objetivo dos subproblemas de mochila são atualizadas, pela definição, subtraindo as duais υ_v de restrições de (3, 0.5)-SRCs v, ativadas na função objetivo pela variável z_v somente quando pelo menos duas tarefas j são atribuídas à mochila (há contribuição para potencial violação da (3, 0.5)-SRC). Essa reformulação dos subproblemas depende do custo reduzido $\bar{r}_k$ (4-4) adaptado às (3, 0.5)-SRCs, onde N_k é o conjunto de tarefas j alocadas à máquina i da coluna k e S_k é o conjunto de desigualdades ativadas pelas tarefas contidas em N_k.

$$\bar{r}_k = \sum_{j \in N_k} \left(c_{ij} - \mu_j \right) - \pi_i - \sum_{v \in S_k} \upsilon_v z_v \tag{4-4}$$

O algoritmo passa a considerar as duais de restrições (L, q)-SRC sobre o custo reduzido computado para colunas em cada máquina, conforme a equação (4-4), onde z_v é uma variável de decisão binária que assume valor 1 se e somente se a (L, q)-SRC de índice v estiver ativada pelo conjunto de tarefas j atribuídas à máquina i na coluna k.

Em consideração à adaptação do *pricing*, ocasionada pela introdução de (3, 0.5)-SRCs, algumas modificações são implementadas para manter a viabilidade da geração de colunas. Ignorar as desigualdades na formulação dos subproblemas induz o *pricing* a computar custos reduzidos errados.

Em virtude disso, a correção da expressão do custo reduzido (4-4) não é suficiente para tornar a otimização do *pricing* compatível com a existência de (3, 0.5)-SRCs. É imperativo criar também novas restrições na formulação dos subproblemas de mochila para garantir a correta ativação das variáveis z_v, onde $z_v = 1$ quando a variável está ativada.

Pela definição de (3, 0.5)-SRC (4-3), quando pelo menos duas tarefas j da desigualdade é atribuída, pelo subproblema, à máquina i, a coluna gerada λ_i^k é somada à restrição da (3, 0.5)-SRC do mestre restrito, contribuindo para o lado esquerdo dessa restrição porque $\lfloor 0.5 \, (h\lambda_i^k) \rfloor \leq 1$, para $h \geq 2$, onde h é o somatório dos respectivos coeficientes d_{ij}^k e a inclusão de λi^k na solução do mestre restrito implica que $\lambda_i^k = 1$.

Considere uma (3, 0.5)-SRC composta por tarefas $T = \{1, 4, 8\}$ para exemplificar a ativação da desigualdade na otimização do *pricing*. Caso a solução

do subproblema da máquina i atribua a essa máquina os subconjuntos de tarefas $\{1,4\}$, $\{1,8\}$, $\{4,8\}$ ou $\{1,4,8\}$, então a coluna $\lambda_i^k = 1$ assumirá essa atribuição, ocorrendo os seguintes casos.

$$\sum_{j \in w_v} d_{ij}^k \lambda_i^k \geq 2 \tag{4-5}$$

Em todos os casos listados, essas alocações fazem que o lado esquerdo da restrição de (3, 0.5)-SRC aumente de valor, possivelmente causando a violação da inequação (4-3) quando a inequação (4-5) é verificada para colunas λ_i^k dessa restrição de desigualdade (3, 0.5)-SRC.

1. Se as tarefas $\{1,4\}$ estão atribuídas à máquina i e $\lambda_i^k = 1$, então:
   (a) $d_{i1}^k = 1$ e $d_{i4}^k = 1$
   (b) $(d_{i1}^k + d_{i4}^k)\lambda_i^k = (1+1)\lambda_i^k = 2$;
2. Se as tarefas $\{1,8\}$ estão atribuídas à máquina i e $\lambda_i^k = 1$, então:
   (a) $d_{i1}^k = 1$ e $d_{i8}^k = 1$
   (b) $(d_{i1}^k + d_{i8}^k)\lambda_i^k = (1+1)\lambda_i^k = 2$;
3. Se as tarefas $\{4,8\}$ estão atribuídas à máquina i e $\lambda_i^k = 1$, então:
   (a) $d_{i4}^k = 1$ e $d_{i8}^k = 1$
   (b) $(d_{i4}^k + d_{i8}^k)\lambda_i^k = (1+1)\lambda_i^k = 2$;
4. Se as tarefas $\{1,4,8\}$ estão atribuídas à máquina i e λ_i^k, então:
   (a) $d_{i1}^k = 1$, $d_{i4}^k = 1$ e $d_{i8}^k = 1$
   (b) $(d_{i1}^k + d_{i4}^k + d_{i8}^k)\lambda_i^k = (1+1+1)\lambda_i^k = 3$;
5. Nesses casos, na restrição de (3, 0.5)-SRC do mestre restrito.

Logo, a coluna avaliada pelo *pricing* soma pelo menos $\lfloor 1.5 \rfloor$ no lado esquerdo da restrição de (3, 0.5)-SRC (4-3), garantindo que essa coluna contribui para uma possível violação dessa restrição. Portanto, a inclusão dessa coluna gerada pelo *pricing* no mestre restrito causa a ativação dessa desigualdade no respectivo subproblema de mochila.

Em situação de ativação da desigualdade v, a variável z_v da mochila assume valor 1 já que as duais de v precisam ser somadas ao custo reduzido (função objetivo). É necessário que os subproblemas de *pricing* incorporem a ativação dessas duais v_v, conhecidas as condições em que isso se verifica.

Nesta dissertação, o *pricing* está resolvendo as mochilas com programação inteira. Embora exista algoritmos muito eficientes de programação dinâmica para problemas de mochila, como o algoritmo proposto em Pisinger (1997), esses algoritmos não supõem a existência de cortes (3, 0.5)-SRC.

A formulação do subproblema de mochila do *pricing* é reescrita, considerando as duais υ_v para a minimização do custo reduzido, conduzindo à equação (4-6).

$$Z^i_{pricing} = \max\left(\sum_{j=1}^{n}\left(\mu_j - c_{ij}\right)y_j\right) + \sum_{v=1}^{|S|}\upsilon_v z_v \tag{4-6}$$

Assim sendo, dado que a condição para habilitar a dual υ_v é que pelo menos duas tarefas da (3, 0.5)-SRC sejam atribuídas a uma mesma máquina i, algumas restrições são criadas para o subproblema. Formalmente, considere uma desigualdade (3, 0.5)-SRC composta por tarefas $T = j_1, j_2, j_3$. Logo, a ativação da variável de corte z_v é verificada para as alocações de tarefas (j_1, j_2), (j_1. j_3), (j_2, j_3), (j_1, j_2, j_3) pela respectiva mochila referente à máquina i.

Essas novas restrições, descritas em (4-7), definem o conjunto w_v com os índices de três tarefas distintas entre si e asseguram que as duais υ_v são somadas apenas quando a desigualdade v é ativada (contribui para possível violação da respective restrição do mestre restrito).

$$\begin{aligned}
&z_v \geq x_{iw^1_v} + x_{iw^2_v} - 1\\
&z_v \geq x_{iw^1_v} + x_{iw^3_v} - 1\\
&z_v \geq x_{iw^2_v} + x_{iw^3_v} - 1\\
&z_v \in \{0,\ 1\}\\
&w_v \subset \{1,\ \ldots,\ n\}\ \mid\ |w_v| = 3
\end{aligned} \tag{4-7}$$

De acordo com essas restrições criadas para desigualdades (3, 0.5)-SRC na mochila, qualquer alocação de ativação prevista para v, representada pelas variáveis y_j, força que z_v assuma valor 1. Isso pode ser verificado, considerando as alocações de ativação para tarefas $T = j_1, j_2, j_3$ e as variáveis y_j.

1. Se as tarefas $\{j_1, j_2\}$ estão atribuídas à máquina i, então $y_{j_1} = 1$, $y_{j_2} = 1$, $y_{j_3} = 0$ e $z_v = 1$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 2 - 1 = 1$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 1 - 1 = 0$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 1 - 1 = 0$

2. Se as tarefas $\{j_1, j_3\}$ estão atribuídas à máquina i, então $y_{j_1} = 1$, $y_{j_2} = 0$, $y_{j_3} = 1$ e $z_v = 1$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 1 - 1 = 0$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 2 - 1 = 1$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 1 - 1 = 0$

3. Se as tarefas $\{j_2, j_3\}$ estão atribuídas à máquina i, então $y_{j_1} = 0$, $y_{j_2} = 1$, $y_{j_3} = 1$ e $z_v = 1$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 1 - 1 = 0$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 1 - 1 = 0$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 2 - 1 = 1$

4. Se as tarefas $\{j_1, j_2, j_3\}$ estão atribuídas à máquina i, então $y_{j_1} = 1$, $y_{j_2} = 1$, $y_{j_3} = 1$ e $z_v = 1$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 2 - 1 = 1$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 2 - 1 = 1$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 2 - 1 = 1$

5. Se as tarefas $\{j_1\}$ estão atribuídas à máquina i, então $y_{j_1} = 1$, $y_{j_2} = 0$, $y_{j_3} = 0$ e $z_v = 0$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 1 - 1 = 0$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 1 - 1 = 0$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 0 - 1 = -1$

6. Se as tarefas $\{j_2\}$ estão atribuídas à máquina i, então $y_{j_1} = 0$, $y_{j_2} = 1$, $y_{j_3} = 0$ e $z_v = 0$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 1 - 1 = 0$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 0 - 1 = -1$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 1 - 1 = 0$

7. Se as tarefas $\{j_3\}$ estão atribuídas à máquina i, então $y_{j_1} = 0$, $y_{j_2} = 0$, $y_{j_3} = 1$ e $z_v = 0$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 0 - 1 = -1$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 1 - 1 = 0$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 1 - 1 = 0$

8. Se as tarefas $\{\ \}$ estão atribuídas à máquina i, então $y_{j_1} = 0$, $y_{j_2} = 0$, $y_{j_3} = 0$ e $z_v = 0$:

   (a) $z_v \geq y_{j_1} + y_{j_2} - 1 = 0 - 1 = -1$

   (b) $z_v \geq y_{j_1} + y_{j_3} - 1 = 0 - 1 = -1$

   (c) $z_v \geq y_{j_2} + y_{j_3} - 1 = 0 - 1 = -1$

Essa demonstração enumerativa revela que a variável z_v assume valor 1 se e somente se alguma das alocações de ativação da (3, 0.5)-SRC ocorre no subproblema de mochila do *pricing*. Portanto, essas restrições atendem suficientemente à especificação de introdução de desigualdades, ganrantindo o *pricing* correto para a geração de colunas com as (3, 0.5)-SRCs.

Recorrendo à formulação de restrições (4-7), adotando a função objetivo (4-6) e definindo o conjunto w_v de três tarefas distintas (sem repetição), os subproblemas de mochila são reescritos com a formulação abaixo em (4-8).

$$
\begin{aligned}
Z^i_{pricing} = \max & \left(\sum_{j=1}^{n} (\mu_j - c_{ij}) y_j \right) + \sum_{v=1}^{|S|} \upsilon_v z_v \\
& s.\,t. \\
& \sum_{j=1}^{n} w_{ij}\, y_j \leq b_i \\
& z_v \geq x_{iw_v^1} + x_{iw_v^2} - 1 && \forall \quad v \in S \\
& z_v \geq x_{iw_v^1} + x_{iw_v^3} - 1 && \forall \quad v \in S \\
& z_v \geq x_{iw_v^2} + x_{iw_v^3} - 1 && \forall \quad v \in S \\
& z_v \in \{0,\ 1\} && \forall \quad v \in S \\
& w_v \subset \{1,\ \ldots,\ n\} \ \mid\ |w_v| = 3 && \forall \quad v \in S \\
& y_j \in \{0,\ 1\} && \forall \quad j = 1, \ldots,\ n
\end{aligned}
\tag{4-8}
$$

4.3 Programação Dinâmica para Pricing com (3, 0.5)-SRCs

O *pricing* da geração de colunas depende da resolução de subproblemas de mochila definidos a partir da decomposição de Dantzig-Wolfe. Esses subproblemas podem ser resolvidos através de programação inteira, conforme dissertado anteriormente. Embora a programação inteira resolva problemas de mochila 0-1, há ainda outros algoritmos na literatura capazes de resolver esses problemas com

um excelente desempenho comparável ao estado-da-arte. Esta dissertação recorre inicialmente à clássica programação dinâmica como alternativa à programação inteira.

Em consideração ao *pricing* da geração de colunas, os subproblemas de mochila podem ser solucionados pela programação dinâmica diretamente a partir da função objetivo da formulação da mochila pela expressão de custo reduzido das colunas já definida na Equação 3-11. Neste caso, a função objetivo da mochila i depende somente dos custos c_{ij} de atribuição de tarefa j à máquina i e das duais μ_j das restrições de atribuição de tarefa j, sendo que esses custos e duais possuem valor constante para cada atribuição (j, i).

Entretanto, a separação de desigualdades (3, 0.5)-SRCs introduz as duais υ_v das restrições dos cortes definidos no modelo mestre da decomposição. Conforme dissertado na Seção 3.3, a dual υ_v é ativada, alterando o valor da função objetivo, se e somente se ocorre pelo menos duas atribuições de tarefas j da (3, 0.5)-SRC à máquina i. A formulação de programação inteira para os subproblemas de mochila foram adaptadas em (4-8) para incluir as (3, 0.5)-SRCs através da adição de restrições específicas para esse propósito.

Em contrapartida, a programação dinâmica não assume as restrições da programação inteira e precisa de uma estratégia diferente para incluir as (3, 0.5)-SRCs. Esta dissertação propõe uma nova versão da programação dinâmica (*labeling*) que considera as duais υ_v de cortes v sempre que pelo menos duas tarefas $\{j_1, j_2\}$ da respectiva desigualdade v são adicionadas à mochila i.

O algoritmo desenvolvido assume desigualdades $v \in S$ com duais υ_v e adota um fluxo semelhante à programação dinâmica padrão, percorrendo todas as n tarefas j segundo uma ordenação J e construindo uma árvore de *labels* limitada pela restrição de capacidade máxima da maquina i, um nível da árvore para a atribuição de cada tarefa j. O *label* é uma possível atribuição de tarefas j já visitadas à máquina i, armazenando o custo total das respectivas atribuições. Esse custo total reflete um valor de função objetivo do problema de mochila.

Em cada nível t da árvore, a inserção da tarefa j deste nível é avaliada individualmente para cada *label* k do nível anterior $t-1$, segundo os critérios discriminados abaixo.

1. Existem duas possibilidades de ramificação do *label* k, podendo ocorrer a criação de um *label* $\bar{k}$ que herda tarefas de k tanto pela adição quanto pela não adição da tarefa $j = J[t]$ do nível t;

2. Considere que um *label* k representa a atribuição de um subconjunto de tarefas j já visitadas pela programação dinâmica;

3. N_k é o conjunto de tarefas j alocadas à máquina i no *label* k;

4. OPT é um mapa que organiza *labels* k por tarefa j (nível t) e peso da *label* k;

5. Considere $l_k = \sum_{j \in N_k} w_{ij}$ como a capacidade ocupada pelas tarefas do *label* k;

6. Considere que o somatório dos custos implicados por todas as tarefas j atribuídas à máquina i pelo *label* k, onde p_j é o custo da atribuição da tarefa j na formulação clássica de mochila e $\sum_{j \in N_k} p_j$ representa o custo da *label* k;

7. Não ocorre a inserção da tarefa j no filho $\bar{k}$ do *label* k caso a tarefa j tenha peso w_{ij} tal que $l_k + w_{ij} > b_i$, onde b_i é a capacidade máxima da máquina i;

8. É permitida a inserção da tarefa j no *label* filho $\bar{k}$ se e somente se essa inserção não exceder a capacidade máxima b_i da máquina e ainda não existir um *label* no nível t ocupando a mesma capacidade $l_k + w_{ij}$ da mochila com um valor de solução igual ou maior que $\sum_{g \in N_k} p_g + p_j$;

9. Caso a tarefa j seja inserida na mochila, o novo *label* $\bar{k}$ terá solução com valor $\sum_{g \in N_{\bar{k}}} p_g = \sum_{g \in N_k} p_g + p_j$ e peso $l_{\bar{k}} = l_k + w_{ij}$;

10. Caso a tarefa j não seja inserida na mochila, o novo *label* $\bar{k}$ terá solução com valor $\sum_{g \in N_k} p_g$ e peso l_k;

Desta forma, o algoritmo constrói uma árvore binária de *labels* com uma solução ótima para cada valor de capacidade menor que b_i em cada nível t até visitar a última tarefa ou até que mais nenhuma tarefa caiba nas *labels* do último nível gerado $t-1$. Embora funcione bem para mochilas cuja função objetivo depende de valores constantes c_{ij} e μ_{ij}, esse algoritmo desconhece a regra de ativação das duais das (3, 0.5)-SRCs e, caso não haja uma adaptação adequada, falha no *pricing* do método com separação de desigualdades (3, 0.5)-SRC proposto nesta dissertação.

Em razão disso, a adaptação desenvolvida mantém uma memória das (3, 0.5)-SRCs ativadas em cada *label* k e ordena a visitação das tarefas j segundo uma ordem J crescente de *score* de eficiência e_{ij}, definido na Equação 4-9, referente às duais υ_v implicadas por cada tarefa em cada momento da programação dinâmica.

Esse *score* reflete o quanto a inserção de uma tarefa j pode penalizar o valor da solução da mochila, por unidade de peso w_{ij}, em dado nível t da árvore. Esse critério de ordenação é inspirado pela ordenação de itens de mochila no algoritmo de Pisinger (1997) que revela um desempenho comparável ao estado-da-arte para mochilas.

$$e_{ij} = \frac{\left(c_{ij} - \mu_j\right) - \sum_{v \in S_j} \upsilon_v \,/\, 3}{w_{ij}} \tag{4-9}$$

Neste caso específico, um *score* de eficiência e_{ij} considera todas as duais das restrições de desigualdades (3, 0.5)-SRC que são definidas, em parte, pela tarefa j. Dado que cada (3, 0.5)-SRC é definida por 3 linhas (referentes a 3 tarefas distintas), a medida e_{ij} considera a contribuição da tarefa j para a ativação das (3, 0.5)-SRC e, por isso, o valor das duais de (3, 0.5)-SRCs é dividido por três. O algoritmo de ordenação de tarefas está descrito em 6.

***Script* 6** Ordenação de Tarefas para Mochila com (3, 0.5)-SRCs

Data: conjunto T de todas as tarefas j atribuível às máquinas i

Result: vetor J com tarefas $j \in T$ ordenadas em ordem crescente de eficiência para nível da árvore de *labeling*.

- Inicializa as eficiências e_{ij}, considerando todas as (3, 0.5)-SRCs para tarefas $j \in T$;
- Ordena todas as tarefas $j \in T$ em ordem crescente de eficiência e_{ij};
- Inicializa o nível da árvore de programação dinâmica: $t = 1$;
- Declara vetor (vazio) de ordenação J das tarefas por nível da árvore;

repeat

- Remove a tarefa menos eficiente j (menor e_{ij}) do conjunto T;
- Insere a tarefa removida j no nível $J[t]$ da árvore;
- Atualiza as eficiências e_{ij} de todas as tarefas $j \in T$:
  - ☐ Identifica S_j, conjunto de (3, 0.5)-SRCs **ativadas** pela inserção da tarefa j no nível t da árvore;
  - ☐ $e_{if} = e_{if} + \dfrac{\sum_{v \in S_j} \left(v_v \,/\, 3\right)}{w_{ij}} \quad \forall\, f \in T$
- Ordena todas as tarefas $j \in T$ em ordem crescente de eficiência e_{ij};
- Atualiza o nível corrente da árvore: $t = t + 1$;

until $T \neq \varnothing$;

- **Retorna** J

Em vista disso, algumas considerações são ressaltadas sobre a utilização da memória de (3, 0.5)-SRCs na programação dinâmica e sobre a ordenação das tarefas segundo *scores* de eficiência.

1. A ordenação J permite priorizar a inserção das tarefas que mais penalizam a solução da mochila;
2. A eficiência e_{ij} de cada tarefa j é inicializada considerando todas as (3, 0.5)-SRCs relacionadas;

3. Durante a ordenação, existe um mapa que associa cada (3, 0.5)-SRC a um contador de tarefas relacionadas já ordenadas, servindo para atualizar devidamente as eficiências e_{ij} com base nas tarefas já ordenadas e ainda não ordenadas (7);

4. A posição da tarefa j na sequência J pode alterar o *score* das tarefas posteriores a j, de modo que se pelo menos duas tarefas de uma (3, 0.5)-SRC já tiverem sido posicionadas em J, então essa desigualdade já está ativada ou jamais será ativada e a sua dual υ_v deve ser removida do cálculo de e_{if} de toda tarefa f ainda não posicionada em J;

5. Durante a programação dinâmica, todo *label* carrega um mapa E atualizado que associa cada (3, 0.5)-SRC v existente a dois contadores, um contador de tarefas adicionadas e outro de tarefas ignoradas, onde cada contador acumula exclusivamente tarefas relacionadas ao respectivo corte v, permitindo ativar v e contabilizar a respectiva dual υ_v na solução do *label*, conforme algoritmo 7;

***Script* 7** Ativação de (3, 0.5)-SRCs na Programação Dinâmica (Mochila)

Data: mapa E de memória das (3, 0.5)-SRCs; tarefa j sendo inserida ou ignorada; booleano z, 1 se inserção, caso contrário 0; máquina i.

Result: mapa E atualizado e conjunto I de cortes ativados/ignorados

- Acessa T_{SRC} que associa cada tarefa ao conjunto de (3, 0.5)-SRCs relacionadas;
- Declara o conjunto I de novas (3, 0.5)-SRCs ativadas ou ignoradas pela tarefa j;

for *each cut* $v \in T_{SRC}[j]$ **do**
    - Atualiza a entrada do mapa: $E[v][z] = E[v][z] + 1$
    if $E[v][z] == 2$ **then**
        - Insere v no conjunto I;

- **Retorna** $\{E, I\}$

Durante o algoritmo de programação dinâmica, sempre que uma tarefa é adicionada ou ignorada na criação de um novo *label*, o contador de v no mapa do *label* é atualizado e verificado, conforme algoritmo 7; e há ativação do corte v se e somente se o contador de tarefas atribuídas de v for incrementado para a 2 pela primeira vez. Logo, pelo menos duas tarefas do corte v foram identificadas na solução representada por este *label*. Então, a dual υ_v é ativada na solução deste *label*. Da mesma forma, se o contador de tarefas de v ignoradas pelo *label* for incrementado para 2, então o corte v nunca será ativado na solução desse *label* e pode ser desconsiderado.

***Script* 8** Programação Dinâmica para Mochila com (3, 0.5)-SRCs (PAG)

Data: conjunto de tarefas T; máquina i

Result: mapa OPT com as soluções da mochila para a máquina i e tarefas T

- Ordena todas as tarefas $j \in T$ para a máquina i, segundo as eficiências e_{ij};
- Define uma ordenação J de tarefas, uma para cada nível da árvore de *labeling*;
- Declara o mapa OPT para armazenar *labels* para cada nível da árvore;
- Inicializa o *label* raíz no nível 0 da árvore;

for $t = 1$; $t \leq |J|$; $t = t + 1$ **do**

- Reconhece a tarefa corrente $J[t]$;
- **for** $y = 0$; $y \leq OPT[t-1]$; $y = y + 1$ **do**
  - Acessa o *label* k em $OPT[t-1][y]$;
  - **if** *label* x ***existe*** *no nível* $OPT[t]$ *com peso* $\sum_{j \in N_k} w_{ij}$ *da label* k **then**
    - **if** $\sum_{j \in N_k} p_j + SRC(k) > \sum_{j \in N_x} p_j + SRC(x)$ **then**
      - Cria *label* $\bar{k}$ no nível $OPT[t]$ com ($\sum_{j \in N_k} p_j + SRC(k)$, $\sum_{j \in N_k} w_{ij}$);
      - Deleta o *label* x;
      - Atualiza devidamente a memória (3, 0.5)-SRCs dos *labels* $\bar{k}$ e k;
  - **else**
    - Cria *label* $\bar{k}$ no nível $OPT[t]$ com ($\sum_{j \in N_k} p_j + SRC(k)$, $\sum_{j \in N_k} w_{ij}$);
    - Atualiza devidamente a memória de (3, 0.5)-SRCs dos *labels* $\bar{k}$ e k;
  - **if** $\sum_{j \in N_k} w_{ij} + w_{iJ[t]} > b_i$ **then**
    - **continue**;
  - Localiza (3, 0.5)-SRCs ativadas pela inserção da tarefa $J[t]$ no *label* k;
  - Computa a soma $s_k^{J[t]}$ das duais de restrições das (3, 0.5)-SRCs ativadas;
  - **if** *label* x ***existe*** *no nível* $OPT[t]$ *com peso* $\sum_{j \in N_k} w_{ij} + w_{iJ[t]}$ *do label* k **then**
    - **if** $\sum_{j \in N_k} p_j + SRC(k) + p_{J[t]} + s_k^{J[t]} > \sum_{j \in N_x} p_j + SRC(x)$ **then**
      - Cria *label* $\bar{k}$ com ($\sum_{j \in N_k} p_j + SRC(k) + p_{J[t]} + s_k^{J[t]}$, $\sum_{j \in N_k} w_{ij} + w_{iJ[t]}$);
      - Insere $\bar{k}$ no nível $OPT[t]$ e deleta o *label* x;
      - Atualiza devidamente a memória (3, 0.5)-SRCs dos *labels* $\bar{k}$ e k;
  - **else**
    - Cria *label* $\bar{k}$ com ($\sum_{j \in N_k} p_j + SRC(k) + p_{J[t]} + s_k^{J[t]}$, $\sum_{j \in N_k} w_{ij} + w_{iJ[t]}$);
    - Insere $\bar{k}$ no nível $OPT[t]$;
    - Atualiza devidamente a memória (3, 0.5)-SRCs dos *labels* $\bar{k}$ e k;

- **Retorna** OPT;

O algoritmo 8 de programação dinâmica adaptado às (3, 0.5)-SRCs funciona para os subproblemas de mochila, segundo experimentos realizados para algumas instâncias do PAG.

Em conformidade com os experimentos computacionais realizados, a geração de colunas com *pricing* resolvido por essa programação dinâmica obteve os mesmos resultados encontrados pela programação inteira para os subproblemas de mochila.

Entretanto, uma redução significativa do tempo de execução não foi observada. Logo, o método *price-and-cut* implementado continuou utilizando programação inteira para o *pricing*.

Todavia, a comprovação de que subproblemas de mochila com (3, 0.5)-SRCs podem ser resolvidos com programação dinâmica é uma importante contribuição desta dissertação à medida que incentiva uma pesquisa futura mais aprofundada para otimização do *price-and-cut* e da enumeração de colunas.

É relevante considerar a pesquisa de uma possível extensão da adaptação proposta a outros algoritmos de programação dinâmica. A pertinência dessa observação é dada pelo fato de existir pelo menos um algoritmo mais eficiente que a clássica programação dinâmica na literatura, o algoritmo de Pisinger *MinKnap*.

O artigo Pisinger (1997) propõe um algoritmo de programação dinâmica para resolver mochilas de formulação inteira, utilizando uma redução por *bounds* para adicionar ou remover tarefas de uma solução *core* encontrada gulosamente como ponto de partida.

Esse algoritmo apresenta uma eficiência significativamente melhor que a programação dinâmica clássica para problemas de mochila, tendo sido testado neste trabalho com a geração de colunas, atingindo os mesmos resultados com uma maior eficiência na resolução de mochilas. Entretanto, o algoritmo de Pisinger também não considera a existência de cortes na formulação da mochila.

Esta dissertação não desenvolve uma adaptação do algoritmo de Pisinger para incluir as (3, 0.5)-SRCs, dado que a redução por *bounds* torna a adaptação mais difícil. Ainda assim, a adaptação da programação dinâmica para mochilas é uma primeira tentativa e abordagem para uma possível adaptação de outros algoritmos mais eficientes.

5
Algoritmo de Enumeração de Colunas

Em condições gerais, a aplicabilidade de métodos enumerativos é muito limitada computacionalmente devido à natureza combinatória desta abordagem, sobretudo para problemas com um número de variáveis de decisão que cresce exponencialmente em função do tamanho das instâncias.

O Problema de Alocação Generalizada (PAG) não é uma exceção, apresentando uma quantidade exponencial de configurações para atribuir n tarefas a m máquinas. Então, a enumeração de colunas tende a ter uma complexidade exponencial, sendo computacionalmente impraticável no cenário em que são percorridas todas as combinações de atribuição de tarefa j à máquina i.

Entretanto, um algoritmo enumerativo que se mostre capaz de percorrer explicitamente apenas uma fração bem selecionada de colunas pode ser perfeitamente viável; ainda melhor se a enumeração estiver restrita a um reduzido espaço de busca formado somente pelas colunas comprovadamente necessárias para encontrar a solução ótima. Essa garantia depende da capacidade de identificar as colunas que não têm potencial para compor uma base promissora.

Essa estratégia é a motivação principal desta dissertação, sendo a enumeração de colunas a finalidade a qual serve todo o processo de decomposição, geração de colunas e separação de desigualdades (3, 0.5)-SRCs. A decomposição de Dantzig-Wolfe é uma forma de discretizar as restrições de capacidade de máquina em colunas λ_i^k, alcançando uma formulação mais simples para o PAG. A relaxação linear da formulação de Dantzig-Wolfe para o PAG pode ser resolvida pela geração de colunas (*pricing*), gerando uma base de colunas geradas e uma solução linear ótima que serve de LB. Esse LB pode ser aumentado ainda mais com a separação das (3, 0.5)-SRCs alternada com a geração de colunas para gerar bases de colunas e soluções duais que asseguram que o custo reduzido de toda coluna do modelo de decomposição é no máximo nulo.

5.1
Definição do Contexto de Enumeração

O algoritmo de enumeração de colunas, descrito em 10, é a finalidade do *price-and-cut* e a última parte do método proposto para encontrar soluções

inteiras para o PAG que sejam comprovadamente ótimas ou bastante próximas do ótimo. Esse algoritmo utiliza o *gap* determinado pelo *price-and-cut* para enumerar explicitamente um subconjunto reduzido de colunas. É trivial determinar um *gap* $d = UB - LB$ a partir de um bom LB encontrado pelo *price-and-cut* e os melhores valores de UB conhecidos na literatura para instâncias de PAG, nomeadamente os UB reportados por Avella (2010) e Michelon (2010).

Neste caso, a geração de colunas com desigualdades (3, 0.5)-SRC serve exatamente ao propósito de maximizar o LB (solução linear do modelo mestre restrito), equivalentemente funcionando como um minimizador de *gap* para instâncias do PAG. O *gap* mínimo d acompanhado da base de colunas E_0, geradas para o mestre restrito da decomposição, potencialmente viabiliza a enumeração de colunas E. As colunas enumeráveis $\bar{k} \in \bar{E}$ são todas aquelas que possuem custo reduzido $\bar{r}_k \leq d$ e que ainda não estão na base de colunas E_0 geradas do mestre restrito relaxado do *price-and-cut*, onde $\bar{E} \subset E$.

Em decorrência disso, as colunas enumeráveis são aquelas que fecham o *gap*, provando a otimalidade de uma solução inteira. Isso ocorre porque a enumeração de colunas complementa a base de colunas geradas, formando uma base de colunas com todas as colunas que têm potencial para compor uma solução ótima para o modelo de Dantzig-Wolfe da instância do PAG, alcançando, assim, uma solução inteira viável. Essa solução inteira tem otimalidade garantida se e somente se a enumeração de colunas tiver sido exaustiva.

- O conjunto E_0 contém todas as colunas geradas pelo *price-and-cut* (*pricing* e desigualdades (3, 0.5)-SRC);
- O conjunto E contém todas as colunas encontradas pela enumeração;
- O conjunto $\bar{E}$ contém todas as colunas $\bar{k}$ encontradas pela enumeração e que não foram geradas pelo *price-and-cut* ($\bar{k} \notin E_0$);
- A solução da relaxação linear do mestre restrito é o LB de referência para definir o *gap* usado pela enumeração de colunas;
- O melhor UB de referência utilizado para a enumeração de colunas é a melhor solução inteira encontrada na literatura por Avella (2010) e Michelon (2010);
- O conjunto de colunas $\bar{k} \in \bar{E}$ é um subconjunto de colunas enumeráveis com custo reduzido $\bar{r}_k \leq UB - LB$;
- O *gap* mínimo determinado para a enumeração é computado como $d = UB - LB$;
- O conjunto de colunas E é uma base de colunas restrita com potencial para encontrar solução ótima para o modelo de decomposição (PAG).

O algoritmo *price-and-cut* produz um *gap* reduzido o suficiente para que a enumeração percorra um número restrito de colunas, considerando apenas aquelas cujo custo reduzido é menor que esse *gap* encontrado. Quanto menor for o *gap*, menor a quantidade de colunas enumeráveis do subconjunto $\bar{E}$ e mais eficiente e eficaz será a enumeração. Em razão disso, o *price-and-cut* é adotado para maximizar o LB no modelo mestre restrito da formulação de Dantzig-Wolfe para o PAG. A partir da base de colunas geradas E_0 e do respectivo modelo mestre restrito otimizado e suas duais, o custo reduzido das colunas enumeradas pode ser computado, seguindo a Equação 5-1, onde N_k é o conjunto de tarefas atribuídas à máquina i pela coluna k e υ_v é a dual da restrição do corte v que pertence ao conjunto S_k de (3, 0.5)-SRCs ativadas pelas tarefas $j \in N_k$. As colunas que excedem o *gap* não são explicitamente enumeradas no conjunto E, assim sendo descartadas durante a enumeração.

$$\bar{r}_k = \sum_{j=1}^{N_k} \big(c_{ij} - \mu_j\big)x_{ij} \; - \; \pi_i \; - \sum_{v \in S_k} \upsilon_v \, z_v \tag{5-1}$$

O desafio da enumeração de colunas é definir uma estratégia eficaz e eficiente para considerar implicitamente todas as colunas, enumerando apenas aquelas com custo reduzido $\bar{r}_k \leq d$, onde d é o *gap* de referência, ou seja, colunas com custo reduzido maior que o *gap* não são adicionadas à base E. Em conformidade com o critério definido em (5-2), as colunas visitadas na árvore de enumeração constituem um subconjunto E reduzido de colunas e a avaliação das colunas enumeráveis está sujeita ao custo reduzido $\bar{r}_k$. O tamanho do *gap* influencia a capacidade da enumeração de colunas de encontrar todas as colunas com potencial para pertencer à solução ótima.

$$\bar{r}_k \leq UB - LB \tag{5-2}$$

5.2 Algoritmo de Enumeração de Colunas para o PAG

Durante o procedimento de enumeração, as colunas são iterativamente construídas ao longo da árvore de enumeração exemplificada na Figura 5.1. Essa árvore é desenvolvida para cada região convexa das restrições de máquina, logo uma enumeração de colunas para cada máquina i da instância. O custo reduzido das colunas enumeradas é calculado através do problema de mochila com a mesma formulação dos subproblemas de *pricing* da geração de colunas.

Essa árvore de enumeração é construída adicionando tarefas iterativamente, uma tarefa a cada nível da árvore. O nó raíz começa com custo reduzido $\bar{r}_k = \pi_i$, onde π_i é a dual de restrição da máquina i que é constante dado que cada árvore de

enumeração corresponde a uma máquina i fixa. Em cada nó da árvore, a respectiva coluna representa uma possível composição de tarefas j atribuídas à máquina i.

O método tenta dividir cada nó em dois nós, um que insere a tarefa $H[h]$ na coluna do nó e outro que ignora essa tarefa, considerando que o nó pai pertence ao nível h da árvore e H é uma ordenação de tarefas. Portanto, o algoritmo de enumeração realiza uma tentativa de adicionar ou ignorar a tarefa $H[h]$ fixa para cada nível h da árvore.

Em termos práticos de implementação, dado que a árvore é construída nível a nível, o algoritmo apenas guarda os nós sem filhos do último nível construído (iteração h). Esses nós são as folhas da árvore ilustrada na Figura 5.1. O custo reduzido é calculado para cada nó criado na árvore, fixando todas as tarefas já adicionadas e todas as tarefas já ignoradas da coluna correspondente a esse nó. Então, quanto maior o índice do último nível construído, maior o número de variáveis fixadas nos modelos MIP para computação do custo reduzido das colunas. Assim sendo, o nó raíz começa com um custo reduzido igual à dual π_i da restrição de máquina i e esse valor é atualizado a cada nível da árvore.

A altura da árvore de enumeração cresce iterativamente e o número de nós tende a aumentar exponencialmente. Esse aumento do número de colunas enumeradas precisa ser evitado, utilizando o *gap* d determinado para podar a árvore de enumeração, descartando nós cujas colunas jamais atingirão um custo reduzido menor ou igual ao *gap* d.

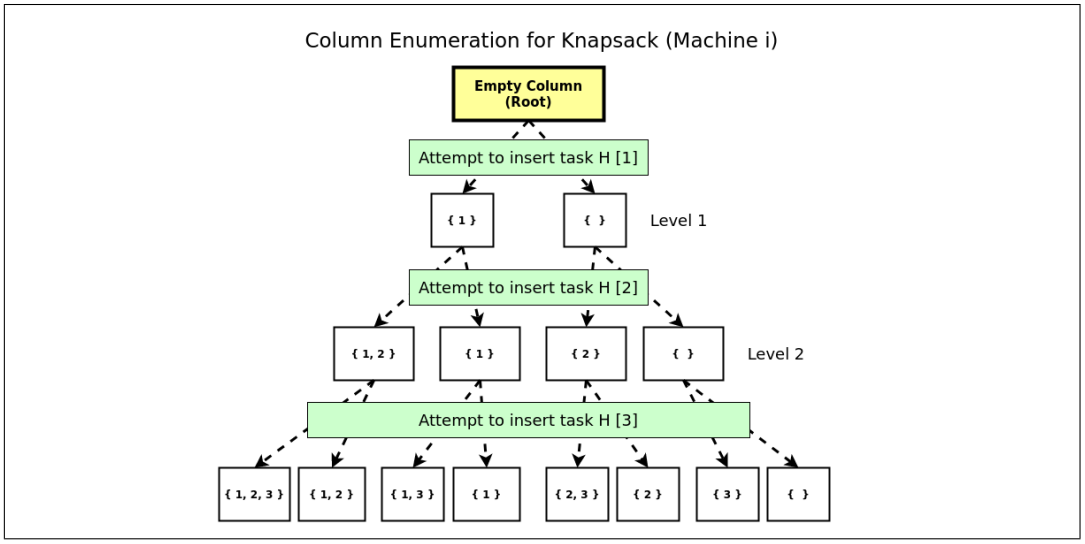

Figura 5.1: Expansão da árvore de enumeração de colunas para instâncias do Problema de Alocação Generalizada (PAG), ilustrando 3 níveis. Todas as colunas enumeradas estão nas folhas da árvore.

Essa redução da árvore supõe que, em qualquer nó e nível, o algoritmo seja capaz de computar o custo reduzido mínimo das colunas de nós descendentes. Em outras palavras, o método computa o menor custo reduzido que pode ser

atingido pelas colunas descendentes, fixando as tarefas da coluna do nó ancestral e resolvendo a mochila com a atribuição das tarefas dos níveis seguintes ainda não construídos. A computação desse custo reduzido mínimo para um dado nó da árvore é um clássico problema de minimização semelhante ao *pricing* da geração de colunas.

Esse processo de enumeração de colunas apresenta muitas semelhanças com a programação dinâmica da mochila com (3, 0.5)-SRCs, abordado na Seção 4.3. Entretanto, particularmente neste caso, apenas os nós sem filhos são guardados pelo algoritmo; a árvore inteira não é guardada; e o custo reduzido mínimo permite reduzir o número de nós visitados, não sendo necessário enumerar explicitamente todas as colunas possíveis para completar uma enumeração exaustiva. Em vista disso, a enumeração de colunas recorre aos mesmos algoritmos de ordenação de tarefas e de ativação de (3, 0.5)-SRCs que foram definidos para a programação dinâmica de mochilas com (3, 0.5)-SRCs.

O algoritmo resolve um problema de mochila com fixação de tarefas, como formulado em (5-3), para determinar o custo reduzido $\bar{r}_k$ mínimo existente para as ramificações que saem de dado nó da árvore.

$$
\begin{aligned}
Z^i_{node} = \max & \left(\sum_{j=1}^{n} \left(\mu_j - c_{ij}\right) x_{ij} \right) + \sum_{v=1}^{|S|} \upsilon_v z_v \\
& s.\,t. \\
& \sum_{j=1}^{n} w_{ij}\, x_{ij} \leq b_i \\
& z_v \geq x_{iw_v^1} + x_{iw_v^2} - 1 && \forall \quad v \in S \\
& z_v \geq x_{iw_v^1} + x_{iw_v^3} - 1 && \forall \quad v \in S \\
& z_v \geq x_{iw_v^2} + x_{iw_v^3} - 1 && \forall \quad v \in S \\
& z_v \in \{0,\ 1\} && \forall \quad v \in S \\
& w_v \subset \{1,\ \ldots,\ m\} \ \mid\ |w_v| = 3 && \forall \quad v \in S \\
& H_i^1 \ \cup \ H_i^0 \ = \ \{1, \ldots, n\} \\
& H_i^1 \ \cap \ H_i^0 \ = \ \varnothing \\
& x_{ij} = 1 && \forall \quad j \ \in \ H_i^1 \\
& x_{ij} \in \{0,\ 1\} && \forall \quad j \ \in \ H_i^0
\end{aligned}
\tag{5-3}
$$

– A árvore de enumeração dispõe de uma altura de tamanho mínimo 1 (nó raíz vazio) e tamanho máximo $n + 1$, onde n é o número de tarefas da instância do PAG;

- Em qualquer nível h da árvore de enumeração, a coluna de um nó qualquer desse nível possui no máximo h tarefas;
- Em qualquer nível h da árvore de enumeração, há dois conjuntos de tarefas H_i^0 e H_i^1 onde H_i^1 armazena tarefas já visitados pela árvore; H_i^0 armazena tarefas ainda não visitadas pela árvore; $H_0 \cup H_1 = H$ e $H_0 \cap H_1 = \varnothing$;
- O custo reduzido mínimo computado para o nó p no nível h é uma minimização do custo reduzido $\bar{r}_k$ com fixação de variáveis (valor 1) referentes às tarefas contidas em H_i^1;
- Considere uma ordenação de tarefas H_i para a máquina i, atribuindo exclusivamente cada tarefa j a um nível da árvore de enumeração, em ordem crescente de eficiência e_{ij} de modo que $H_i[1]$ é a tarefa do primeiro nível abaixo do nó raíz e $H_i[n]$ é a tarefa do último nível da árvore;
- A divisão do nó p no nível h da árvore de enumeração é uma decisão binária sobre a inclusão da tarefa $H_i[h]$ na coluna deste nó, sujeita à capacidade máxima b_i da máquina i dessa árvore de enumeração;
- Quando o custo reduzido mínimo computado para uma coluna é maior que o *gap* $UB - LB$ ou quando nenhuma nova tarefa de H_i^0 pode ser adicionada à coluna, o nó correspondente da árvore se torna uma folha definitiva;
- O custo reduzido mínimo de colunas com desigualdades (3, 0.5)-SRC é computado pela mesma estratégia adotada para o subproblema de mochila do *pricing*, seguindo a mesma formulação.

Essa minimização de $\bar{r}_k$ segue a formulação do *pricing*, admitindo a fixação de variáveis das tarefas $j \in H_i^1$. Caso o custo reduzido mínimo para um nó seja calculado com valor $Z_{node} > d$, maior que o *gap* d, então todas as ramificações derivadas desse nó conduzem a colunas com custo reduzido $\bar{r}_k > d$, maior que o *gap*.

Essa previsão por inspeção de *bound* mínimo é o diferencial que permite a interrupção de ramificações em diferentes nós da árvore de enumeração, reduzindo significativamente o tamanho da enumeração (o número de nós instanciados).

Em contrapartida, quantidade de vezes que o custo reduzido mínimo precisa ser computado é proporcional ao número de nós visitados. Em instâncias específicas para as quais a programação inteira da mochila (5-3) apresenta um alto custo computacional, essa inspeção de *bound* para custo reduzido mínimo pode ser computacionalmente custosa, tornando a enumeração lenta.

Em vista disso, a viabilidade da enumeração de colunas ainda pode ser custosa para algumas instâncias, a depender sobretudo e fundamentalmente do *gap* mínimo obtido.

Em situações de instâncias com muitas tarefas ou com muitas máquinas, a enumeração de colunas ainda pode ser demasiadamente lenta, apesar de teoricamente finita. Então, a enumeração de colunas se torna muito difícil quando as causas listadas abaixo são verificadas..

1. O número de tarefas n é demasiadamente grande ($n > 400$);
2. O número de máquinas m é demasiadamente grande ($m > 20$);
3. A programação inteira para minimização do custo reduzido $\bar{r}_k$ é muito lenta, tornando a enumeração muito demorada;
4. O número de colunas enumeradas ainda é muito grande, excedendo 1 milhão de colunas, de modo que inviabiliza a programação inteira para o modelo de decomposição de instâncias do PAG. O *pricing* por inspeção pode ser implementado para viabilizar a programação inteira, neste caso;
5. O tamanho da árvore ainda é computacionalmente inviável, exigindo uma quantidade muito grande de memória (mais de 15 GB).

Em decorrência disso, quando qualquer uma das condições listadas é verificada, a enumeração de colunas implementada é interrompida prematuramente (não sendo exaustiva). Consequentemente, uma enumeração interrompida não compõe uma base de colunas suficiente para comprovar a otimalidade de uma eventual solução inteira determinada pelo modelo de Dantzig-Wolfe para o PAG.

Essas limitações são reguladas por alguns parâmetros fornecidos ao algoritmo, como o valor máximo de memória, o tempo máximo de enumeração por máquina e o número máximo de colunas enumeradas por máquina. A condição 3 ocorre para algumas instâncias PAG do tipo D da OR-Library, exigindo que a enumeração em cada máquina seja parada após um período de tempo limite. A condição 4 ocorre para algumas instâncias PAG do tipo D da OR-Library, nomeadamente as instâncias d10200 e d20200. A condição 5 é prevista para instâncias cujo *gap* ainda seja significativamente grande, tornando a redução da árvore de enumeração insuficiente. Eventualmente, uma gestão eficiente de memória, salvando parte das colunas enumeradas em disco, pode permitir contornar essa limitação do algoritmo. Entretanto esse tratamento de memória para colunas enumeradas não foi implementado nesta dissertação. Essa dissertação implementa heurísticas para identificar essas condições e interromper a enumeração prematuramente.

A ordenação H_i das tarefas é uma estratégia de otimização que visa analisar primeiramente as tarefas que mais reduzem o custo reduzido das colunas, minimizando o tamanho máximo atingido pela árvore. Isso ocorre porque o custo

reduzido das primeiras colunas enumeradas depende muito da ordem das tarefas visitadas pela árvore. Dado que o propósito do algoritmo é reduzir ao máximo o número de colunas enumeráveis ($\bar{E}$) e manter a altura da árvore a menor possível ao longo da enumeração, a pioridade é sempre visitar primeiro as tarefas que mais contribuem para o custo reduzido mínimo. Em virtude disso, a ordenação de tarefas é projetada para manter a árvore de enumeração o mais reduzida possível com base no *bound* de custo reduzido mínimo que permite não construir colunas que jamais alcançarão custo reduzido menor que o *gap* d. Desta forma, o custo reduzido mínimo serve de critério para podar ramificações da árvore de enumeração durante a execução do algoritmo.

O critério de ordenação escolhido é inspirado no algoritmo de programação dinâmica para *knapsack* proposto por Pisinger (1997) que resolve problemas de mochila 0-1 com eficiência comparável ao estado-da-arte. Esse algoritmo de mochila demonstra que existe uma ordenação de tarefas que garante uma solução inicial muito próxima da solução ótima, a partir da qual algumas remoções e inserção de tarefas conduzem à solução ótima, otimizando ainda mais com reduções feitas através de *bounds*.

Inicialmente, o algoritmo proposto por Pisinger ordena em O($n\ log\ n$) as tarefas em ordem decrescente de eficiência, lucro por unidade de peso, $c_j\ /\ w_j$ (guloso). Em seguida, percorre essas tarefas, somando os respectivos pesos w_j, enchendo a mochila até encontrar o primeiro item que não cabe mais. Essa primeira tarefa que não entra na mochila é chamada de tarefa de parada. A seleção inicial de tarefas obtida é a solução de núcleo, a solução gulosa que serve como solução inicial próxima da solução ótima.

Em comparação com o algoritmo de mochila 0-1 de Pisinger, a minimização do custo reduzido $\bar{r}_k$ é também uma mochila 0-1 com fixação de tarefas e lucro escrito em função do custo reduzido para a coluna. Então, a expressão de eficiência e_{ij} considerada para a ordenação das tarefas é uma expressão de custo reduzido da tarefa por unidade de peso da tarefa j na máquina i, conforme escrito na Equação 5-4.

$$e_{ij} = \frac{\left(c_{ij} - \mu_j\ \right)\ -\ \sum\limits_{v \in S_j} v_v\ /\ 3}{w_{ij}} \tag{5-4}$$

Em contapartida, as tarefas j são organizadas em ordem crescente de eficiência e_{ij} para a coluna k da máquina i porque as tarefas que mais reduzem o custo reduzido da coluna do nó são prioridade para atingir mais rapidamente um custo reduzido mínimo $\bar{r}_k\ \leq\ UB - LB$, mantendo a árvore menor ao longo do processo enumerativo.

No primeiro nível da árvore, nenhuma tarefa foi inserida ainda nas colunas. Então, a inserção da tarefa j diminuirá o custo reduzido em $c_{ij} - \mu_j - v$, onde v é o somatório das duais de restrição referentes a todas as (3, 0.5)-SRCs ativadas pela inserção da tarefa j. Nos níveis seguintes da árvore, cada inserção de tarefa diminui esse mesmo custo reduzido $c_{ij} - \mu_j - v$, lembrando sempre quais tarefas já foram visitadas pela enumeração e, consequentemente, as desigualdades (3, 0.5)-SRC que já não serão mais ativadas.

1. Quando a inserção de uma tarefa j no nível da árvore h é realizado e alguma (3, 0.5)-SRC é definida por j junto com pelo menos uma outra tarefa do conjunto H_i^1 (tarefas já visitadas pela enumeração), então essa desigualdade é ativada (caso ainda não tenha sido);

2. Quando pelo menos duas tarefas que definem uma (3, 0.5)-SRC já foram visitadas, essa desigualdade já está ativada ou não poderá mais ser ativada nas colunas enumeradas.

Logo, a ordenação considera que a escolha de posição de uma tarefa afeta a contribuição ao custo reduzido das demais tarefas ainda não ordenadas. Quando uma tarefa é posicionada na ordenação, verifica-se quais (3, 0.5)-SRCs são ativadas por essa decisão, atualizando a eficiência e_{ij} das tarefas que ainda aguardam ordenação e, assim, somando as duais dos cortes ativados à eficiência e_{ij}.

Considere o conjunto S_j de (3, 0.5)-SRCs que são ativadas pela inserção da tarefa j na ordenação. Caso a inserção da tarefa j no nível h da árvore ative as (3, 0.5)-SRCs do conjunto S_j, então as duais $v_v/3 \; \forall \; v \in S_j$ são somadas à eficiência de cada tarefa $f \in H_i^0$ ainda não ordenada (para níveis da árvore maiores que h), conforme a expressão de atualização abaixo na Equação 5-5.

Portanto, a ordenação organiza iterativamente as tarefas para cada nível da árvore em função da eficiência, selecionando a tarefa com menor e_{ij}, atualizando a eficiência e_{ij} das tarefas ainda não ordenadas e repetindo isso enquanto houver tarefas ainda não ordenadas. Assuma que H_i^0 é o conjunto de colunas ainda não ordenadas como previamente definido, sendo h a posição da última tarefa adicionada à ordenação pretendida.

$$e_{if} \; = \; e_{if} \; + \; \frac{\sum\limits_{v \, \in \, S_j} \left(v_v \; / \; 3 \right)}{w_{ij}} \qquad \forall \; f \; \in \; H_i^0 \tag{5-5}$$

A enumeração de colunas implementa uma política de ordenação de tarefas por nível da árvore que obedece o mesmo critério guloso, priorizando as tarefas que mais contribuem para a redução do custo reduzido da coluna, acelerando a convergência do custo reduzido mínimo já nas primeiras colunas enumeradas. Essa ordenação é realizada antes da enumeração de colunas, conforme algoritmo 9.

***Script* 9** Ordenação de Tarefas para Enumeração de Colunas (PAG)

Data: conjunto de tarefas T; máquina i

Result: vetor H_i com tarefas $j \in T$ ordenadas em ordem crescente de eficiência e_{ij} para os níveis da árvore de enumeração (máquina i).

- Constrói o modelo (MIP) de mochila (custo reduzido mínimo para a máquina i);
- Resolve o modelo (MIP) de mochila (i), computando o custo reduzido mínimo;
- Declara um conjunto T vazio para armazenar tarefas da solução ótima do MIP;

for $j \in$ *{tarefas atribuídas à máquina* i *na solução ótima do MIP}* **do**
    - Insere a tarefa j no conjunto T;

- Inicializa as eficiências e_{ij}, considerando todas (3, 0.5)-SRCs para tarefas $j \in T$;
- Ordena todas as tarefas $j \in T$ em ordem crescente de eficiência e_{ij};
- Inicializa o nível da árvore de enumeração $h = 1$;
- Define uma ordenação H_i de tarefas, uma para cada nível da árvore de enumeração;
- Declara vetor (vazio) de ordenação H_i das tarefas por nível da árvore;

repeat
    - Remove a tarefa menos eficiente j (menor e_{ij}) do conjunto T;
    - Insere a tarefa removida j no nível $H_i[h]$ da árvore;
    - Atualiza as eficiências e_{ij} de todas as tarefas $j \in T$:
        - ☐ Identifica S_j, conjunto de (3, 0.5)-SRCs **ativadas** pela inserção da tarefa j no nível h da árvore;
        - ☐ $e_{if} = e_{if} + \dfrac{\sum_{v \in S_j} (\upsilon_v \,/\, 3)}{w_{ij}} \quad \forall\, f \in T$

    if $T == \varnothing$ ***and*** $|H_i| < n$ **then**
        for $j \notin$ *{tarefas atribuídas à máquina* i *na solução ótima do MIP}* **do**
            - Insere a tarefa j no conjunto T;
        - Atualiza a eficiência e_{ij} de cada tarefa $j \in T$:

    - Ordena todas as tarefas $j \in T$ em ordem crescente de eficiência e_{ij};
    - Atualiza o nível corrente da árvore: $h = h + 1$;
until $T \neq \varnothing$;

- **Retorna** H

Em meio à enumeração de colunas, após a construção de um novo nível da árvore, algumas colunas dos nós (folhas) podem ter custo reduzido $\bar{r}_k$ nulos.

Esses nós carregam colunas já com custo reduzido mínimo, logo essas colunas não têm mais descendentes na árvore e podem ser ignoradas nas iterações seguintes da enumeração, sendo salvas na base de colunas E para resolver posteriormente o modelo de decomposição com programação inteira.

Em termos mais gerais, há três verificações que são feitas ao término de cada iteração de expansão da árvore (a cada nível construído), identificando alguns tipos de nós (colunas) que podem ser podados.

1. Coluna Perfeita: nó folha carregando coluna k com custo reduzido nulo ($\bar{r}_k = 0$);

2. Coluna Pesada: nó folha carregando coluna k com atribuição de tarefas T_k no nível t da árvore, onde $H_i^t \subset H_i$ é o conjunto de tarefas ainda não visitadas pela enumeração e $\sum_{j \in T_k} w_{ij} = b_i$ ou $\sum_{j \in T_k} w_{ij} + w_{if} > b_i \; \forall \; f \in H_i^t$;

3. Coluna Rasa: nó folha carregando coluna k com *bound* de custo reduzido mínimo $\bar{r}_{k_{min}} > d$.

***Script* 10** Enumeração de Colunas para Modelo Mestre Restrito (PAG)

Data: modelo mestre restrito (Dantzig-Wolfe) DWM-R; número máximo $s > 0$ de linhas (*rows*) combinadas para análise de cortes (3, 0.5)-SRCs.

Result: conjunto E de colunas enumeradas.

for `i` ∈ *{conjunto de máquinas}* **do**

- Define uma ordenação H de tarefas, uma para cada nível da árvore;
- Declara um vetor W para armazenar os nós da árvore de enumeração;
- Inicializa a árvore de enumeração com um nó raíz vazio, inserido em W;

for $h = 1; h \leq |H|; h = h + 1$ **do**

- Atualiza $l = |W|$;

for $k = 1; k \leq l; k = k + 1$ **do**

if *tarefa* $H[h]$ *cabe na coluna* $W[k]$ **then**

- Cria uma nova coluna y com todas as tarefas de $W[k]$ e a tarefa y;
- Insere coluna y no vetor W;

- Move para E todas as colunas $k \in W$ com custo reduzido $\bar{r}_k == 0$;
- Move para E todas as colunas $k \in W$ cheias com custo reduzido $\bar{r}_k \leq d$;

- Atualiza o custo reduzido mínimo (*bound*) $\bar{r}_{k_{min}}$ para toda coluna $k \in W$;
- Deleta todas as colunas $k \in W$ com custo reduzido mínimo $\bar{r}_{k_{min}} > d$;

- **Retorna** E;

As colunas perfeitas são aquelas cujo custo reduzido não sofre mais alterações na enumeração, portanto são separadas da árvore para compor a base E de colunas.

As colunas pesadas são aquelas que não possuem mais descendentes porque qualquer tarefa adicionada causa um excesso de peso em relação à capacidade máxima da respectiva máquina i. Essas colunas são removidas da árvore de enumeração; e caso tenham custo reduzido $\bar{r}_k \leq d$, elas são adicionadas à base E.

As colunas rasas são aquelas que jamais atingirão um custo reduzido suficientemente baixo para compor a base E de colunas do modelo de Dantzig-Wolfe para o PAG. Então, essas colunas são removidas da árvore de enumeração, simplesmente descartadas. Esse mecanismo de separação de colunas permite uma melhoria significativa do desempenho do algoritmo de enumeração. A especificação desse algoritmo abrange tanto a eficiente ordenação de tarefas quanto as heurísticas para gerir as colunas enumeradas e considerar explicitamente apenas aquelas colunas que podem fazer parte da solução ótima do modelo.

A ordenação H_i da máquina i define uma tarefa para cada nível da árvore de enumeração que já pode ser expandida iterativamente, partindo do nó raíz que é inicializado com custo reduzido igual à dual de máquina π_i. Diante de todas as considerações, qualidades e especificações dispostas, o algoritmo de enumeração de colunas está elaborado conforme descrito em 10.

5.3 Finalidade da Enumeração de Colunas

O resultado do algoritmo de enumeração de colunas é uma base E de colunas com custo reduzido $\bar{r}_k \leq d$, onde $d = UB - LB$ é o *gap* de referência. Essa base E contém as colunas necessárias para fechar o *gap*, ou seja, para satisfazer uma solução inteira ótima para o modelo de decomposição (Dantzig-Wolfe) do PAG.

A finalidade da enumeração de colunas é localizar e adicionar todas as colunas que faltam ao mestre restrito do *price-and-cut*, não contidas por E_0, para que exista uma solução inteira ótima para esse modelo. Se o *gap* for nulo (0), então a solução do mestre restrito já é a solução inteira ótima, visto que o LB é igual ao UB. A enumeração de colunas não é executada nesse caso.

Em síntese, todas as colunas da base E formada pela geração e enumeração de colunas são adicionadas ao modelo da formulação de Dantzig-Wolfe do PAG. Em seguida, esse modelo é resolvido por programação inteira (utilizando o Gurobi) para encontrar uma solução inteira ótima para a respectiva instância do PAG.

Em caso de enumeração não exaustiva, as bases E estão incompletas como ocasionado pela interrupção da enumeração de colunas; logo, uma solução inteira pode não existir e, caso exista, não há comprovação de otimalidade para a solução

inteira encontrada para o modelo de Dantzig-Wolfe. A enumeração de colunas dissertada é a última parte do método proposto e sucede ao *price-and-cut*, conforme o algoritmo completo descrito em 11.

***Script* 11** Algoritmo *Price-and-Cut* com Desigualdades (3, 0.5)-SRC

Result: Solução inteira (potencialmente ótima) do modelo de Dantzig-Wolfe para a instância do PAG.

- Inicializa o modelo de formulação Dantzig-Wolfe com variáveis artificiais;
- Declara um conjunto R de colunas geradas;
- Declara um conjunto S de desigualdades separadas;

repeat
- Esvazia o conjunto R de colunas geradas;
- Esvazia conjunto S de desigualdades separadas;

repeat
- Resolve o modelo mestre (restrito) da formulação Dantzig-Wolfe;

for $i \in \{$ *máquinas* $\}$ **do**
- Executa o *pricing*, resolvendo o subproblema associado à máquina i;
- Calcula o custo reduzido $\bar{r}_k$ (*pricing*) da melhor coluna k (máquina i);

if $\bar{r}_k \leq -\epsilon$ **then**
- **Gera** coluna k, adicionando ao mestre restrito (Dantzig-Wolfe);
- Insere a coluna gerada k no conjunto R;

until $R \neq \varnothing$;

if $R == \varnothing$ **then**
- ***break***;

- **Separa** desigualdades (3, 0.5)-SRC para o modelo mestre (Dantzig-Wolfe);
- Insere as desigualdades separadas ao conjunto S;

until $S \neq \varnothing$;

for $i \in \{$ *máquinas* $\}$ **do**
- **Enumera** colunas para a máquina i, a partir das duais da solução linear ótima (mestre restrito);

- **Constrói** modelo DWM-R (Dantzig-Wolfe) para a instância do PAG apenas com colunas geradas e colunas enumeradas;

if *DWM-R* ***viável*** **then**
- **Retorna** solução inteira (potencialmente ótima) do modelo DWM-R;

- **Retorna** -1;

6
Resultados Computacionais

6.1
Descrição dos Parâmetros

Os métodos empregados pelo algoritmo de *price-and-cut* são definidos por alguns parâmetros descritos no decorrer do texto desta dissertação. A implementação utilizada nos experimentos computacionais permite ajustar todos esses parâmetros. A especificação de valores para esses parâmetros é ajustada empiricamente.

A geração de colunas é executada em função do parâmetro $\epsilon > 0$ que regula a condição de parada, interrompendo o algoritmo de geração de colunas quando todos os custos reduzidos do *pricing* são maiores que $-\epsilon$. Alguns outros parâmetros são ajustáveis para esse algoritmo, todos servindo para interromper a geração quando necessário. O número máximo de iterações n_i, o tempo máximo de execução t_{max} sem alteração da função objetivo e a quantidade máxima n_k de colunas geradas são alguns desses parâmetros disponíveis. Evidentemente que a interrupção precoce da geração de colunas é indesejável e deixa de oferecer a garantia de que todas as colunas enumeráveis terão custo reduzido nulo ou positivo. Então, sempre que a geração de colunas é abortada, o *price-and-cut* falha para que exceda os recursos computacionais disponíveis para a execução, seja o tempo de CPU ou a memória RAM disponíveis.

- O parâmetro ϵ foi ajustado com o valor 0.0001 e serve como uma tolerância para identificar custo reduzido não negativo no *pricing* da geração de colunas;
- O parâmetro n_i foi ajustado com o valor de 10000, interrompendo a geração de colunas após 10000 iterações;
- O parâmetro t_{max} foi ajustado com o valor de 7000 segundos, interrompendo a geração de colunas após 7000 segundos consecutivos sem melhoria da função objetivo do mestre restrito;
- O parâmetro n_k foi ajustado com o valor de 100000 colunas, interrompendo a geração de colunas quando pelo menos 100000 colunas forem geradas.

A estabilização de Wentges (1997) com decomposição de Dantzig-Wolfe ponderada funciona com dois parâmetros, um fator $0.0 < \alpha \leq 1.0$ para a ponderação das duais consideradas pelo *pricing* de colunas e um $\gamma > 0.0$ para identificar a convergência das duais na estabilização.

– O parâmetro α foi ajustado com o valor 0.1, aumentando o peso das duais da melhor solução encontrada. Esse valor é heuristicamente aumentado durante a geração de colunas para cada instância resolvida;
– O parâmetro γ foi ajustado com o valor 0.01, estabelecendo uma tolerância para identificar a convergência das duais na estabilização, suspendendo a estabilização a partir dessa convergência;

A geração de desigualdades (3, 0.5)-SRC recebe apenas a dimensão d_L da *subset row cut* (igual a 3), o coeficiente q (igual a 0.5) e um inteiro l que indica o número máximo de restrições de atribuição de tarefa consideradas para analisar as diferentes combinações de restrições que definem esses cortes (3, 0.5)-SRC. Quanto maior for o número de tarefas para análise das (3, 0.5)-SRCs, maior será o número de combinações enumeráveis, onde cada combinação é um conjunto L com três tarefas distintas. O aumento desse valor l implica o aumento exponencial do número de combinações (desigualdades), sendo inviável para $l > 200$, segundo os testes realizados. Portanto, as instâncias resolvidas nos experimentos computacionais foram instâncias com 200 tarefas no máximo.

– O parâmetro d_L assume valor 3 para desigualdades (3, 0.5)-SRCs;
– O parâmetro q assume valor 0.5 para desigualdades (3, 0.5)-SRCs;
– O parâmetro l foi ajustado com o valor 200, considerando apenas as 200 melhores tarefas para analisar (3, 0.5)-SRCs (selecionadas heuristicamente);

A enumeração de colunas recebe um valor de *gap* d, um período limite de tempo de execução t, um limite w de memória ocupada pelas colunas enumeradas (árvore e colunas salvas), uma quantidade máxima k_{tree} de colunas na árvore e uma quantidade máxima k_{max} de colunas salvas para resolver o modelo de Dantzig-Wolfe para a instância PAG.

– O parâmetro d é o *gap* para restringir as colunas enumeradas e reduzir o espaço de enumeração, sendo d definido pelo próprio algoritmo com base no LB encontrado pelo *price-and-cut* e considerando o melhor UB conhecido na literatura;
– O parâmetro t foi ajustado para considerar um período máximo de 24 horas para a enumeração, distribuído unniformemente pelo número de máquinas da instância (uma enumeração para cada máquina);

– O parâmetro k_{tree} foi ajustado para 3 milhões, interrompendo a enumeração de colunas para uma máquina quando a árvore alcança o tamanho de 3 milhões de nós totais (esse valor pode ser aumentado para máquinas com mais de 15 GB de memória RAM);

– O parâmetro k_{max} foi ajustado para 2 milhões, interrompendo a enumeração de colunas para uma máquina quando o número de colunas enumeradas pela respectiva árvore alcança o tamanho de 2 milhões de nós salvos.

6.2 Análise de Experimentos Computacionais

Os experimentos computacionais foram realizados com uma implementação em C++ do método proposto, compilação com GNU C++ e execução em Ubuntu 20.04.2, 64-bit, com 15.5 GiB de memória RAM e processador Intel ® Core ™ i7-4720HQ CPU @ 2.60GHz x 8.

Os resultados apresentados são referentes às instâncias da OR-Library com um máximo de 200 tarefas, abrangendo as 5 categorias de instâncias do Problema de Alocação Generalizada (PAG), especificamente as instâncias listadas abaixo, organizadas por categoria. O nome das instâncias são padronizados no formato xMMTTT, onde MM representa o número de máquinas e TTT indica o número de tarefas.

– Categoria A: a05100, a05200, a10100, a10200, a20100 e a20200;

– Categoria B: b05100, b05200, b10100, b10200, b20100 e b20200;

– Categoria C: c05100, c05200, c10100, c10200, c20100 e c20200;

– Categoria D: d05100, d05200, d10100, d10200, d20100 e d20200;

– Categoria E: e05100, e05200, e10100, e10200, e20100 e e20200;

Em instâncias muito degeneradas, como a05200, b05200, c05200, d05200 e e05200, a geração de colunas demora, em média, mais de 6 horas, porém esse tempo de execução é reduzido para uma média de 11 minutos quando a estabilização de Wentges é adotada, mostrando um enorme ganho de eficiência com essa técnica de ponderação da decomposição de Dantzig-Wolfe. Essas instâncias degeneradas estão marcadas com o símbolo $+$ nas tabelas de resultado, incluindo algumas instâncias não tão degeneradas que também se beneficiaram da estabilização para reduzir o tempo de geração de colunas. Entretanto, a principal motivação para utilizar essa técnica é a constatada necessidade de estabilização para resolver as instâncias indicadas como degeneradas.

Entre as instâncias consideradas para os experimentos computacionais, há algumas que são especialmente difíceis e outras que são triviais. Em geral, as

instâncias da categoria A são triviais e as instâncias da categoria B não são difíceis, embora sejam menos fáceis que instâncias A. Essas instâncias mais fáceis foram utilizadas para validar o correto funcionamento do método. Ambos o Gurobi e a geração de colunas do *price-and-cut* conseguem resolver as instâncias A até a otimalidade (solução inteira), sendo que o *price-and-cut* encontra essa solução ótima (conhecida) para o modelo de decomposição (Dantzig-Wolfe) apenas com a base de colunas geradas, sem precisar separar desigualdades (3, 0.5)-SRC nem enumerar colunas.

O conjunto de instâncias B não é tão facilmente resolvido pela geração de colunas, sendo necessário adicionar cortes (3, 0.5)-SRC e enumerar colunas para chegar à solução inteira ótima. Porém, o Gurobi sozinho já consegue encontrar uma solução inteira ótima para todas essas instâncias.

O Gurobi consegue resolver a maioria das demais instâncias de categorias C, D e E, mas essas instâncias não são resolvidas somente pela geração de colunas implementada pelo *price-and-cut* e, consequentemente, existe uma maior dificuldade para encontrar soluções inteiras com otimalidade comprovada pela enumeração de colunas. Nos trabalhos Avella (2010) e Michelon (2010), a maioria dessas instâncias são solucionadas a ponto de alcançar um *gap* nulo (solução inteira ótima), exceto algumas instâncias D cujos resultados estão marcados na tabela de resultados com o símbolo *.

Os resultados estão organizados em dois tipos de tabela, tabelas com resultados de solução e tabelas que dispõem resultados de tempo de execução. Essas tabelas também estão agrupadas e classificadas por categoria de instância, para facilitar a leitura categorizada.

A tabela com resultados de solução avalia a eficácia do algoritmo, comparando os valores de solução encontrados pela geração de colunas e enumeração de colunas, inserindo e não inserindo as (3, 0.5)-SRCs. A resumida descrição listada abaixo dispõe uma explicação sobre o significado de cada sigla de coluna das tabelas.

1. UB indica a melhor solução inteira encontrada pelo Gurobi (possível UB);

2. LB indica o melhor LB encontrado pela solução da geração de colunas, com ou sem desigualdades (3, 0.5)-SRC;

3. Gap indica a distância entre o LB e o melhor UB conhecido $\max$(UB, A B&C, M B&B), ou seja $\max$(UB, A B&C, M B&B) - LB;

4. # BEC indica o número de colunas geradas durante o *price-and-cut* (sem enumeração de colunas);

5. # EC indica o número de colunas geradas e enumeradas, considerando a geração e enumeração de colunas;

6. IC indica o número de árvores de enumeração incompletas (interrompidas), considerando haver uma árvore para cada máquina e que a otimalidade da solução inteira encontrada com as colunas geradas e enumeradas é comprovada apenas quando essa coluna tem valor zero (nenhuma árvore de enumeração interrompida);

7. LPO indica a solução da relaxação linear do mestre restrito da decomposição de Dantzig-Wolfe com a base de colunas composta pelas colunas geradas e enumeradas;

8. IPO indica a solução inteira do mestre restrito da decomposição de Dantzig-Wolfe com a base de colunas composta pelas colunas geradas e enumeradas;

9. SRCs indica o número de desigualdades (3, 0.5)-SRC separadas e adicionadas ao mestre restrito;

10. A B&C indica a melhor solução inteira ótima encontrada pelo método de Branch-and-Cut de Avella (2010);

11. M B&B indica a melhor solução inteira ótima encontrada pelo método de Branch-and-Bound de Michelon (2010);

12. Best indica o melhor resultado encontrado para dada instância do PAG, logo $\max$(IPO, A B&C, M B&B).

A tabela com os tempos de execução avalia a eficiência do algoritmo, comparando valores de tempo para realizar a geração e a enmumeração de colunas, com e sem inserção das (3, 0.5)-SRCs. Essa tabela admite colunas com as siglas listadas e descritas abaixo.

1. TA indica o tempo total de execução da geração de colunas sem a separação de desigualdades (3, 0.5)-SRC;

2. TB indica o tempo total de execução da geração de colunas com a separação de desigualdades (3, 0.5)-SRC;

3. CG indica o tempo de execução da geração de colunas (contabilizando todas as vezes que a geração de colunas é realizada, entre adição de cortes);

4. CGK indica o tempo total de execução gasto resolvendo subproblemas de mochila para o *pricing* da geração de colunas;

5. CE indica o tempo total de execução da enumeração de colunas;

6. CEBK indica o tempo total de execução gasto resolvendo subproblemas de mochila para calcular o custo reduzido mínimo de colunas nos nós das árvores;

7. SRC indica o tempo total de execução gasto para a geração de desigualdades (3, 0.5), separação e adição dos cortes ao mestre restrito.

Instância		**Enumeração de Colunas para modelo Dantzig-Wolfe (CG)**							**Análise**
ID	**UB**	**LB**	**Gap**	**# BEC**	**# EC**	**IC**	**LPO**	**IPO**	**Best**
a05100	**1698.000**	1698.000	**0.000**	251	-	-	1698.00	**1698.0**	**1698**
a05200	**3235.000**	3235.000	**0.000**	42098	-	-	3235.00	**3235.0**	**3235**
+ a10100	**1360.000**	1360.000	**0.000**	1737	-	-	1360.00	**1360.0**	**1360**
a10200	**2623.000**	2623.000	**0.000**	676	-	-	2623.00	**2623.0**	**2623**
a20100	**1158.000**	1158.000	**0.000**	213	-	-	1158.00	**1158.0**	**1158**
a20200	**2339.000**	2339.000	**0.000**	509	-	-	2339.00	**2339.0**	**2339**

Tabela 6.1: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, sem desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria A da OR-Library.

Instância	**Tempo de CPU (CG + CE) [minutos]**					**Análise [minutos]**
ID	**TA**	**CG**	**CGK**	**CE**	**CEBK**	**TA + CE**
a05100	**0.8874**	**0.8813**	0.1321	**0.0000**	0.0000	**0.8874**
a05200	**403.3423**	**403.2014**	7.4778	**0.1180**	0.0000	**403.3423**
a10100	**0.4237**	**0.4179**	0.1385	**0.0174**	0.0139	**0.4411**
a10200	**23.955**	**23.912**	1.1785	**0.0000**	0.0000	**23.955**
a20100	**0.1857**	**0.1815**	0.0947	**0.0000**	0.0000	**0.1857**
a20200	**4.0467**	**4.0261**	0.7722	**0.0000**	0.0000	**4.0467**

Tabela 6.2: Tabela de resultados com os valores de tempo de execução que servem à análise de eficiência do algoritmo proposto. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria A da OR-Library.

As tabelas 6.1 e 6.2 mostram resultados já esperados para as instâncias da categoria A, sendo bastante fáceis. Em todas instâncias, a geração de colunas é suficiente para encontrar um LB igual à solução inteira ótima, obtendo *gap*

nulo. Então, a introdução de desigualdades e a enumeração de colunas não são necessárias.

Embora a instância a05200 seja degenerada, ela já havia sido resolvida antes da implementação da estabilização de Wentges. Portanto, esse resultado não recorre à estabilização, ocasionando a geração de 42098 colunas em aproximadamente 403 minutos (quase 7 horas), uma quantidade de colunas que extrapola bastante o número de colunas geradas em média e um tempo extremamente longo para a geração de colunas em comparação com a média de duração desse processo. Neste caso, a geração de colunas ocorre em meio à oscilações de duais para o *pricing*.

Em Avella (2010) e Michelon (2010), essas instâncias não são tratadas. Logo, a referência utilizada é a solução encontrada pelo Gurobi.

Instância		**Enumeração de Colunas para modelo Dantzig-Wolfe (CG)**							**Análise**
ID	**UB**	**LB**	**Gap**	**# BEC**	**# EC**	**IC**	**LPO**	**IPO**	**Best**
b05100	1843.000	1838.837	4.163 (4)	3852	448168	**0**	1838.83	**1843.0**	**1843**
+ b05200	3552.000	3549.337	2.663	3453	**654954**	5	3549.34	**3555.0**	**3552**
b10100	1407.000	1407.000	**0.000**	214	-	-	1407.00	**1407.0**	**1407**
b10200	2827.000	2825.509	1.491 (1)	629	59144	**0**	2825.509	**2827.0**	**2827**
b20100	1166.000	1166.000	**0.000**	167	**279**	**0**	1166.00	**1166.0**	**1166**
b20200	2339.000	2338.521	0.479	516	**4496**	**0**	2338.52	**2339.0**	**2339**

Tabela 6.3: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, sem desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria B da OR-Library.

Instância		**Enumeração de Colunas para modelo Dantzig-Wolfe (CG) + 3-SRCs**								**Análise**
ID	**UB**	**SRCs**	**LB**	**Gap**	**# BEC**	**# EC**	**IC**	**LPO**	**IPO**	**Best**
b05100	1843.000	**39**	1840.531	**2.469 (2)**	4622	**2297**	**0**	1838.837	**1843.0**	**1843**
+ b05200	3552.000	**66**	3549.871	**2.124**	7706	676869	**1**	3549.34	**3552.0**	**3552**
b10100	1407.000	-	-	-	-	-	-	-	-	**1407**
b10200	2827.000	**25**	2825.750	**1.250 (1)**	812	**11389**	**0**	2825.51	**2827.0**	**2827**
b20100	1166.000	-	-	-	-	-	-	-	-	**1166**
b20200	2339.000	**2**	2338.555	**0.444**	516	9166	**0**	2338.52	**2339.0**	**2339**

Tabela 6.4: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, com desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria B da OR-Library.

As tabelas 6.3, 6.4 e 6.5 mostram que a geração de colunas, na maioria das instâncias B, não determina uma base de colunas suficientes para resolver a formulação inteira do mestre restrito de Dantzig-Wolfe. Em quase todas as instâncias, exceto as instâncias b10100 e b20100, a separação de desigualdades (3, 0.5)-SRC funciona para aumentar o LB, reduzindo significativamente o número de colunas enumeráveis e, consequentemente, a enumeração de colunas se mostra viável e imprescindível para encontrar uma solução inteira. O LB aumenta para as instâncias b05100, b05200, b10200 e b20200, levando a queda do *gap*. Esse estreitamento do *gap* decorrente da adição de cortes é mais acentuado para a instância b05100, caindo de 4.163 para 2.469, uma redução de 40.7

Instância	Tempo de CPU (CG + CE) [minutos]					Tempo de CPU (CG + 3-SRCs + CE) [minutos]						Análise [minutos]
ID	TA	CG	CGK	CE	CEBK	TB	CG	SRC	CGK	CE	CEBK	TA + CE / TB + CE
b05100	**0.8601**	**0.8542**	0.2822	**44.524**	41.280	**20.667**	**15.271**	0.8935	4.6207	**1.5702**	1.4978	**45.384 / 22.2372**
+ b05200	**6.6518**	**6.6323**	0.7179	**722.067**	680.465	**124.156**	**112.330**	11.5336	11.3998	**407.556**	372.552	**728.719 / 531.712**
b10100	**0.3121**	**0.3087**	0.1534	-	-	-	-	-	-	-	-	**0.3121 / -**
b10200	**13.332**	**13.303**	3.1805	**17.093**	16.531	**33.381**	**27.628**	1.8886	5.8174	**2.6630**	2.4431	**30.425 / 36.044**
b20100	**0.1664**	**0.1625**	0.0997	**0.0240**	0.0188	-	-	-	-	-	-	**0.1904 / -**
b20200	**1.2925**	**1.2794**	0.3359	**0.3187**	0.2946	**2.2969**	**1.8643**	0.3652	0.3486	**8.8242**	7.9412	**1.6112 / 11.121**

Tabela 6.5: Tabela de resultados com os valores de tempo de execução que servem à análise de eficiência do algoritmo proposto. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria B da OR-Library.

A coluna IC apresenta valor 0 (zero) para a enumeração da maioria das instâncias, indicando que a enumeração de colunas não foi interrompida em nenhuma árvore (para nenhuma máquina), comprovando a otimalidade da solução inteira encontrada (IPO). A instância b05200 é a única que não obteve enumeração de colunas completa devido ao alto custo de computar o custo mínimo reduzido em cada nó de uma árvore que se mantém muito grande durante todo o processo. Os valores de tempo estão dispostos na segunda tabela, mostrando que a enumeração (sem adição de desigualdades) para essa instância demorou pouco mais de 722 minutos dos quais mais de 680 minutos foram dedicados a resolver a otimização de mochilas para o custo reduzido de colunas da árvore.

A eficiência da enumeração de colunas melhora bastante para a instância b05200 quando as (3, 0.5)-SRCs são empregadas, interrompendo a enumeração apenas para 1 máquina e executando toda a enumeração em pouco mais de 407 minutos dos quais pelo menos 372 minutos são dedicados à otimização para o *bound* de custo reduzido mínimo. Entretanto, ainda há interrupção, a enumeração não é completa e a otimalidade não é comprovada, apesar da solução inteira (IPO) encontrada ser igual à solução inteira ótima computada pelo Gurobi. Em análise de eficiência, cabe perceber também que o tempo da geração de colunas

para essa instância b05200 é muito maior com a inserção de (3, 0.5)-SRCs e isso acontece porque a estabilização não funciona com essas desigualdades e, consequentemente, a geração de colunas é muito lenta após a adição dos cortes (devido à degenerescência).

Instância		**Enumeração de Colunas para modelo Dantzig-Wolfe (CG)**							**Análise**		
ID	**UB**	**LB**	**Gap**	**# BEC**	**# EC**	**IC**	**LPO**	**IPO**	**A B&C**	**M B&B**	**Best**
c05100	1931.000	1929.666	1.333 (1)	4382	12632	**0**	1929.67	**1931.0**	1931	1931	**1931**
+ c05200	3456.000	3454.493	1.508	3234	685709	**0**	3454.49	**3456.0**	3456	3456	**3456.0**
c10100	1402.000	1399.857	2.143 (2)	1867	5727	**0**	1399.86	**1402.0**	1402	1402	**1402**
+ c10200	2806.000	2803.949	2.051 (2)	3182	1472886	**0**	2803.95	**2806.0**	2806	2806	**2806**
c20100	1243.000	1241.666	1.333 (1)	219	772	**0**	1241.67	**1243.0**	1243	1243	**1243**
c20200	2391.000	2390.171	0.829	591	**5019**	**0**	2390.17	**2391.0**	2391	2391	**2391**

Tabela 6.6: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, sem desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria C da OR-Library.

Instância		**Enumeração de Colunas para modelo Dantzig-Wolfe (CG) + 3-SRCs**								**Análise**		
ID	**UB**	**SRCs**	**LB**	**Gap**	**# BEC**	**# EC**	**IC**	**LPO**	**IPO**	**A B&C**	**M B&B**	**Best**
c05100	1931.000	**48**	1930.572	**0.428**	5094	**356**	**0**	1929.67	**1931.0**	1931	1931	**1931**
+ c05200	3456.000	**33**	3455.175	**0.825**	5719	**25811**	**0**	3454.49	**3456.0**	3456	3456	**3456.0**
c10100	1402.000	**8**	1400.708	**1.292 (1)**	2088	**3190**	**0**	1399.86	**1402.0**	1402	1402	**1402**
+ c10200	2806.000	**39**	2804.251	**1.749 (1)**	4126	**16681**	**0**	2803.95	**2806.0**	2806	2806	**2806**
c20100	1243.000	**1**	1242.000	**1.000 (1)**	246	**728**	**0**	1241.67	**1243.0**	1243	1243	**1243**
c20200	2391.000	**10**	2390.210	**0.790**	648	7376	**0**	2390.17	**2391.0**	2391	2391	**2391**

Tabela 6.7: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, com desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria C da OR-Library.

As tabelas 6.6, 6.7 e 6.8 mostram que a geração de colunas sozinha, para instâncias C, não forma base de colunas satisfatória para computar uma solução inteira ótima pelo mestre restrito. O valor de UB de referência adotado para o cálculo do *gap* é a maior solução inteira, considerando os trabalhos Avella (2010) e Michelon (2010). Em todas as instâncias C listadas, o *gap* mínimo obtido pelo *price-and-cut* é maior que 0 (zero). Todavia, a enumeração de colunas completa é viável sem a introdução das (3, 0.5)-SRC, independemente da eficiência observada.

Instância	Tempo de CPU (CG + CE) [minutos]					Tempo de CPU (CG + 3-SRCs + CE) [minutos]						Análise [minutos]
ID	TA	CG	CGK	CE	CEBK	TB	CG	SRC	CGK	CE	CEBK	TA + CE / TB + CE
c05100	**1.3693**	**1.3622**	0.4303	**1.6310**	1.5600	**10.3199**	**5.7503**	0.5569	1.1637	**0.5292**	0.4894	**3.0003 / 10.849**
+ c05200	**3.7388**	**3.730**	0.3494	**64.4143**	57.2365	**48.8292**	**45.8952**	2.8738	4.8186	**7.8033**	6.8950	**68.1531 / 56.6325**
c10100	**0.2380**	**0.2349**	0.1241	**0.3080**	0.2746	**0.7797**	**0.6662**	0.1090	0.2529	**0.3318**	0.2656	**0.546 / 1.1115**
+ c10200	**3.2759**	**3.2651**	0.4714	**227.859**	212.576	**12.7496**	**9.4756**	3.2404	1.9295	**4.3559**	3.7927	**231.1349 / 17.1055**
c20100	**0.1211**	**0.1178**	0.0743	**0.0452**	0.0384	**0.2018**	**0.1575**	0.0394	0.0689	**0.0962**	0.0728	**0.1663 / 0.298**
c20200	**1.4014**	**1.3872**	0.4619	**0.7618**	0.7226	**2.8079**	**1.9081**	0.6791	0.4646	**2.0403**	1.8199	**2.1632 / 4.8482**

Tabela 6.8: Tabela de resultados com os valores de tempo de execução que servem à análise de eficiência do algoritmo proposto. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria C da OR-Library.

Em contrapartida, as desigualdades (3, 0.5) aumentam significativamente a eficiência da enumeração de colunas para todas as instâncias, exceto pela instância c20200. O tempo de enumeração também diminui muito mais que 50% na maioria das instâncias dessa categoria, sendo a maior redução observada para a instância c05200 cujo tempo de enumeração desce de mais de 64 minutos para quase 8 minutos, uma queda de cerca de 88%. O numero de colunas enumeradas para essa instância também tem uma brusca queda, caindo de 685709 para 25811 colunas (redução de 96.24%). Essa magnitude de ganho em eficiência é também observada para as outras instâncias C, excluindo a c10100 e c20200. A instância c10100 apresenta um ganho de eficiência bem reduzido e o método não revela uma melhoria de desempenho para a instância c20200.

Portanto, o método proposto mostra aptidão para resolver as instâncias PAG do tipo C listadas, computando a solução inteira com otimalidade comprovada e ganhando muita eficiência com a utilização de cortes (3, 0.5)-SRC.

Instância		Enumeração de Colunas para modelo Dantzig-Wolfe (CG)							Análise		
ID	UB	LB	Gap	# BEC	# EC	IC	LPO	IPO	A B&C	M B&B	Best
d05100	6353.000	6349.921	3.079 (3)	3396	202850	1	6349.92	**6354.0**	6353	6353	**6353**
+ d05200	12742.000	12740.039	1.961	4089	142545	5	12740.04	-	12742	12742	**12742**
d10100	6347.000	6341.450	5.550 (5)	489	109539	**0**	6341.45	**6347.0**	* 6347	6347	**6347**
d10200	12430.00	12425.615	4.385	3847	-	-	-	-	* 12430	12430	**12430**
d20100	6211.000	6176.142	8.858 (8)	573	**94445**	**0**	6176.14	**6185.0**	* 6185	6185	**6185**
d20200	12272.000	1229.664	3.336	3242	1301412	**0**	12229.66	**Killed**	* 12233	* 12235	***** 12233**

Tabela 6.9: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, sem desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria D da OR-Library.

Instância		Enumeração de Colunas para modelo Dantzig-Wolfe (CG) + 3-SRCs								Análise		
ID	**UB**	**SRCs**	**LB**	**Gap**	**# BEC**	**# EC**	**IC**	**LPO**	**IPO**	**A B&C**	**M B&B**	**Best**
d05100	6353.000	**83**	6350.825	**2.175 (2)**	3817	**106902**	**0**	6349.921	**6353.0**	6353	6353	**6353**
+ d05200	12742.000	**149**	12740.74	**1.258**	5282	**117659**	**0**	12740.039	**12742.0**	12742	12742	**12742**
d10100	6347.000	**48**	6342.037	**4.963 (4)**	630	**48930**	**0**	6341.450	**6347.0**	* 6347	6347	**6347**
d10200	12430.00	**106**	12426.525	**3.475**	4606	**126660**	**10**	12425.615	**12440.0**	* 12430	12430	**12430**
d20100	6211.000	**21**	6176.419	**8.581 (8)**	599	108270	**0**	6176.140	**6185.0**	* 6185	6185	**6185**
d20200	12272.000	**64**	1229.989	**3.011**	3517	**847034**	**0**	12229.664	**Killed**	* 12233	* 12235	*** 12233**

Tabela 6.10: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, com desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria D da OR-Library.

Instância	Tempo de CPU (CG + CE) [minutos]					Tempo de CPU (CG + 3-SRCs + CE) [minutos]						Análise [minutos]
ID	**TA**	**CG**	**CGK**	**CE**	**CEBK**	**TB**	**CG**	**SRC**	**CGK**	**CE**	**CEBK**	**TA + CE / TB + CE**
d05100	**0.8733**	**0.8687**	0.3653	**565.199**	536.258	**21.518**	**13.700**	1.0101	4.9608	**235.46**	232.72	**566.0723 / 256.9800**
+ d05200	**12.4513**	**12.4377**	1.2968	**720.692**	704.227	**52.3098**	**40.6218**	11.2172	4.6522	**340.798**	334.543	**733.1433 / 393.1078**
d10100	**0.6636**	**0.6592**	0.5080	**52.100**	49.243	**3.5486**	**2.4493**	0.5273	1.4513	**39.909**	38.830	**52.764 / 43.458**
d10200	**15.7316**	**15.7191**	2.0760	-	-	**46.3222**	**34.7581**	11.2959	7.7509	**834.439**	823.907	**15.7316 / 880.7612**
d20100	**0.6189**	**0.6130**	0.5095	**24.705**	22.284	**1.6997**	**1.1587**	0.3838	0.8143	**27.190**	24.381	**25.324 / 28.890**
d20200	**4.0549**	**4.0371**	1.9700	**512.818**	448.134	**8.9642**	**5.4779**	3.4026	2.0641	**385.85**	350.148	**516.8729 / 394.8142**

Tabela 6.11: Tabela de resultados com os valores de tempo de execução que servem à análise de eficiência do algoritmo proposto. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria D da OR-Library.

As tabelas 6.9, 6.10 e 6.11 mostram que o método de *price-and-cut* com (3, 0.5)-SRCs e enumeração de colunas também funciona, em certa medida, para as instâncias D que são as mais difíceis. Naturalmente, somente a geração de colunas não resolve essas instâncias até um *gap* nulo. Novamente, o valor de UB de referência adotado para o cálculo do *gap* é a maior solução inteira, considerando os trabalhos Avella (2010) e Michelon (2010).

Há uma redução do número de colunas enumeradas causada pela introdução das (3, 0.5)-SRCs para a maioria das instâncias D listadas, mostrando que o método mais uma vez cumpre com o seu propósito. Essa redução é mais acentuada para a instância d10100, com uma descida de 109539 para 48930 colunas (redução de 55.33%).

Não apenas ocorre essa redução da base de colunas enumeradas como também os experimentos demonstram a necessidade imprescindível das (3, 0.5)-SRCs para viabilizar a enumeração de colunas completa, como observado para as instâncias d05100, d05200 e d10200 para as quais a enumeração de colunas completa não é atingida apenas com a geração de colunas (price).

Em relação aos tempos de execução, há também um ganho de eficiência entre 23% e 58% em quase todas as instâncias, ocorrendo uma diminuição máxima de tempo de 565 para 235 minutos (58%).

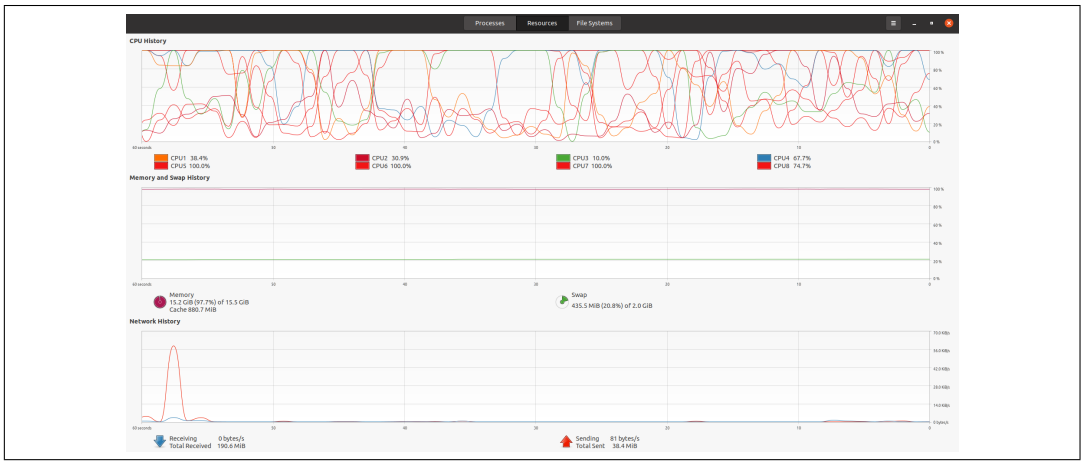

Figura 6.1: Monitorização da memória RAM utilizada pela máquina durante a otimização do modelo de Dantzig-Wolfe MIP pelo Gurobi com as colunas geradas e as colunas enumeradas. A memória RAM utilizada ultrapassa 15 GB, causando a interrupção do experimento após um tempo.

Entretanto, as desigualdades (3, 0.5)-SRC não são capazes de reduzir suficientemente o conjunto de colunas enumeradas para as instâncias d10200 e d20200 a ponto de tornar o mestre restrito inteiro viável. Embora a enumeração de colunas seja completa para a instância d20200, o Gurobi não consegue resolver o modelo mestre restrito de formulação inteira construído com as colunas geradas e enumeradas. Em detalhe, o Gurobi demora mais de 12 horas até que a memória ocupada pelo solucionador Gurobi (modelo) exceder 15 GB de memória RAM, forçando que o Ubuntu aborte o programa (*killed*), como indicado na Figura 6.1 acima. É necessário otimizar o solucionador do modelo restrito para evitar o alto consumo de memória RAM. Porém, dado que a enumeração é completa, a solução inteira do mestre restrito com essas colunas será teoricamente a solução ótima.

A instância d10200 é mais difícil de resolver porque a enumeração completa de colunas não é possível nem com a separação de (3, 0.5)-SRCs. Ainda assim, a solução inteira encontrada (12440) não é tão distante da melhor solução conhecida (12430), mostrando que o método não se rende inútil nem mesmo na instância mais difícil entre as listadas para os experimentos. Caso as desigualdades não sejam empregadas, a enumeração de colunas para a instância d102300 é temporalmente inviável. Isso ocorre devido à dificuldade do problema de mochila para computar o *bound* do custo reduzido mínimo das colunas nos nós da árvore. Em decorrência disso, a enumeração de colunas para essa instância demora 834 minutos dos quais 824 minutos são exclusivamente dedicados à otimização do *bound* de custo reduzido mínimo. A implementação de um algoritmo de mochila mais eficiente

poderia tornar essa enumeração viável, como o algoritmo do Pisinger (1997) adaptado para considerar as desigualdades (3, 0.5)-SRC separadas.

Instância		Enumeração de Colunas para modelo Dantzig-Wolfe (CG)							Análise		
ID	UB	LB	Gap	# BEC	# EC	IC	LPO	IPO	A B&C	M B&B	Best
e05100	12681.000	12673.046	7.954 (7)	340	**15690**	**0**	12673.05	**12681.0**	12681	12681	**12681**
+ e05200	24931.000	24926.643	3.357	8272	1186739	3	**Killed**	**Killed**	24930	24930	**24930**
e10100	11577.000	11568.023	8.977 (8)	428	34868	**0**	11568.02	**11577.0**	11577	11577	**11577**
e10200	23307.000	23302.050	4.950 (4)	1525	437014	4	23302.05	**23308.0**	23307	23307	**23307**
e20100	8436.000	8431.510	4.490 (4)	1977	**4378**	**0**	8431.51	**8436.0**	8436	8436	**8436**
e20200	22379.000	22376.763	2.237 (2)	5422	87929	**0**	22376.76	**22379.0**	22379	22379	**22379**

Tabela 6.12: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, sem desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria E da OR-Library.

Instância		Enumeração de Colunas para modelo Dantzig-Wolfe (CG) + 3-SRCs								Análise		
ID	UB	SRCs	LB	Gap	# BEC	# EC	IC	LPO	IPO	A B&C	M B&B	Best
e05100	12681.000	**59**	12674.992	**6.008 (6)**	551	16516	**0**	12673.05	**12681.0**	12681	12681	**12681**
+ e05200	24931.000	**38**	24926.643	**2.500**	7995	**518934**	**0**	24926.64	**24930.0**	24930	24930	**24930**
e10100	11577.000	**41**	11569.651	**7.349 (7)**	583	**23894**	**0**	1568.023	**11577.0**	11577	11577	**11577**
e10200	23307.000	**55**	23303.410	**3.591 (3)**	1726	**873548**	**0**	23302.05	**23307.0**	23307	23307	**23307**
e20100	8436.000	**17**	8431.715	**4.285 (4)**	2023	5399	**0**	8431.51	**8436.0**	8436	8436	**8436**
e20200	22379.000	**25**	22377.719	**1.281 (1)**	5782	**13705**	**0**	22376.76	**22379.0**	22379	22379	**22379**

Tabela 6.13: Tabela de resultados com os valores de solução inteira encontrados que servem à análise de eficácia do algoritmo proposto, com desigualdades (3, 0.5)-SRC. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria E da OR-Library.

Instância	Tempo de CPU (CG + CE) [minutos]					Tempo de CPU (CG + 3-SRCs + CE) [minutos]						Análise [minutos]
ID	TA	CG	CGK	CE	CEBK	TB	CG	SRC	CGK	CE	CEBK	TA + CE / TB + CE
e05100	**1.2096**	**1.2034**	0.5191	**2.3965**	2.3001	**11.3016**	**6.8133**	0.8500	1.6804	**6.7757**	6.5100	**3.6061 / 18.0773**
+ e05200	**20.4997**	**20.4743**	1.8476	**480.615**	443.201	**68.9668**	**62.9093**	5.9359	5.5446	**157.764**	139.926	**501.1147 / 226.7308**
e10100	**0.4315**	**0.4276**	0.2574	**2.8229**	2.6947	**2.0282**	**1.2861**	0.2280	0.4110	**3.1362**	2.9469	**3.2544 / 5.1644**
e10200	**7.7271**	**7.6917**	1.7492	**260.69**	251.51	**47.843**	**32.167**	5.3388	4.4628	**122.99**	116.22	**268.41 / 170.83**
e20100	**0.4833**	**0.4783**	0.3397	**0.9553**	0.8721	**1.2543**	**0.9823**	0.2615	0.5993	**1.2682**	1.0794	**1.4386 / 2.5225**
e20200	**2.8740**	**2.8615**	0.9161	**23.347**	20.948	**7.4462**	**4.8040**	2.6090	1.1206	**6.8781**	5.8198	**26.221 / 14.324**

Tabela 6.14: Tabela de resultados com os valores de tempo de execução que servem à análise de eficiência do algoritmo proposto. Os valores desta tabela foram obtidos pelos experimentos computacionais para instâncias da categoria E da OR-Library.

As tabelas 6.12, 6.13 e 6.14 mostram uma completa eficácia do método de *price-and-cut* com desigualdades (3, 0.5)-SRC e enumeração de colunas, visto que uma solução inteira ótima é encontrada para todas as instâncias da categoria E.

Em destaque, a instância e05200 demonstra mais uma vez a eficácia do método proposto. A enumeração de colunas a partir de colunas geradas é interrompida para três máquinas, adicionando mais de 1 milhão de colunas ao mestre restrito resolvido com programação inteira. Em decorrência dessa quantidade de colunas, o solucionador do Gurobi excede a memória RAM disponível na tentativa encontrar uma solução inteira para o mestre restrito, conduzindo o SO a abortar o programa. Então, nenhuma solução inteira é encontrada.

Entretanto, a separação de desigualdades (3, 0.5)-SRC viabiliza a enumeração completa de colunas, compondo uma base de colunas bem menor (redução de 56.27%), pequena o suficiente para que o solucionador Gurobi consiga encontrar uma solução inteira ótima para o modelo mestre restrito de Dantzig-Wolfe. Não somenrte ocorre essa diminuição do número de colunas enumeradas como também uma redução de 67.17% do tempo de enumeração de coluans é observada.

A instância e10200 mostra também se beneficiar das (3, 0.5)-SRCs para formar uma base de colunas que não conduza o Ubuntu a abortar o solucionador Gurobi. Neste caso, o numéro de colunas enumeradas aumenta mas o tempo de enumeração diminui 52.82%.

Portanto, o método proposto mostra aptidão para resolver as instâncias PAG do tipo E listadas, computando a solução inteira com otimalidade comprovada para todas essas instâncias e ganhando muita eficiência com a utilização de cortes (3, 0.5)-SRC.

7
Trabalhos Futuros

Os resultados reportados nesta dissertação revelam algumas limitações do método *price-and-cut* defendido, pontualmente comentadas na análise de resultados e brevemente descritas abaixo.

1. A enumeração de colunas para o mestre restrito de Dantzig-Wolfe (*price-and-cut*) torna-se computacionalmente muito custosa (lenta) para instâncias com mais de 20 máquinas e com mais de 200 tarefas;

2. Em algumas instâncias mais difíceis, o *gap* mínimo encontrado pela geração de colunas com desigualdades (3, 0.5)-SRC não é suficientemente reduzido para viabilizar uma enumeração de colunas exaustiva;

3. Em algumas instâncias mais difíceis, a enumeração de colunas torna-se muito lenta devido ao insuficiente desempenho da programação inteira (Gurobi) para resolver o problema de otimização do custo reduzido mínimo das colunas nos nós das árvores de enumeração.

A constatação dessas limitações motiva a investigação de novas abordagens que permitam otimizar ainda mais a enumeração de colunas. O algoritmo de enumeração de colunas apresenta um desempenho insuficiente causado sobretudo pelo tamanho da instância, pelo valor do *gap* mínimo encontrado e pela dificuldade dos problemas de mochila para cálculo do *bound* de custo reduzido mínimo das colunas nos nós da árvore de enumeração. Em consideração a essas limitações e às causas identificadas, há pelo menos quatro trabalhos futuros de pesquisas pertinentes para seguir, discriminados abaixo.

1. Pesquisar a adaptação de algoritmos mais eficientes que a programação inteira e a programação dinâmica adaptada para resolver problemas de mochila 0-1 com desigualdades (3, 0.5)-SRC;

2. Pesquisar a separação de outros cortes, além das desigualdades (3, 0.5)-SRC, para aumentar ainda mais o LB e reduzir, assim, ainda mais o *gap* para uma mais eficiente enumeração de colunas;

3. Pesquisar outras estratégias que reduzam o consumo de memória RAM exigida na otimização do modelo de decomposição (Dantzig-Wolfe) com colunas geradas e enumeradas de modo a encontrar uma solução inteira (a resolução da instância d20200 seria beneficiada);

4. Pesquisar a implementação de inspeção por *pricing* para tornar a enumeração de colunas mais eficiente (a resolução da instância d20200 seria beneficiada);

5. Implementar a técnica de *branching*, transformando o *price-and-cut* em *branch-and-cut-and-price* para resolver instâncias maiores e mais difíceis.

Embora o escopo de pesquisa desta dissertação não abranja estratégias para avaliar e implementar essas melhorias, a conclusão deste trabalho reconhece a possibilidade de investigar melhorias do método defendido. Em caso de instâncias como a d20200, onde o solucionador Gurobi não consegue encontrar a solução inteira ótima por escassez de recursos computacionais, uma abordagem alternativa pode ser a resolução em partes do modelo de decomposição, adicionando gradualmente as colunas geradas e enumeradas de acordo com alguma heurística. Esse experimento também pode ser realizado em máquinas com mais memória RAM disponível ou uma inspeção por *pricing* pode ser adotada.

A adaptação do algoritmo de mochila 0-1 implementado em Pisinger (1997) para considerar as (3, 0.5)-SRCs pode melhorar significativamente a eficiência da enumeração de colunas. A introdução de outros cortes pode ajudar a aumentar mais o LB, acentuando o efeito positivo comprovado pela separação das (3, 0.5)-SRCs.

8
Conclusão

Em síntese, esta dissertação apresenta um método exato *price-and-cut* com separação de desigualdades (3, 0.5)-SRC para viabilizar a enumeração de colunas, buscando resolver o máximo de instâncias da literatura com solução inteira ótima.

O texto deste trabalho apresenta as formulações do Problema de Alocação Generalizada (PAG) já conhecidas, desde a formulação clássica até a formulação de decomposição de Dantzig-Wolfe. A partir da decomposição de Dantzig-Wolfe, a geração de colunas é desenvolvida, recorrendo também à ponderação de Dantzig-Wolfe para a estabilização de Wentges (1997) aplicada em casos de degenerescência das instâncias tratadas. A solução ótima encontrada pela geração de colunas foi adotada como um LB para definir um *gap* de acordo com um UB extraído dos trabalhos Avella (2010) e Michelon (2010). Esse *gap* atendeu a necessidade de restringir o conjunto de colunas enumeráveis, tornando a enumeração de colunas computacionalmente viável para quase todas as instâncias com até 200 tarefas e 20 máquinas.

O algoritmo desenvolvido produziu resultados que confirmam a utilidade da separação de desigualdades (3, 0.5)-SRC para a formulação do PAG, satisfazendo às expectativas traçadas. Apesar da frustração com o resultado de algumas poucas instâncias mais difíceis, nomeadamente d10200 e d20200, o algoritmo apresentado obteve um ótimo desempenho para as demais instâncias.

Em razão de algumas limitações encontradas, esta dissertação indica extensão da pesquisa para trabalhos futuros (Capítulo 7), especificamente a adaptação do algoritmo de Pisinger (1997) para otimização de problemas de mochila 0-1 com cortes, a separação de outras desigualdades que aumentem ainda mais o LB e uma estratégia eficiente para reduzir o consumo de memória RAM na resolução de modelo mestre restrito pelo solucionador adotado (i.e. Gurobi).

Esta dissertação contribui com a análise do impacto positivo da adição de cortes (3, 0.5)-SRC na tentativa de solucionar instâncias pela enumeração de colunas e se aproximar o máximo possível do estado-da-arte. Assim sendo, existe a expectativa de que esse método possa ser melhorado em futuras pesquisas, de modo a encontrar uma solução inteira ótima para cada vez mais instâncias do PAG, tanto instâncias maiores quanto mais difíceis.

Referências Bibliográficas

AVELLA, P.; BOCCIA, M. ; VASILYEV, I.. **A computational study of exact knapsack separation for the generalized assignment problem**. Computational Optimization and Applications, 45:543–555, 04 2010. 1.2, 1.3, 6, 5.1, 6.2, 10, 6.2, 6.2, 6.2, 8

BENJAAFAR, S.; ELHAFSI, M. ; DEVERICOURT, F.. **Demand allocation in multiple-product, multiple-facility make-to-stock production systems**. Management Science, 50(10):1431–1448, 2004. 1.2

BRESSOUD, T. C.; RASTOGI, R. ; SMITH, M. A.. **Optimal configuration for bgp route selection**. In: IEEE INFOCOM 2003. TWENTY-SECOND ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES (IEEE CAT. NO.03CH37428), volumen 2, p. 916–926 vol.2, 2003. 1.2

CATTRYSSE, D.; SALOMON, M. ; WASSENHOVE, L. N. V.. **A set partitioning heuristic for the generalized assignment problem**. European Journal of Operational Research, 72(1):167–174, 1994. 1.2

DANTZIG, G. B.; WOLFE, P.. **The decomposition algorithm for linear programs**. Econometrica, 29(4):767–778, 1961. 3.1

FERLAND, J. A.. **Generalized assignment-type problems a powerful modeling scheme**. In: Burke, E.; Carter, M., editors, PRACTICE AND THEORY OF AUTOMATED TIMETABLING II, p. 53–77, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. 1.1

FISHER, M. L.; JAIKUMAR, R.. **A generalized assignment heuristic for vehicle routing**. Networks, 11(2):109–124, 1981. 1.2

FRELING, R.; HUISMAN, D. ; WAGELMANS, A.. **Models and algorithms for integration of vehicle and crew scheduling**. Journal of Scheduling, 6:63–85, 01 2003. 1.2

FUKASAWA, R.; GOYCOOLEA, M.. **On the exact separation of mixed integer knapsack cuts**. In: PROCEEDINGS OF THE 2007

INTEGER PROGRAMMING AND COMBINATORIAL OPTIMIZATION CONFERENCE, p. 225–239. Springer, 2007. 1.2

JEPSEN, M.; PETERSEN, B.; SPOORENDONK, S. ; PISINGER, D.. **Subset-row inequalities applied to the vehicle-routing problem with time windows**. Oper. Res., 56(2):497–511, Mar. 2008. 4.1

POSTA, M.; FERLAND, J. ; MICHELON, P.. **An exact method with variable fixing for solving the generalized assignment problem**. Computational Optimization and Applications, 52, 07 2012. 1.2, 1.3, 6, 5.1, 6.2, 11, 6.2, 6.2, 6.2, 8

PIGATTI, A.; POGGI, M. ; UCHOA, E.. **Stabilized branch-and-cut-and-price for the generalized assignment problem**. Electronic Notes in Discrete Mathematics, 19:389–395, 06 2005. 1.1, 1.2, 3.4, 3.6

PISINGER, D.. **A minimal algorithm for the 0-1 knapsack problem**. Oper. Res., 45(5):758–767, Oct. 1997. 4.2, 4.3, 4.3, 5.2, 6.2, 7, 8

DE ARAGAO, M. P.; UCHOA, E.. **Integer program reformulation for robust branch-and-cut-and-price algorithms**. In: IN PROCEEDINGS OF THE CONFERENCE MATHEMATICAL PROGRAM IN RIO: A CONFERENCE IN HONOUR OF NELSON MACULAN, p. 56–61, 2003. 1.2

PECIN, D.; PESSOA, A.; POGGI, M. ; UCHOA, E.. **Improved branch-cut-and-price for capacitated vehicle routing**. Mathematical Programming Computation, 9, 06 2016. 1.2

ROSS, G.; SOLAND, R.. **A branch and bound algorithm for the generalized assignment problem**. Mathematical Programming, 8:91–103, 12 1975. 1.2

SAHNI, S.; GONZALEZ, T.. **P-complete approximation problems**. J. ACM, 23(3):555–565, July 1976. 1.2

SAVELSBERGH, M.. **A branch-and-price algorithm for the generalized assignment problem**. Operations Research, 45:831–841, 12 1997. 1.1, 1.2, 2.1

ONCAN, T.. **A survey of the generalized assignment problem and its applications**. Infor, 45:123–141, 08 2007. 1.1, 1.2, 2.1, 2.1

WENTGES, P.. **Weighted dantzig-wolfe decomposition for linear mixed-integer programming**. International Transactions in Operational Research, 4(2):151–162, 1997. 1.2, 4, 3.4, 3.4, 3.6, 6.1, 8

YAGIURA, M.; IBARAKI, T.. **Recent metaheuristic algorithms for the generalized assignment problem**. In: INTERNATIONAL CONFERENCE ON INFORMATICS RESEARCH FOR DEVELOPMENT OF KNOWLEDGE SOCIETY INFRASTRUCTURE, 2004. ICKS 2004., p. 229–237, 2004. 1.2