Universidade Estadual de Campinas
Instituto de Computação

Pedro Henrique Del Bianco Hokama

Algoritmos para Problemas com Restrições de Empacotamento

CAMPINAS
2016

Pedro Henrique Del Bianco Hokama

Algoritmos para Problemas com Restrições de Empacotamento

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Orientador: Prof. Dr. Flávio Keidi Miyazawa

Este exemplar corresponde à versão final da Tese defendida por Pedro Henrique Del Bianco Hokama e orientada pelo Prof. Dr. Flávio Keidi Miyazawa.

CAMPINAS
2016

Agência(s) de fomento e nº(s) de processo(s): FAPESP, 2011/13382-3

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

H689a

Hokama, Pedro Henrique Del Bianco, 1986-
Algorithms for problems with loading constraints / Pedro Henrique Del Bianco Hokama. – Campinas, SP : [s.n.], 2016.

Orientador: Flávio Keidi Miyazawa.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Problema de roteamento de veículos. 2. Problema de empacotamento. 3. Programação por restrições. 4. Programação linear. 5. Programação inteira. 6. Algoritmos. I. Miyazawa, Flávio Keidi,1970-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Algoritmos para problemas com restrições de empacotamento
Palavras-chave em inglês:
Vehicle routing problem
Packing problem
Constraint programing
Linear programing
Integer programming
Algorithms
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Flávio Keidi Miyazawa [Orientador]
Horacio Hideki Yanasse
Reinaldo Morabito Neto
Kelly Cristina Poldi
Luis Augusto Angelotti Meira
Data de defesa: 04-03-2016
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Pedro Henrique Del Bianco Hokama

Algoritmos para Problemas com Restrições de Empacotamento

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa
  IC/UNICAMP
- Prof. Dr. Horacio Hideki Yanasse
  ICT/UFSP
- Prof. Dr. Reinaldo Morabito Neto
  DEP/UFSCAR
- Profa. Dra. Kelly Cristina Poldi
  IMECC/UNICAMP
- Prof. Dr. Luis Augusto Angelotti Meira
  FT/UNICAMP

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 04 de março de 2016

Agradecimentos

Primeiramente agradeço à minha mãe e a minha irmã por todo apoio incondicional ao longo dos anos.
Agradeço ao Flávio por tantos anos de paciência e apoio, tanto nos problemas do doutorado como pessoais.
Agradeço ao Mário e à Mariana por cuidarem de mim.
Agradeço ao Fábio, ao Xein, ao Chineis, à Marcela, ao Esteban, ao Aloisio, ao Vinicius e à Priscila, por tantos anos de amizade.
Agradeço aos colegas e professores Fábio, Evandro, Rafael, Thiago e Eduardo.
Agradeço à Karen pela companhia.
Agradeço aos professores e funcionários do IC.
Agradeço à CAPES, ao CNPQ e à FAPESP pela infraestrutura e financiamento.

Resumo

Nesta tese investigamos classes de problemas com restrições de empacotamento. Três diferentes problemas foram investigados, e algoritmos foram propostos para cada um deles. O primeiro é o problema de roteamento de veículos com restrições de empacotamento, em que um conjunto de veículos parte de um depósito e deve entregar a demanda de itens de todos os clientes. Cada veículo possui um contêiner retangular, e cada cliente deseja receber uma diversidade de itens também retangulares. Consideramos tanto o caso em que contêineres e itens são bidimensionais, como o caso em que eles são tridimensional. Em cada rota realizada por um veículo é preciso encontrar uma forma de empacotar os itens de todos os clientes dentro do contêiner, de modo que a cada visita, a retirada dos itens daquele cliente possa ser realizada sem que os outros itens precisem ser movidos. O objetivo geral do problema é minimizar o deslocamento total dos veículos. O segundo é o problema online de empacotamento de círculos em contêineres. Nesse problema devemos empacotar círculos em recipientes retangulares. Os círculos chegam de forma online, ou seja, cada círculo que chega deve ser empacotado, e não se sabe a priori quais os tamanhos dos círculos que virão. O objetivo é minimizar o número de contêineres utilizados. O terceiro é o problema da mochila bidimensional com conflitos. Nesse problema é dado um conjunto de itens e um recipiente, bidimensionais e retangulares, sendo que cada item possui um valor associado e alguns pares de itens são conflitantes, ou seja, não podem estar simultaneamente dentro do recipiente. O objetivo do problema é escolher um subconjunto de valor máximo dos itens, mas que seja possível empacotar esses itens dentro do recipiente. A tese é composta por três artigos, cada um dedicado a um dos problemas. Em todos eles, diferentes técnicas de projeto de algoritmos foram utilizadas de forma integrada, dentre as principais estão: programação linear inteira, programação por restrições, heurísticas, meta-heurísticas e algoritmos aproximados. Além das contribuições de cada um dos artigos, o conjunto da obra evidencia a eficiência da integração das técnicas citadas, abrindo o caminho para que as metodologias estudadas possam ser aplicadas a outros problemas.

Abstract

In this thesis we investigate classes of problems with loading constraints. Three different problems were investigated, and algorithms were proposed for each one of them. The first is the vehicle routing problem with loading constraints, in which a set of vehicles departs from a depot and have to deliver the items demands of all clients. Each vehicle has a rectangular container (bin), and each client must receive some rectangular items. We consider both cases, in which bin and items are two-dimensional or three-dimensional. In each route, it is necessary to find a packing of all clients' items in the bin. This packing has to be done in such a way that, in each visit, the unloading of the current client's items can be performed without moving the remaining items. The goal is to minimize the total travel distance of the vehicles. The second is the online circle packing problem. In this problem we have to pack circles in rectangular bins. The circles are delivered online, that is, each arrived circle has to be packed, and the sizes of future circles is unkwown. The goal is to minimize the number of used bins. The third is the two-dimensional disjunctively constrained knapsack problem, in which a bin and a set of items, rectangular and two-dimensional, are given. Each item has an associated value and some pairs of items are forbidden to be packed together. The goal is to choose a subset of items, with maximum value, that can be packed in the given bin. The thesis is composed of three articles, each one dedicated to one of the problems. Different techniques of algorithm design were used combined, such as: integer linear programing, constraint programming, heuristics, meta-heuristics and approximated algorithms. Besides the contribution of each individual article, efficiency of the combination of cited techniques is shown, considering that they might provide a good strategy for solving other problems.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Problemas de corte e empacotamento aparecem em diversos contextos dentro das empresas. Os mais comuns são no aproveitamento de matérias primas, como cortes de madeira, isopor, couro, tecido, placas de aço etc. Ou ainda na estocagem e transporte de produtos em contêineres. Além de surgirem em problemas não tão diretos, como alocação de dispositivos em uma placa eletrônica, ou escalonamento de recursos em computadores, entre outros. Neste trabalho investigamos alguns problemas que exigem como subproduto, encontrar empacotamentos com características particulares. Outro problema bem conhecido, o de roteamento de veículos, é considerado. Um bom planejamento de transporte, armazenagem e um bom aproveitamento de materiais podem diminuir significativamente os custos de uma empresa. Nesta seção descrevemos brevemente os problemas mais importantes tratados nesta tese, começando pelos problemas mais essenciais até problemas mais complexos. Também descrevemos brevemente as principais técnicas utilizadas.

1.1 Descrição dos Problemas

Nesta seção descrevemos os problemas que são abordados nesta tese. Alguns dos problemas se caracterizam por serem compostos por outros sub-problemas, por isso definimos primeiramente os problemas bases, para então definir os problemas mais complexos. Na seção 1.1.1 definimos o problema de roteamento de veículos (*Vehicle Routing Problem* - VRP), seguido pelo problema de empacotamento ortogonal com restrições de descarregamento (*Orthogonal Packing Problem with Unloading Constraints* - OPPUL). Combinados, esses dois problemas resultam no problema de roteamento de veículos com restrições de empacotamento (*Vehicle Routing Problem with Loading Constraints*), investigados tanto na sua versão bidimensional (2L-CVRP) quanto na versão tridimensional (3L-CVRP). Na seção 1.1.2 descrevemos o problema de empacotamento de círculos em um único contêiner (*Single Bin Circle Packing Problem*), que é utilizado posteriormente na definição do problema online de empacotamento de círculos (*Online Circle Packing Problem*). Na seção 1.1.3 descrevemos o problema da mochila bidimensional com conflitos (*Two-dimensional Disjunctively Constrained Knapsack Problem* - 2D-DCKP).

1.1.1 Problema de Roteamento de Veículos com Restrições de Empacotamento

Primeiramente definimos o problema de roteamento de veículos (VRP) no qual é dado um grafo, uma função de custo nas arestas e um número inteiro de veículos. Os veículos partem e devem retornar ao estacionamento representado por um dos vértices do grafo. De fato o VRP é uma família de problemas com muitas variantes, diferentes restrições e objetivos, aqui iremos considerar aquela cujo o objetivo é encontrar um conjunto de rotas de custo mínimo, com início e fim no estacionamento de forma que todos os outros vértices do grafo sejam visitados exatamente uma vez. O VRP é vastamente estudado na literatura por se tratar de um problema desafiador e altamente aplicável no cotidiano. A definição de rota considera uma sequência que define a ordem em que os vértices serão visitados a partir do depósito. Assim, mesmo que duas rotas contemplem o mesmo conjunto de clientes, estas rotas podem ter custos diferentes. Quando o número de veículos é limitado a um, e devemos portanto encontrar uma única rota, temos o clássico problema do caixeiro viajante (*Travelling Salesman Problem* - TSP).

Existem inúmeras variantes para o problema de roteamento de veículos. O leitor interessado poderá consultar o livro editado por Toth e Vigo [90], *The Vehicle Routing Problem*, que contém variantes, técnicas, modelos, algoritmos e heurísticas para o VRP.

O segundo problema a ser definido é o do empacotamento ortogonal com restrições de descarregamento (*Orthogonal Packing Problem with Unloading Constraints* - OPPUL), nesse problema recebemos uma lista de conjuntos de retângulos bidimensionais. Também é fornecido um contêiner bidimensional de dimensões fixas, com uma abertura em apenas um dos lados. O objetivo é encontrar um empacotamento dos itens dentro do contêiner, de forma que, nenhum item ultrapasse os limites do contêiner; dois itens quaisquer não se sobreponham; deve ser possível retirar os conjuntos de retângulos na ordem definida na lista de entrada. Essa retirada deve ser feita em um único movimento na direção da porta, sem que os demais itens precisem ser rearranjados.

Foi investigado tanto a versão bidimensional (2OPPUL) descrita, como a versão tridimensional (3OPPUL) em que os retângulos e o contêiner são substituídos por caixas tridimensionais. Consideramos um empacotamento ortogonal, em que os itens só podem ser empacotados paralelos aos lados do contêiner e não são permitidas rotações dos itens. A figura 1.1 mostra um exemplo de um contêiner carregado que respeita uma ordem de remoção dos itens. As caixas mais escuras devem ser descarregadas antes das mais claras, e a saída deve ser realizada na direção indicada pela seta.

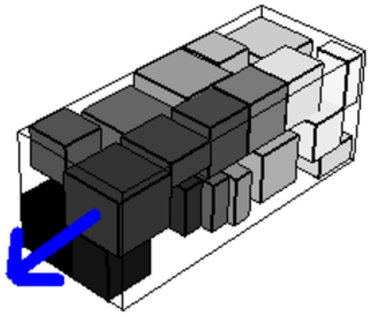


Figura 1.1: Exemplo de um empacotamento tridimensional que respeita restrições de descarregamento, no qual há apenas um lado para descarregar os itens.

Finalmente, no problema de roteamento de veículos com restrições de empacotamento (Vehicle Routing Problem with Loading Constraints - DL-VRP), é dado um conjunto de veículos que partem de um depósito entregando itens aos seus clientes e ao final retornam ao depósito. Cada veículo possui um contêiner retangular, e cada cliente deseja receber uma diversidade de itens também retangulares. Em cada rota realizada por um veículo é preciso encontrar uma forma de empacotar os itens de todos os clientes dentro do contêiner, de modo que a cada visita, a retirada dos itens daquele cliente possa ser realizada sem que os outros itens precisem ser movidos. O objetivo geral do problema é minimizar o deslocamento total dos veículos. Nesse problema além de resolver o problema de roteamento de veículos ainda é necessário encontrar, para cada rota, se existir, um empacotamento que respeite as restrições de descarregamento (2OPPUL, 3OPPUL) dos itens de seus clientes dentro do contêiner.

Na versão capacitada do problema (Capacitated Vehicle Routing Problem with Unloading Constraints - DL-CVRP) os itens possuem um peso associado e cada veículo possui uma capacidade de peso máximo associado. Consideraremos essa versão do problema. A figura 1.2 exemplifica, para o caso bidimensional, como seria uma solução viável. O círculo branco central representa o depósito e os demais círculos representam os clientes. O exemplo apresenta quatro rotas que partem e chegam no depósito. Dentro de cada uma destas rotas, temos um empacotamento dos itens em cada contêiner. O descarregamento é feito pela parte superior do contêiner, os espaços em branco são vazios. Próximo a cada círculo, pintados da mesma cor que o interior deles, estão os itens que cada cliente deve receber.

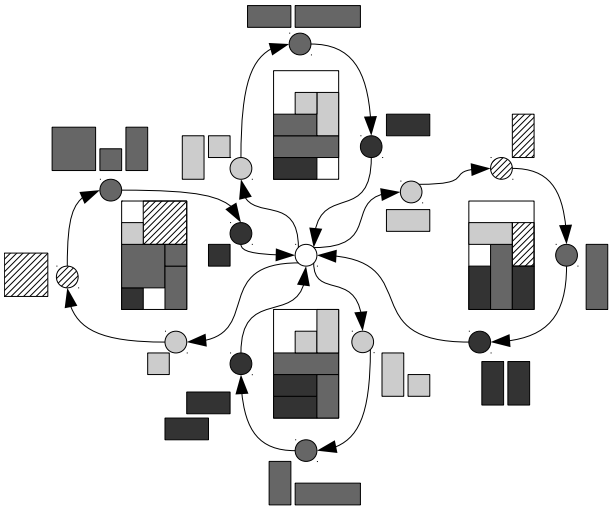


Figura 1.2: Exemplo de solução para o caso bidimensional.

1.1.2 Problema de empacotamento de círculos

Primeiramente é necessário definir o problema de empacotamento de círculos em um único contêiner. Nesse problema são dados um conjunto de círculos e um contêiner retangular, o objetivo é encontrar um empacotamento desses círculos no contêiner ou provar que tal empacotamento não existe. Os raios dos círculos são números racionais e podem ser empacotados em qualquer posição do contêiner. Para tratar esse problema foi feita uma discretização do contêiner.

O segundo é o problema online de empacotamento de círculos. Nesse problema, a instância é dada de forma online, ou seja, os círculos só são conhecidos por partes, e não se sabe a priori a quantidade e dimensões deles. Os círculos devem ser empacotados em contêineres retangulares a medida que chegam e não podem ser rearranjados posteriormente. O objetivo do problema é minimizar o número de contêineres utilizados.

1.1.3 Problema da Mochila Bidimensional com Conflitos

No problema da mochila bidimensional com conflitos são dados um conjunto de itens e um recipiente, todos retangulares, sendo que cada item possui um valor associado e alguns pares de itens são conflitantes, ou seja, não podem estar simultaneamente dentro do recipiente. O objetivo do problema é escolher um subconjunto, de valor máximo, dos itens que possam ser empacotados dentro do recipiente.

O problema da mochila também é um clássico da literatura e largamente utilizado como exemplo de problema NP-difícil. No problema estudado, além do clássico problema da mochila, é necessário resolver como sub-problema o empacotamento ortogonal bidimensional, que também é NP-difícil.

Além disso também estendemos o problema para considerar os *carregamentos completos*, nos quais os itens são divididos em subconjuntos, que devem ser completamente carregados ou deixados completamente fora da solução.

1.2 Técnicas

Nesta seção apresentamos resumidamente algumas das técnicas que foram utilizadas na resolução dos problemas. Devido a complexidade dos problemas investigados, é de se esperar que essas e outras técnicas sejam combinadas, na elaboração de algoritmos rápidos. Outras técnicas como heurísticas, meta-heurísticas, são apresentadas nos próximos capítulos na medida do necessário.

1.2.1 Programação por Restrições

Programação por restrições é uma técnica de programação declarativa que tem se mostrado bastante promissora nas últimas décadas, especialmente para problemas em que uma formulação em programação linear inteira é grande ou complicada demais. Essa técnica tem se mostrado bastante eficiente na resolução de grandes problemas combinatórios, especialmente em planejamento e escalonamento. Possui uma forte fundamentação teórica e também atrai o interesse comercial e prático, particularmente em áreas de modelagem de problemas de otimização e de problemas de satisfação.

Restrições estão presentes na maioria das áreas do conhecimento, elas formalizam as relações entre o mundo físico e suas modelagens matemáticas de forma natural e transparente. Uma restrição é simplesmente uma relação lógica entre diversas variáveis. Ela limita os possíveis valores que as variáveis podem assumir simultaneamente, representando uma informação parcial sobre as variáveis do modelo. A principal característica das restrições é sua forma declarativa, ou seja, elas especificam qual relação existe entre as variáveis sem precisar descrever um procedimento computacional para garantir essa relação.

As primeiras ideias de programação por restrições podem ser encontradas na área de Inteligência Artificial nas décadas de sessenta e setenta. A Programação Lógica foi um de seus precursores, sendo que esta e outras formas de programação declarativa em geral, utilizam a ideia de que o usuário estabelece *o que* deve ser revolvido, e não *como* resolver.

Satisfação de Restrições

Problemas de Satisfação de Restrições (CSP) são definidos da seguinte forma:

- um conjunto de variáveis $X = \{x_1, \ldots, x_n\}$,
- um conjunto finito $Dom(x_i)$ para cada variável x_i, com os valores possíveis de serem atribuídos a cada variável. Chamamos $Dom(x_i)$ de domínio de x_i,
- um conjunto de restrições $\mathcal{C}(X)$ para os valores que as variáveis podem assumir simultaneamente.

Note que os valores não necessariamente precisam ser inteiros consecutivos, ou mesmo numéricos. Uma solução para um CSP é uma atribuição de valores de cada domínio para cada

uma das respectivas variáveis, de forma que todas as restrições são satisfeitas simultaneamente. Nós podemos encontrar:

- somente uma solução, com nenhuma preferência sobre as outras,
- todas as soluções,
- uma solução boa ou ótima, dada uma certa função objetivo definida sobre as variáveis.

Soluções para um CSP podem ser encontradas explorando sistematicamente possíveis atribuições de valores às variáveis.

Restrições. As restrições são representadas por uma expressão envolvendo um subconjunto de variáveis. Formalmente, uma restrição $C_{1,\ldots,n}$ entre as variáveis x_1, ..., x_n é qualquer subconjunto das possíveis combinações de valores de x_1, ..., x_n, isto é, $C_{ijk\ldots} \subseteq Dom(x_1) \times \ldots \times Dom(x_n)$. O subconjunto define as combinações de valores que a restrição admite. As restrições práticas não são representadas dessa forma, mas essa definição enfatiza que as restrições não precisam ser expressões simples, de fato elas também não precisam ser lineares. Alguns exemplos de restrições seriam, $x_1 \neq x_2$, $2x_1 = 10x_2 + x_3$ e $x_1x_2 < x_3$.

Técnica de Busca. A técnica se baseia em uma busca exaustiva, utilizando um *backtracking* associado com uma poderosa propagação de restrições, que reduzem significativamente os domínios das variáveis. Dois níveis de propagação são realizados, a propagação inicial, que vale para toda a árvore do *backtracking*, e a propagação durante a busca, que elimina temporariamente valores do domínio a cada escolha feita. O domínio é restaurado caso a escolha se mostre inviável, sendo realizada uma nova escolha.

A propagação analisa cada restrição individualmente, removendo do domínio escolhas que não correspondam a atribuições possíveis, ou seja, que violem a restrição. Como a eliminação de alguns valores do domínio de uma variável podem implicar na redução do domínio de outras variáveis, repete-se o processo até que não se consiga mais reduzir nenhum dos domínios.

Para mais informações sobre programação por restrições, referenciamos os trabalhos de Gaschnig [39], Haralick et. al. [44] e Sabin et. al [83].

1.2.2 *Branch-and-bound*

O método *Branch-and-bound* é bastante utilizado para buscar soluções em diversos problemas de otimização. O método visa uma enumeração das soluções possíveis, fazendo isso de forma sistemática baseada no paradigma de "Divisão e Conquista", procurando manter a eficiência na busca. Utilizando limitantes para o custo de uma solução parcial, o método é capaz de encontrar uma solução ótima sem a necessidade de explorar todo o espaço de busca, *i.e.*, todas as soluções possíveis.

Para exemplificar o funcionamento do método de *Branch-and-bound*, considere o seguinte problema:

$$\begin{aligned} \min \quad & cx & (1.1)\\ \text{sujeito a} \quad & x \in E. & (1.2) \end{aligned}$$

Nesse problema, E é o conjunto de todas as soluções viáveis. Agora E é dividido em uma coleção finita de subconjuntos $E_1, \ldots, E_k$, tal que $E_1 \cup \ldots \cup E_k = E$. E então resolvemos um subproblema para cada $i = 1, \ldots, k$:

$$\begin{aligned} \min \quad & cx & (1.3)\\ \text{sujeito a} \quad & x \in E_i. & (1.4) \end{aligned}$$

Então é escolhida a melhor solução obtida entre as encontradas nos subproblemas. Cada subproblema pode ser resolvido utilizando o mesmo método, dividindo em subproblemas e resolvendo problemas menores. Essa é a parte da divisão (*branch*), e ela pode ser entendida como uma árvore de subproblemas. Na figura 1.3, E foi dividido em E_1 e E_2, posteriormente E_2 foi dividido em E_3 e E_4.

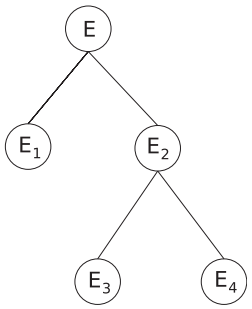


Figura 1.3: Árvore de subproblemas.

Além da fase de divisão, também é necessário um algoritmo eficiente capaz de computar para todo E_i um limitante inferior $\underline{z}(E_i)$ da solução ótima daquele subproblema, ou seja:

$$\underline{z}(E_i) \leqslant \min_{x \in E_i} cx.$$

Uma forma tradicional de se obter esse limitante inferior é utilizar a relaxação linear do problema. Da mesma forma, seja $\overline{z}(E_i)$ um limitante superior para a solução ótima de E_i, ou seja:

$$\overline{z}(E_i) \geqslant \min_{x \in E_i} cx.$$

Esse limitante pode ser dado por uma solução viável do problema, obtida de forma heurística. Dessa forma podemos limitar o espaço de busca, podando os subproblemas que não podem conter a solução ótima. A figura 1.4 exemplifica a poda por limitante. Cada nó E_i da árvore tem indicado o valor de $\overline{z}(E_i)$ na parte superior e $\underline{z}(E_i)$ na parte inferior. Como $20 = \overline{z}(E_1) <$

$\underline{z}(E_2) = 21$, e se tratando de um problema de minimização, podemos podar o nó E_2, ou seja, não precisamos mais explorar os subproblemas originários de E_2, pois a solução ótima desse subproblema possui um custo maior que a solução viável já obtida em E_1

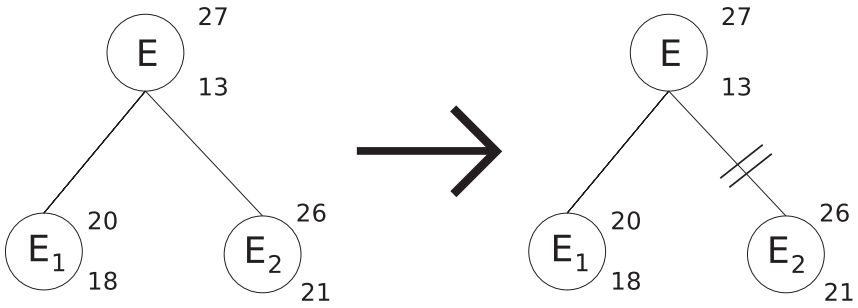


Figura 1.4: Poda por limitante.

Essa parte do método é chamada de poda (*bound*), na qual limitamos o espaço de busca do problema, a eficiência dessa parte está diretamente ligada com os limitantes, quanto mais justos forem, melhor será o desempenho do algoritmo. Infelizmente nem sempre é fácil encontrar limitantes justos para os problemas de otimização.

Outras podas (além da poda por limitante) e outras formas de exploração também são utilizadas nos algoritmos de *Branch-and-Bound*, para conhecer melhor sobre o método consulte as referências [96, 97].

Branch-and-Cut. O método de *Branch-and-Cut* é uma técnica para resolver problemas lineares inteiros ou inteiros mistos, e consiste em um método híbrido de planos de corte e do método *Branch-and-Bound*. Resolvendo a cada nó da árvore de *Branch-and-Bound*, uma relaxação linear do problema.

Na ideia básica do método *Branch-and-Cut*, quando o número de desigualdades é muito grande para ser adicionado ao modelo e resolvidas eficientemente, algumas classes de desigualdades válidas podem não ser adicionadas no início da relaxação linear. A maioria dessas restrições não estará relacionada com a solução ótima, ou dificilmente será violada. Ou ainda mesmo que o modelo inicial descreva uma solução viável, existem restrições que fortalecem a formulação, eliminando muitas soluções fracionárias e fazendo o algoritmo convergir mais rapidamente, essas desigualdades não precisam ser colocadas no modelo a priori. No decorrer do algoritmo, se uma solução ótima para uma relaxação linear é inviável, um subproblema, chamado de problema da separação, é resolvido para tentar identificar as inequações violadas. Se uma ou mais inequações são encontradas, algumas são adicionadas à relaxação linear para eliminar a solução inviável e então a nova relaxação é re-otimizada. A ramificação (*branch*) ocorre quando nenhuma desigualdade violada é encontrada. O *Branch-and-Cut* é uma generalização do *Branch-and-Bound* com relaxações lineares, permite que a separação e corte sejam aplicados na árvore de *Branch-and-Bound*.

1.3 Resultados e Organização da Tese

Os problemas descritos são difíceis de serem resolvidos na prática, em geral são abordados como um todo, utilizando algum algoritmo em particular, normalmente programação linear inteira ou meta-heurísticas. Nesta tese dividimos os problemas em sub-partes e abordamos cada parte com uma técnica que gera melhores resultados. De forma geral os problemas aqui apresentados foram divididos em um problema de otimização, resolvido com programação linear inteira; e um problema de decisão, resolvido com programação por restrições. Mas outras abordagens também foram utilizadas em partes específicas.

Os métodos propostos para os problemas descritos estão apresentados em três artigos que compõem esta tese. Cada capítulo corresponde a um dos artigos. Os dois primeiros foram publicados em importantes revistas internacionais [53, 55]. e o terceiro artigo aguarda sua publicação. A seguir, descrevemos brevemente alguns dos resultados apresentados em cada um destes artigos.

O primeiro artigo, *A branch-and-cut approach for the vehicle routing problem with loading constraints*, foi publicado na revista *Expert Systems With Applications*. Nesse artigo investigamos o problema de roteamento de veículos com restrições de empacotamento. Foram estudados tanto a versão bidimensional quanto a versão tridimensional do problema. Superficialmente falando, dividimos o problema em uma parte de roteamento, resolvendo-a pelo método *branch-and-cut*, e uma parte de empacotamento, que foi abordada como um problema de decisão utilizando programação por restrições. Essas partes se retroalimentam para garantir uma solução viável ótima. Dessa forma propomos neste artigo um novo algoritmo *branch-and-cut* exato. Além disso propusemos heurísticas e algoritmos exatos para o problema de empacotamento com restrições de descarregamento. Obtivemos resultados competitivos com os da literatura, conseguindo resultados ótimos em um maior número de instâncias e em um tempo menor. Esse artigo contou com a co-autoria do Prof. Dr. Flávio K. Miyazawa, e do Prof. Dr. Eduardo C. Xavier, ambos do Instituto de Computação da Universidade Estadual de Campinas (IC - UNICAMP).

O segundo artigo, *A bounded space algorithm for online circle packing*, foi publicado no *Information Processing Letters*. Nesse artigo investigamos o problema online de empacotamento de círculos. Propusemos nesse artigo um algoritmo de espaço limitado 2.4394-competitivo e demonstramos que qualquer algoritmo desse tipo pode ser no melhor caso 2.2920-competitivo. Para obter esse limitante também realizamos uma divisão do problema, primeiramente em uma parte que escolhe as combinações de círculos de valor máximo, seguida por uma parte que resolve o problema de decisão de encontrar um empacotamento destes círculos no contêiner. A primeira parte é resolvida por um programa linear inteiro, enquanto a segunda através de um algoritmo de programação por restrições. Esse artigo teve a co-autoria do Prof. Dr. Flávio K. Miyazawa, e do Prof. Dr. Rafael C. S. Schouery, ambos do Instituto de Computação da Universidade Estadual de Campinas (IC - UNICAMP).

O terceiro artigo, *Two-dimensional Disjunctively Constrained Knapsack Problem: Heuristic and Exact approaches*, foi submetido na revista *Computers & Industrial Engineering* e nele investigamos o problema da mochila bidimensional com conflitos. Nesse trabalho propusemos

uma heurística e um novo algoritmo *branch-and-cut* para o problema. Para isso também dividimos o problema em duas partes principais, a primeira resolve o problema da mochila com conflitos, por uma modelagem em programação linear inteira, na segunda parte é verificado a viabilidade dessa configuração de itens através de um algoritmo de programação por restrições. Obtivemos resultados muito superiores às formulações anteriores. Esse artigo teve a co-autoria do Prof. Dr. Flávio K. Miyazawa, Prof. Dr. Rafael C. S. Schouery, ambos do Instituto de Computação da Universidade Estadual de Campinas (IC - UNICAMP), e do Prof. Dr. Thiago A. de Queiroz, do Instituto de Matemática e Tecnologia da Universidade Federal de Goiás - Campus Catalão.

Todos os algoritmos propostos foram testados com instâncias da literatura ou adaptadas e mostram a eficiência das metodologias propostas. Além de cada uma dessas contribuições, foi possível evidenciar a pertinência e eficiência de se agregar diferentes técnicas e modelagens no esforço de solucionar problemas complexos. Os algoritmos envolveram programação linear inteira, programação por restrições, heurísticas, meta-heurísticas, algoritmos aproximados entre outros.

Capítulo 2

A Branch-and-Cut Approach for the Vehicle Routing Problem with Loading Constraints *

ABSTRACT

In this paper we describe a branch-and-cut algorithm for the vehicle routing problem with unloading constraints. The problem is to determine a set of routes with minimum total cost, each route leaving a depot, such that all clients are visited exactly once. Each client has a demand, given by a set of items, that are initially stored in a depot. We consider the versions of the problem with two and tri dimensional parallelepiped items. For each route in a solution, we also need to construct a feasible packing for all the items of the clients in this route. As it would be too expensive to rearrange the vehicle cargo when removing the items of a client, it is important to perform this task without moving the other client items. Such packings are said to satisfy unloading constraints.

In this paper we describe a branch-and-cut algorithm that uses several techniques to prune the branch-and-cut enumeration tree. The presented algorithm uses several packing routines with different algorithmic approaches, such as branch-and-bound, constraint programming and metaheuristics. The careful combination of these routines showed that the presented algorithm is competitive, and could obtain optimum solutions within significantly smaller computational times for most of the instances presented in the literature.

KEYWORDS. Vehicle routing. Two-dimensional packing. Three-dimensional packing. Branch-and-cut. Combinatorial Optimization.

*This work was partially supported by CNPq (grants 311499/2014-7, 306358/2014-0, 477692/2012-5) and FAPESP (grant 2011/13382-3).

2.1 Introduction

Several problems in transportation systems involve route planning for vehicles with containers attached and accommodation of cargo into these containers. The route planning problem and the packing problem are well known problems in the research literature, and they were largely explored separately. However, in recent years there has been some interest in considering both problems combined, leading to better global solutions.

In the Vehicle Routing Problem with D-Dimensional Unloading Constraints (DL-CVRP), clients have a demand for goods stored into a depot, represented by D-dimensional parallelepipeds, and k vehicles must be used to deliver these goods. Each travel from the depot to a client, or from a client to a next one has a cost. The problem is to find k routes leaving the depot, one route for each vehicle, such that all clients are visited exactly once, and such that the items of clients of a route can be packed in the vehicle's container. The objective function of the problem is to minimize the total cost of the routes. As it would be too expensive to rearrange the cargo at each visit, we add a condition that the goods to be unloaded in a client must be removed without moving the remaining goods, these are the so called *unloading constraints*. We consider two versions of the problem, one with two dimensional items and another with three-dimensional ones.

The literature in vehicle routing problem is extensive, with different variants including practical constraints. In the last decade, there has been some interest in variants that include two- and three-dimensional packing constraints. Iori et al. [58] were the first to present an exact branch-and-cut approach for the capacitated vehicle routing problem with two-dimensional unloading constraints (2L-CVRP). To separate infeasible routes, these authors used known separations routines for the CVRP. To separate routes that lead to infeasible packings, they used an adaptation of the exact algorithm, presented in [68], to satisfy unloading constraints. Following this work, Azevedo et al. [2] also presented an exact method for 2L-CVRP, using a different set of separation routines for the CVRP, made available by Lysgaard et al. [66]. To cut routes that lead to infeasible packings, these authors also used an adaptation of the algorithm presented by Martello et al. [68]. Due to the difficulty of solving this problem exactly, several heuristics were also proposed. Gendreau et al. [41] presented a tabu search method to the 2L-CVRP. Fuellerer et al. [37] employed an ant colony method. Zachariadis et al. [101] introduced a guided tabu search method. Duhamel et al. [28] presented a GRASP approach for the case without unloading constraints. Silveira and Xavier [21] considered the pick-up and delivery version of the problem. In this case both loading and unloading constraints must be taken into account when generating a route. They presented an exact algorithm and also a GRASP heuristic for the problem. Approximation algorithms for the associated packing problem that considers unloading constraints were proposed by Silveira et. al. [22, 23, 24].

The heterogeneous fleet variant with unloading constraint , where vehicles containers have different sizes, was considered by Wei et. al. [94], who present an adaptive variable neighbourhood search metaheuristic for the three dimensional case. Dominguez et. al. [27] presented a randomised multi-start biased metaheuristic for the two dimensional case.

The capacitated vehicle routing problem with three-dimensional unloading constraints (3L-

CVRP) was first considered by Gendreau et al. [40]. They presented a tabu search method to solve the problem. Junqueira et al. [61] presented an exact model for the 3L-CVRP via an integer linear programming model with practical constraints, namely stability, multidrop and load-bearing strength. The formulation was able to solve instances of moderate size. The constraints considered differ from the ones here. For more references on routing and loading problems we refer to a survey by Iori et al. [57].

In the 2L-CVRP and 3L-CVRP problems we face a packing subproblem of determining if a set of items of one route can be packed in a bin. This is the so called Orthogonal Packing Problem (OPP). In this problem it is given a set of items and a bin with bounded dimensions, and the objective is to find a placement of these items in the bin, or prove that it is unfeasible. When the items and the bins are two-dimensional objects (resp., three-dimensional), we have the two-dimensional orthogonal packing problem - 2OPP (resp. the three-dimensional orthogonal packing problem - 3OPP). Clautiaux et al. [17] presented an efficient method for solving the 2OPP using Constraint Programming. Their work was further extended to include new bounds by Mesyagutov et al. [70], and to consider the three-dimensional case by Mesyagutov et al. [71]. Recently the problem was considered with the unloading constraint by Côté et al. [20], who presented an exact algorithm using branch-and-cut and a set of lower bounds.

The Constraint Programming (CP) paradigm has been proved to be a very efficient method for solving many different problems [82]. This paradigm was well known by other areas but just in the last decades was rediscovered by researchers in the combinatorial optimization community. The Integer Linear Programming (ILP) is probably the most used method for solving combinatorial optimization problems [95]. The use of both, CP and ILP together is a hot topic and has obtained good results for many problems [56].

In this paper we propose an exact branch-and-cut algorithm to solve the 2L-CVRP and the 3L-CVRP, following the branch-and-cut approach presented in [2, 58], using more elaborate cutting plane routines to cut not only routes whose items cannot be packed in one bin, using CVRP routines [66], but also fast routines that check the feasibility of packings for sub-routes. These cuts are obtained using algorithms to solve the OPP. To this purpose, we tested a number of original and adapted algorithms, and also sophisticated lower bounds that can prove that a set of items cannot be packed in a bin. Besides the adaptation of the exact packing algorithm in [68], we also adapted and improved the constraint programming algorithm presented by [17]. We also present new heuristics, one is based on the Bottom-Left heuristic, and another one is a BRKGA metaheuristic.

The declarative approach of the constraint programming technique allowed us to obtain a packing algorithm for which practical constraints are easily to be incorporated, as opposed to the branch and bound approach used in [58], such as balancing constraints and weight distribution [11], grouping items [11] and more specific application constraints.

The efficiency of the proposed algorithms are evaluated by an experimental analysis with instances from the literature, comparing them with other algorithms from the literature. The careful application of separation routines, not only made possible to obtain solutions for larger instances, but also to deal with the three-dimensional problem version, which previous experi-

mental results showed difficulty to solve.

The proposed use of constraint programming with integer linear programming models, can be applied to other practical variants, as the multi-depot [64] and cross-docking [75] vehicle routing problems, among others.

2.2 Orthogonal Packing Problem With Unloading Constraints

In this section we formally describe the Orthogonal Packing Problem with Unloading Constraint (OPPUL). We present exact algorithms, some heuristics, and lower bounds for the two and three-dimensional version of this problem.

2.2.1 Problem Description

The orthogonal packing problem with unloading constraint can be defined as follows: It is given a D-dimensional container B of dimensions $(W^1, \ldots, W^D)$ with total volume $\mathcal{V}(B) = \prod_{d=1}^{D} W^d$, where $W^d \in \mathbb{Z}^+$, $1 \leqslant d \leqslant D$; n sets of D-dimensional items $(I_1, \ldots, I_n)$, let $I = \bigcup_{v=1}^{n} I_v$. Each item $i \in I_v$ has dimensions $(w_i^1, \ldots, w_i^D)$, where $w_i^d \in \mathbb{Z}^+$. The volume of an item i is denoted by $\mathcal{V}(i) = \prod_{d=1}^{D} w_i^d$ and the volume of a set of items I is denoted by $\mathcal{V}(I) = \sum_{i \in I} \mathcal{V}(i)$. The problem is to find a *packing* $\mathcal{P}_I$ of the items I in the bin B that respects the *unloading constraints* in the direction of the last dimension D.

More precisely, a packing $\mathcal{P}_I$ of items I in a container $B = (W^1, \ldots, W^D)$ that satisfy unloading constraints is a function $\mathcal{P}_I : I \to [0, W^1) \times \ldots \times [0, W^D)$ such that:

(i) The packing must be orthogonal, i.e. the edges of the items must be parallel to the respective container's edges.

(ii) The packing must be oriented, i. e., the items must be packed in the original orientation given in I.

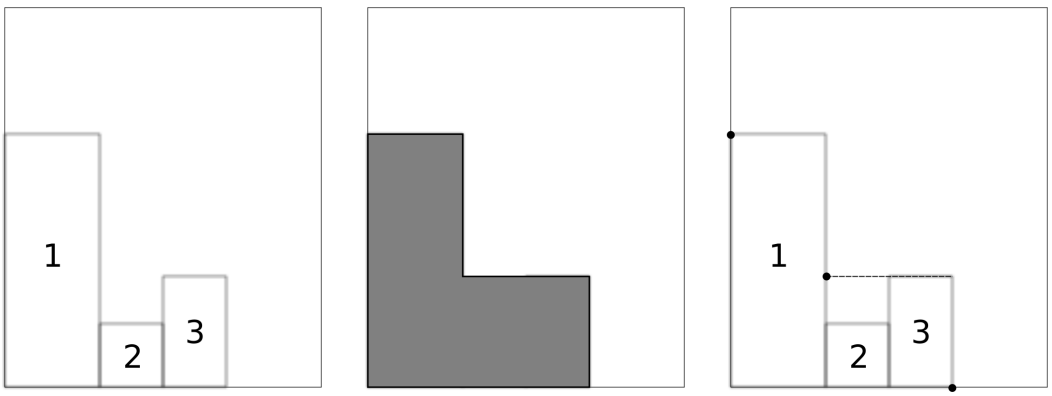
(iii) Items of I must be packed within the container's boundaries. That is, if the position where the item is packed is given by $\mathcal{P}_I(i) = (x_i^1, \ldots, x_i^D)$, for each $i \in I$, then

$$0 \leqslant x_i^d \leqslant x_i^d + w_i^d \leqslant W^d, \text{ for } 1 \leqslant d \leqslant D. \tag{2.1}$$

(iv) Items must not overlap. That is, if the region occupied by the item i is given by $\mathcal{D}(i) = [x_i^1, x_i^1 + w_i^1) \times \ldots \times [x_i^D, x_i^D + w_i^D)$ then

$$\mathcal{D}(i) \cap \mathcal{D}(j) = \varnothing, \text{ for all pairs } i \neq j \in I. \tag{2.2}$$

(v) Items belonging to a set I_v are not blocked by any item belonging to a set I_u if $u > v$. That is, if item i must be unloaded before item j, j can not be packed in the region between i and the end of the container, in the unloading dimension D. More precisely consider the region that includes the item i and its way to the exit of the container defined by $\mathcal{D}^e(i) = \bigtimes_{d=1}^{D-1} [x_i^d, x_i^d + w_i^d) \times [x_i^D, W^D)$. A packing $\mathcal{P}_I$ of $I = I_1 \cup \ldots \cup I_n$ in the bin B respects the



(a) Packing (b) Envelope (c) Corner Points

Figura 2.1: (a) Example of a two-dimensional packing, (b) its envelope and (c) the corner points for this packing.

unloading constraints if it satisfy

$$\mathcal{D}^e(i) \cap \mathcal{D}(j) = \varnothing \quad \text{for all } i \in I_v \text{ and } j \in I_u \text{ with } 1 \leqslant v < u \leqslant n, \tag{2.3}$$

Definitions

An important concept used in several algorithms for packing problems is the *envelope* of a packing or a partial packing. Let I be a set of items, each item $i \in I$ with dimensions $(w_i^1, \ldots, w_i^D)$, and $\mathcal{P}_I$ a packing of these items in a bin of dimensions $(W^1, \ldots, W^D)$. An envelope of $\mathcal{P}_I$ is the region defined by

$$S(\mathcal{P}_I) = \{(x^1, \ldots, x^D) \in \mathbb{R}_+^D : \exists i \in I \text{ with } x^1 < x_i^1 + w_i^1 \wedge \ldots \wedge x^D < x_i^D + w_i^D\}.$$

The complement of the envelope $\bar{S}(\mathcal{P}_I)$ is the region of the Bin B that is not in the envelope. The complement of the envelope is often considered as a feasible region to pack new items in different algorithms. The volume of the envelope $S(\mathcal{P}_I)$ and its complement $\bar{S}(\mathcal{P}_I)$ is denoted respectively by $\mathcal{V}(S(\mathcal{P}_I))$ and $\mathcal{V}(\bar{S}(\mathcal{P}_I))$. Figure 2.1 shows an example of a two-dimensional packing and it's envelope.

Another important concept is the set of *Corner Points* $\hat{C}(\mathcal{P}_I)$, which is defined by,

$$\hat{C}(\mathcal{P}_I) = \{(x^1, \ldots, x^D) \in \bar{S}(\mathcal{P}_I) : \nexists (x'^1, \ldots, x'^D) \in \bar{S}(\mathcal{P}_I) \backslash \{(x^1, \ldots, x^D)\}, x'^1 \leqslant x^1 \wedge \ldots \wedge x'^D \leqslant x^D\}.$$

Figure 2.1c presents the set of corner points for the packing, the black dots are the corner points.

2.2.2 Two-dimensional Orthogonal Packing Problem with Unloading Constraints

To solve the Two-dimensional Orthogonal Packing Problem with Unloading Constraints (2OPPUL), we consider two different exact algorithms found in the literature, which we modified to include unloading constraints. The first is the OneBin algorithm presented by Martello et al. [68]. The second is a constraint programming algorithm presented by Clautiax et al. [17]. Both algorithms were modified to consider the customer's order of visit and the associated unloading constraints when generating a feasible packing.

To make the notation clear, when considering the Two-dimensional Orthogonal Packing Problems the Bin has dimensions (W, H) and each item i has dimensions (w_i, h_i) and it is packed in the position (x_i, y_i) in a feasible solution.

Branch-and-Bound algorithm

The algorithm OneBin works as follows: let I be the set of all items, J be a set of packed items and $\mathcal{P}_J$ a packing of these items in J. Let $\hat{C}(\mathcal{P}_J)$ be the set of corner points defined by $\mathcal{P}_J$ and let $\bar{J} = I \backslash J$ be the set of items not packed yet. At each iteration, $\hat{C}(\mathcal{P}_J)$ and $\mathcal{V}(S(\mathcal{P}_J))$ are computed. For each item $j \in \bar{J}$ and for each corner point $c \in \hat{C}(\mathcal{P}_J)$, j is assigned at c and OneBin is called recursively. Whenever $\mathcal{V}(B) - \mathcal{V}(S(\mathcal{P}_J)) \leqslant \mathcal{V}(\bar{J})$ happens, backtracking occurs, because the remaining items cannot be packed. Empty space inside the envelope is not used by this algorithm. Our algorithm is an adaptation of the OneBin source code provided by Martello et al. [68].

To deal with the unloading constraint, we modified the algorithm as follows: when an item is assigned to a corner point, the algorithm verifies if the item blocks any item that will be removed before it. In this case, the algorithm backtracks. We also modified the initial ordering of items, ordering first by the customer's visiting order and then by nondecreasing volume.

CP model

The second exact algorithm was presented by Clautiax et al. [17], based on the constraint programming paradigm, to which we refer from now on as CP2D. In the CP2D algorithm, variables X_i and Y_i are associated to the bottom left corner of each item i. The domain of each variable is defined by $X_i \in \{0, \ldots, W - w_i\}$ and $Y_i \in \{0, \ldots, H - h_i\}$. For each pair (i, j) of items the following constraints must be satisfied: $([X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_i + h_i \leqslant Y_j]$ or $[Y_j + h_j \leqslant Y_i])$. These constraints guarantee that two items do not overlap. In order to consider the unloading constraints, we adapted the algorithm as follow: if i will be unloaded before item j, we replace the previous constraints by $([X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_j + h_j \leqslant Y_i])$. With the aim of improving this constraint programming model, Clautiax et al. [17] proposed a redundant constraint programming model for a non-preemptive cumulative-scheduling problem associated with two relaxations of 2OPPUL, which are linked to the original problem. A set of activities $\{A_1^w, \ldots, A_{|I|}^w\}$, and a resource R_w are defined. Each activity A_i^w requires h_i of resource R_w, and has processing time w_i. The resource R_w has

maximum capacity H. All activities must be executed by time W, and at any given time all activities being executed must not exceed the resource's maximum capacity. Analogously, we define the set $\{A_1^h, \ldots, A_{|I|}^h\}$ and resource R_h with maximum capacity W. Each activity A_i^h requires w_i of resource R_h, and has processing time h_i and all activities must be executed by time H.

To link these scheduling problems with the packing problem, the start time of an activity A_i^w must be equal to X_i and the start time of an activity A_i^h must be equal to Y_i. Using this model Clautiax et al. [17] proposed a series of pruning and propagation methods.

Reducing the Domain of Variables

In the formulation above, for each variable X_i, Clautiax et al. [17] considered all integer values between 0 and $W - w_I$ and for each variable Y_i the integer values between 0 and $H - h_i$. This can be improved by reducing the domain of each variable, reducing the number of choices where each item can be placed. For each dimension, we must find the *discretization points*, a reduced set of coordinates where items can be placed without changing the feasibility of the instance. The set of discretization points in a certain dimension is defined by all possible combinations of item's sizes in that dimension. Following the notation used by Côté et al. [20], consider $I_i^{\geqslant}$ the set of items to be unloaded with, or after i. These are the only items that can be packed bellow i. Let P_i^H be a set of coordinates over the Y-axis that item i can assume, defined as

$$P_i^H = \left\{ y = \sum_{j \in I_i^{\geqslant} \setminus \{i\}} h_j \xi_j : 0 \leqslant y \leqslant H - h_i, \xi_j \in \{0,1\}, j \in I_i^{\geqslant} \setminus \{i\} \right\}. \tag{2.4}$$

Let P_i^W be a set of coordinates over the X-axis that item i can assume, defined as

$$P_i^W = \left\{ x = \sum_{j \in I \setminus \{i\}} w_j \xi_j : 0 \leqslant x \leqslant W - w_i, \xi_j \in \{0,1\}, j \in I \setminus \{i\} \right\}, \tag{2.5}$$

This way, we define for each item $i \in I$:

$$Dom(X_i) = P_i^W, \tag{2.6}$$
$$Dom(Y_i) = P_i^H. \tag{2.7}$$

Herz [45] shows that any feasible solution for a given instance of the problem, has a corresponding solution over the discretization points, this way we do not lose any solution by reducing the domain to the discretization points.

The top-bottom mixfill strategy

To take advantage of the unloading constraints we decided to apply the top-bottom mixfill stra-

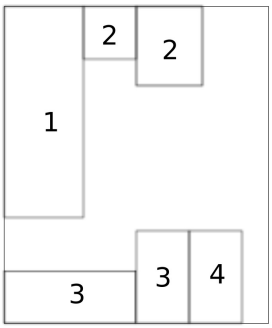


Figura 2.2: Example of packing using top-bottom mixfill.

tegy presented by Côté et al. [20] in the formulation of Clautiax et al. [17]. The unloading order can be explored to fill the bin not only from the bottom to the top, but from the top to the bottom too. This can be done by dividing the items into those who will be unloaded first, and the others, to be unloaded later. Let c be the cut point between these two sets, that is, items in $I^T = I_1 \cup \ldots \cup I_c$ will be packed from the top to the bottom, and items in $I^B = I_{c+1} \cup \ldots \cup I_n$ will be packed from the bottom to the top. This way, we can redefine the coordinates in the Y-axis where an item can be placed. To this purpose consider $I_i^{T,\leqslant}$ the set of items from I^T that will be unloaded with or before i, and $I_i^{B,\geqslant}$ the set of items from I^B that will be unloaded with or after i. Let P_i^H be the new set of coordinates y where item i can be packed, if $i \in I^T$:

$$P_i^H = \left\{ y' = H - y : y = \sum_{j \in I_i^{T,\leqslant} \setminus \{i\}} h_j \xi_j, 0 \leqslant y \leqslant H - h_i, \xi_j \in \{0,1\}, j \in I_i^{T,\leqslant} \setminus \{i\} \right\}. \quad (2.8)$$

If $i \in I^B$:

$$P_i^H = \left\{ y = \sum_{j \in I_i^{B,\geqslant} \setminus \{i\}} h_j \xi_j : 0 \leqslant y \leqslant H - h_i, \xi_j \in \{0,1\}, j \in I_i^{B,\geqslant} \setminus \{i\} \right\}. \quad (2.9)$$

Figure 2.2 presents an example of packing using the top-bottom mixfill strategy. Côté et al. [20] proved that if a solution is feasible for a given cut point c, it is feasible for every cut point. This way we can choose the cut point that generates the least number of discretization points.

2.2.3 Heuristic and Hash

Besides the exact approach, a fast heuristic was employed to help decreasing the time spent solving the 2OPPUL. The exact algorithms are not called if a feasible packing can be found by some of our heuristics. One is based on the Bottom Left Decreasing Width heuristic (BLDW) [3], modified to consider the unloading constraints. The other ones are based on the BRKGA and will be described later.

To reduce the computational effort, known routes are stored in a hash table. A route is stored whether it is feasible or not, therefore the same route will not have its feasibility checked twice. Two routes will have the same hash, excluding collisions, if they serve the same customers and have an identical visiting order.

2.2.4 Metaheuristic for the Two-Dimensional Orthogonal Packing Problem With Unloading Constraints

With the aim to reduce the total processing time of our exact algorithm, we also present some heuristics to the 2OPPUL based on the Biased Random-Key Genetic Algorithm (BRKGA). We first give an overview of the BRKGA, and then we show details about our proposed heuristics for the 2OPPUL problem under this approach.

The BRKGA

The BRKGA presented by [42] is a general search metaheuristic for finding solutions to combinatorial optimization problems. This algorithm uses a chromosome of fixed size m of random keys over the interval $[0, 1)$, where the value of m depends on the instance of the optimization problem. An evolutionary process involves crossing-over different chromosomes and exchanges among different populations. The BRKGA also introduces new chromosomes called mutants to add variety.

The BRKGA involves the following main parameters:

- m is the size of a chromosome;
- p is the number of individuals (chromosomes) in a population;
- p_e is the percentage of elite individuals in a population;
- p_m is the percentage of new mutants to be introduced in a population at each generation;
- ρ_e is the probability that a gene is inherited from the elite parent;
- K is the number of independent populations;
- $MaxGen$ is the number of generations evolved;
- $Exch$ is the number of generations evolved before Exchange best individuals among populations;
- $NExch$ is the number of best individuals to be exchanged among populations.

The BRKGA initializes each population with p randomly generated chromosomes, each having m random keys. Then it evolves each population by $MaxGen$ generations. The evolution of a population is composed of the following steps:

1. Compute the fitness function for each chromosome. A chromosome when associated with its fitness is called an individual.

2. Producing the next generation includes:

   (a) The elite set of the previous generation with $p \cdot p_e$ individuals,

   (b) $p \cdot p_m$ new randomly generated mutants.

   (c) Chromosomes produced by matching two individuals from previous generations, one from the elite set and another from the non-elite set. Each gene has a probability ρ_e to be copied from the elite parent.

To use this metaheuristic as a framework to an optimization problem we need the following steps: define the number of genes in a chromosome, define a decoder which maps a chromosome into a solution (feasible or not) to our specific problem, and define the fitness value of the chromosome, which will measure the quality of the solution. In the subsequent subsections we describe these steps for some developed heuristics.

Heuristic Bottom-Left

Chromosome Each item $i \in I$ has an associated gene g_i.

Decoder In this heuristic items are placed in the inverse sequence which they will be unloaded, that is, set I_u is packed before set I_v if $u > v$. Within each set I_v items are sorted according to the corresponding genes' values in the chromosome, that is, if $i \in I_v$ and $j \in I_v$, i is placed before j if $g_i < g_j$.

Let J be the set of all packed items, initially J is empty, and let be $\mathcal{P}_J$ the packing of those items in the Bin. Consider the set of corner points $\hat{C}(\mathcal{P}_J)$ as defined in section 2.2.1. We call an eligible *bottom-left corner point*, the point $p = (x_p, y_p) \in \hat{C}(\mathcal{P}_J)$, such that, y_p is minimum and $x_p + w_i \leqslant W$. Each item i in the order obtained from the chromosome is placed in the eligible *bottom-left corner point*. The order which items are placed guarantee that the unloading constraints are respected. We allow items to overflow the height H of the bin.

Fitness Value The fitness value of an individual is given by the height used by the packing in the heuristic. If at any time we have an individual with fitness less than or equal to H we have a feasible packing.

Heuristic Bottom-Left and Left-bottom

Chromosome Each item $i \in I$ has two associated genes g_i^1 and g_i^2.

Decoder In this heuristic items are placed in the order given by the genes, item i is placed before j if $g_i^1 < g_j^1$. Based on the heuristic proposed by Gonçalves et al. [43], items can be placed in two different positions. Let J be the set of all packed items, initially J is empty, and let $\mathcal{P}_J$ be the packing of those items in the Bin. An item i is placed in the eligible bottom left corner point $p = (x_p, y_p)$ or in the eligible *left-bottom corner point* denoted by $q = (x_q, y_q) \in \hat{C}(\mathcal{P}_J)$, such that x_q is minimum. Item i is placed in point p or q according to the gene g_i^2, if $g_i^2 < 0.5$ it is placed in p, otherwise, it is placed in point q. We allow items to overflow the height H of the bin.

Fitness Value The fitness value of an individual is given by the height used in the packing given in the heuristic, plus a penalty of value H for each unloading constraint violated, that is, for each pair of items, if the unloading constraint is violated one penalty is applied. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

Heuristic Tetris

Chromosome Each item $i \in I$ has two associated genes g_i^1 and g_i^2.

Decoder In this heuristic items are placed in the inverse sequence which they will be unloaded, that is, set I_u is packed before set I_v if $u > v$. Within each set I_v items are sorted according to the genes g^1, that is, if $i \in I_v$ and $j \in I_v$, i is placed before j if $g_i^1 < g_j^1$.

The gene g_i^2 represents the position x where the item i will be packed, and y is the lowest possible value, respecting the overlapping constraints. That is, item i will have position x_i defined by:

$$x_i = \lfloor g_i^2(W - w_i + 1) \rfloor. \tag{2.10}$$

The order in which items are placed guarantee that the unloading constraints are respected.

Fitness Value The fitness value of an individual is given by the height used in the packing given in the heuristic. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

Heuristic Double-Layer Tetris

This heuristic has two stages. In the outer stage the order that each item will be placed is decided using the BRKGA, and then for a given sequence, in the inner stage the x positions are decided also using a BRKGA.

Chromosome In the outer stage each chromosome has size equal to n. Each item $i \in I$ has an associated gene g_i^1. We will call this a sequence chromosome.

Decoder Items are placed in the inverse sequence which they will be unloaded, that is, set I_u is packed before set I_v if $u > v$. Within each set I_v items are sorted according to the chromosome, that is, if $i \in I_v$ and $j \in I_v$, item i is placed before item j if $g_i^1 < g_j^1$.

For each sequence chromosome we search for the packing positions by executing another BRKGA. In this inner stage we call a chromosome by position chromosome, where each item has an associated gene g_i^2 representing the position x where the item i will be packed, that is, item i will have position x_i defined by:

$$x_i = \lfloor g_i^2(W - w_i + 1) \rfloor, \tag{2.11}$$

and y is the lowest possible value, respecting the overlapping constraints. The fitness value in this step is given by the height used by the packing obtained. The order which items are placed guarantee that the unloading constraints are respected.

Fitness Value The fitness value is the height used by the packing given by the best position chromosome found in the inner stage. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

Heuristic Bottom-Left Penalty

Chromosome Each item $i \in I$ has an associated gene g_i.

Decoder The packing order of item i is given exclusively by the gene g_i, that is, if $g_i < g_j$, i will be packed before j independently of the unloading order. Items are packed in this order using the bottom left heuristic. This may lead to an unfeasible solution, so for each violation a penalty of value H is added to the fitness value.

Fitness Value The fitness value of an individual is given by the height used in the packing given by the heuristic, plus a penalty for each violated unloading constraint, that is, for each pair of items, if the unloading constraint is violated one penalty is applied. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

Heuristic Absolute Position

Chromosome Each item $i \in I$ has two associated genes g_i^x and g_i^y.

Decoder In this heuristic an item i is placed, if possible, on the position (x_i, y_i) given by:

$$x_i = \lfloor g_i^x(W - w_i + 1) \rfloor, \tag{2.12}$$

$$y_i = \lfloor g_i^y(H - h_i + 1) \rfloor. \tag{2.13}$$

Items are packed on the reverse order which they will be unloaded. Items from a same set I_v are packed in the order given in the input. If item i violates the overlap constraint or the unloading constraint we do not pack the item i. At the end there might be items not packed. Let I be the set of all items, J be the set of packed items, and c the total number of conflicts.

Fitness Value The fitness value is given by

$$|I| - |J| + (c/n^2). \tag{2.14}$$

If an individual has fitness value 0, then all items are packed and the solution is feasible. The last term of the fitness value helps to compare two individuals with the same number of packed items. If one solution has less conflicts, then it is closer to a feasible solution.

Heuristic Best Corner Point

Chromosome Each item i has an associated gene g_i.

Decoder Each gene represents the order of the packing, that is, if $g_i < g_j$, i will be packed before j. For each item i the heuristic packs it in the corner point that generates the minimum amount of waste, and do not violate the unloading constraints. For each corner point s, it verifies if i can be placed on s without exceeding the limits of the Bin, and without violating the unloading constraints with the items already packed. If no violation occurs, s is a candidate point to i. For each candidate s, the heuristic computes the volume of the complement of the envelope with item i packed on s, and chooses the one with the biggest volume. An item may have no candidates, in this case it is not packed.

Fitness Value Let I be the set of all items, J be the set of packed items, and S the total volume of the envelope. The fitness value is given by

$$|I| - |J| + (W\,H - S)/(2WH). \tag{2.15}$$

If an individual has fitness value less than 1, them all items were packed and the solution is feasible. The last term of the fitness value helps to compare two individuals with the same number of packed items, preferring those who let more remaining space to be used.

2.2.5 Lower Bounds for the Orthogonal Packing Problem

Some lower bounds presented by Côté et al. [20], are used to obtain a minimum dimension size to pack a set of items in a Bin. If the lower bound is greater than the dimension size of the bin, then the packing is unfeasible.

Lower Bound L_1

The first Lower Bound, L_1, considers the minimum height required do pack all items.

$$L_1 = \left\lceil \frac{\sum_{i \in I} w_i h_i}{W} \right\rceil. \tag{2.16}$$

Lower Bound L_2

To present the second lower bound, L_2, we need the following definitions:

- $I_i^{>,W-w_i}$ is the set of all items to be delivered after i, with width greater than $W - w_i$, and $\sum_{(i)}^{>,W-w_i}$ is the total volume of these items.
- $\sum_{(i)}^{>}$ is the total volume of items to be delivered after i.
- $I_i^{\leqslant,w_i}$ is the set of items to be delivered before or with i and with width greater than w_i.
- $I_i^{<,W-w_i}$ is the set of all items to be delivered before i, with width greater than $W - w_i$, and $\sum_{(i)}^{<,W-w_i}$ is the total volume of these items.
- $\sum_{(i)}^{<}$ is the sum of volumes of items to be delivered before i.
- $I_i^{\geqslant,w_i}$ is the set of items to be delivered after or with i and has width greater than w_i.
- P_i^H is the set of discretization points where item i can be placed.

The bound L_2 also uses the values of $y_i^{\min}$ and $y_i^{\max}$, which are respectively the minimal position that can be occupied by the bottom of item i and the maximal position that can be occupied by the top of item i. To calculate $y_i^{\min}$ and $y_i^{\max}$ the following values must be computed:

$$y_i^{\min,1} = \max\left\{ \left\lceil \frac{\sum_{(i)}^{>,W-w_i}}{W} \right\rceil, \max\{h_j : j \in I_i^{>,W-w_i}\} \right\}, \tag{2.17}$$

$$y_i^{\min,2} = \left\lceil \frac{[\sum_{(i)}^{>} - H(W - w_i) + \sum_{j \in I_i^{\leqslant,w_i}} h_j(w_j - w_i)]^+}{w_i} \right\rceil, \tag{2.18}$$

$$y_i^{\max,1} = H - \max\left\{ \left\lceil \frac{\sum_{(i)}^{<,W-w_i}}{W} \right\rceil, \max\{h_j : j \in I_i^{<,W-w_i}\} \right\}, \tag{2.19}$$

$$y_i^{\max,2} = H - \left\lceil \frac{[\sum_{(i)}^{<} - H(W - w_i) + \sum_{j \in I_i^{\geqslant,w_i}} h_j(w_j - w_i)]^+}{w_i} \right\rceil. \tag{2.20}$$

$$\tag{2.21}$$

This way $y_i^{\min}$ and $y_i^{\max}$ are computed as follow:

$$y_i^{\min} = \max\{y_i^{\min,1}, y_i^{\min,2}, \max_{j \in I_i^{>,W-w_i}} \{y_j^{\min} + h_j\}, \min\{t | t \in P_i^H\}\}, \tag{2.22}$$

$$y_i^{\max} = \min\{y_i^{\max,1}, y_i^{\max,2}, \min_{j \in I_i^{<,W-w_i}} \{y_j^{\max} - h_j\}, \max\{t | t \in P_i^H\} + h_i\}. \tag{2.23}$$

Finally using definitions of $y_i^{\min}$ and $y_i^{\max}$, L_2 can be computed as follow:

$$L_2 = \max_{i \in I}\{y_i^{\min} + h_i + (H - y_i^{\max})\}. \tag{2.24}$$

Lower Bound L_3

Based on the Cutting Stock Problem we can obtain the lower bound L_3^H. A cutting pattern is defined as a subset $I' \subset I$, such that $\sum_{i \in I'} w_i \leqslant W$. Let $\mathcal{K}^W$ be the set of all feasible patterns, and $a_{ik} = 1$ if item i belongs to pattern k, and $a_{ik} = 0$ otherwise. The following Integer Linear Program is constructed, where v_k is the variable that represents the number of times pattern k appears in the solution.

$$\min \sum_{k \in \mathcal{K}^W} v_k \tag{2.25}$$

$$\sum_{k \in \mathcal{K}^W} a_{ik} v_k \geqslant h_i \qquad i \in I \tag{2.26}$$

$$v_k \geqslant 0, v_k \in \mathbb{Z} \quad k \in \mathcal{K}^W. \tag{2.27}$$

The Cutting Stock Problem is usually solved by column generation, solving a series of knapsack problems with reduced costs. The knapsack problem is efficiently solved by Dynamic Programming. The optimal solution of the Cutting Stock Problem (CSP) gives a lower bound on the minimum height of the Bin necessary to pack all items. Let L_3^H be the value of the optimal solution of CSP, if $L_3^H < H$ we know that the packing of items I into bin $B = (W, H)$ is unfeasible.

Analogously, the bound L_3^W can be defined by doing the same process over the transversal axis. The resulting Cutting Stock Problem will give a lower bound on the width necessary to pack the items.

2.2.6 Three-dimensional Orthogonal Packing Problem With Unloading Constraints

To solve the Three-dimensional Orthogonal Packing Problem With Unloading Constraints (3OP-PUL), we used two different exact algorithms. The first is the OneBin_General algorithm presented by Martello et al. [69]. The algorithm was slightly modified to consider the customer's items and their unloading order when generating a feasible packing. The second is a natural CP

model to the problem.

CP based algorithm - relative positions

The first algorithm used to solve the 3OPPUL is the OneBin_General algorithm, introduced by Martello et al. [69]. This algorithm is based on constraint programming, and uses the following idea, two items (i, j) do not overlap if item i is *left of*, *right of*, *under*, *above*, *behind*, or *in front of* item j. To represent this relation between items i and j, it is created a variable r_{ij} for each pair of items, this variable can assume one of the following values $\{\texttt{l}, \texttt{r}, \texttt{u}, \texttt{a}, \texttt{b}, \texttt{f}\}$.

The problem is solved recursively by the algorithm OneBin_General. At each recursive call, two items i, j are considered and one of the relative positions are assigned to r_{ij}. The new assignment must be checked for its feasibility with the previous assignments made.

In each call, after assignments were made in the variables r, the algorithm needs to find the positions (x, y) for the placement of each item. For this purpose it initializes the position (x_i, y_i) of each item i with $x_i = 0$ and $y_i = 0$. For each relation already assigned, one of the following assignments is performed: if $r_{ij} = \texttt{l}$ and $x_j < x_i + w_i$, the item j is "pushed"left by doing $x_j = x_i + w_i$; otherwise if $r_{ij} = \texttt{r}$ and $x_i < x_j + w_j$, item i is "pushed"left by doing $x_i = x_j + w_j$. A similar assignment is made for the other possible relations. The algorithm repeats this procedure until no modifications are made. If at any point an item is positioned such that it exceeds the size of the bin, the last assignment has been proved to be unfeasible.

We have adapted the original algorithm to consider the unloading constraints. For a certain variable r_{ij}, we remove from its domain the value $\texttt{f}$(*front*) if i must be unloaded after j, or remove $\texttt{b}$(*behind*) if i must be unloaded before j.

A natural CP model - absolute positions

We now describe a natural constraint programming formulation for the Three-dimensional Loading problem. Similarly to the CP2D, variables X_i, Y_i and Z_i are defined for each item $i \in B$, where (X_i, Y_i, Z_i) is the position where item i will be packed. The initial domains are defined as $X_i \in \{0, \ldots, W - w_i\}, Y_i \in \{0, \ldots, H - h_i\}$ and $Z_i \in \{0, \ldots, D - d_i\}$.

For each pair of items i, j, we include a constraint $[X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_i + h_i \leqslant Y_j]$ or $[Y_j + h_j \leqslant Y_i]$ or $[Z_i + d_i \leqslant Z_j]$ or $[Z_j + d_j \leqslant Z_i]$. To attend the unloading constraint, if i must be unloaded before j, we replace this constraint by $[X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_i + h_i \leqslant Y_j]$ or $[Y_j + h_j \leqslant Y_i]$ or $[Z_j + d_j \leqslant Z_i]$. We call this algorithm by CP3D.

2.3 Capacitated Vehicle Routing Problem with Unloading Constraints

In this section we formally describe the Capacitated Vehicle Routing Problem with Unloading Constraints (DL-CVRP). We first formally describe the problem, then we present the ILP for-

mulation considered.

2.3.1 Problem Description

The Capacitated Vehicle Routing Problem with Unloading Constraints (DL-CVRP) can be defined as follows: It is given a complete undirected graph $G = (V, E)$, such that, V is a set of $n + 1$ vertices, where vertex 0 corresponds to the depot, and vertices $1, \ldots, n$ correspond to the n customers, and E is the set of edges. For each edge $e \in E$ there is an associated cost $c_e \in \mathbb{Q}^+$. It is also given a set of K identical vehicles, each having a weight capacity of M and a container of dimensions $W^1, \ldots, W^D$ to carry the customer's items, where D is the dimension of the problem. Let $V_+ = V \setminus \{0\}$ be the set of customers. Each customer $v \in V_+$ has a demand of a set I_v of items with total weight m_v. Each item i in $I = \bigcup_{v=1}^{n} I_v$ has dimensions $(w_i^1, \ldots, w_i^D)$.

We need to find a route for each vehicle, where each route has a feasible Packing that respects the unloading constraints. A feasible route C is a cycle in G that contains the depot and satisfies the following conditions:

(i) The total weight of all items of customers in C must not exceed the vehicle load capacity. That is, if V_C is the set of customers in C then $\sum_{v \in V_C} m_v \leqslant M$.

(ii) There is a packing $\mathcal{P}_C$ for the items of the clients in C in the container of the vehicle, such that this packing respects the unloading constraints.

The DL-CVRP is to find a set of K feasible routes $\mathcal{C} = \{C_1, \ldots, C_K\}$, that minimizes the total routing cost, that is, minimize $c(\mathcal{C}) = \sum_{i=1}^{k} \sum_{e \in C_i} c_e$.

2.3.2 Formulation

In this section we present the integer linear programming formulation used to model and solve the DL-CVRP.

Let q_e be a variable that indicates the use of an edge e in the solution, that is, q_e is equal to one if a vehicle travels along the edge e, and zero otherwise. Given a subset of customers $S \subseteq V_+$, $m(S)$ is the total weight of all items of the customers in S, i.e., $m(S) = \sum_{i \in S} m_i$. We denote by $\mathcal{V}(B)$ the volume of the container and $\mathcal{V}(S)$ the total volume of the items of the customers in S, i.e, $\mathcal{V}(S) = \sum_{v \in S} \mathcal{V}(I_v)$. Let $\delta(S)$ be the set of edges in the cut of G defined by S, i.e. the edges with exactly one vertex in S. Let $r(S)$ be the minimum number of vehicles needed to supply the demand of S considering the weight of the demands of the items. Also let $\mathcal{R}$ be the set of routes that are infeasible to be packed respecting the unloading constrains. The integer programming formulation is:

$$\text{minimize} \sum_{e \in E} c_e q_e \tag{2.28}$$

$$\text{s.a.} \sum_{e \in \delta(\{i\})} q_e = 2 \qquad \forall i \in V_+ \tag{2.29}$$

$$\sum_{e \in \delta(S)} q_e \geqslant 2r(S) \qquad \forall S \subseteq V_+, |S| \geqslant 2 \tag{2.30}$$

$$\sum_{e \in \delta(\{0\})} q_e = 2K \tag{2.31}$$

$$\sum_{e \in R} q_e \leqslant |R| - 1 \qquad \forall R \in \mathcal{R} \tag{2.32}$$

$$q_e \in \{0, 1\} \qquad \forall e \in E. \tag{2.33}$$

Constraints (2.29) ensure that each customer is visited exactly once. Constraints (2.30), the Capacity Inequalities, impose connectivity and capacity conditions. If $r(S)$ is replaced by $k(S) = max\{\lceil m(S)/M \rceil, \lceil \mathcal{V}(S)/\mathcal{V}(B) \rceil\}$, a valid lower bound is achieved. These are known as Rounded Capacity Inequalities (see [76]).

Constraints (2.31) ensure that exactly K vehicles are used. Constraints (2.32) ensures that there exists a packing respecting unloading constraints for each route. Note that to compute if a route R belongs do $\mathcal{R}$ is an NP-Hard problem, since it is equivalent to finding the optimal solution of the Orthogonal Packing Problem with Unloading Constraints (OPPUL), given the vehicle container and the customers items. Constraints (2.33) impose that the variables must be binary. Following [58], we do not allow routes with only one customer.

2.4 Branch-and-Cut Algorithm for the DL-CVRP

The DL-CVRP consists of a routing and a packing problem, therefore routing and packing strategies are needed. Following Azevedo et al. [2] our routing separation routine consists of separating Capacity Inequalities to ensure connectivity and capacity constraints, and using other families of inequalities trying to reduce the space to be explored. The packing constraints are evaluated when no more routing cuts can be found at the current node.

2.4.1 Routing Separation Routine

The formulation has an exponential number of constraints. The algorithm starts with a small set of constraints and add others by separation routines. Constraints (2.29) and (2.31) gives rise to a relaxed model that can be easily solved, generating a weak lower bound that can be strengthened by adding cutting planes. For this reason, the following separation routines were used: Capacity Inequalities, Framed Capacity Inequalities, Multistar Inequalities, 2-Edges Extended Hypotour Inequalities and Strengthened Comb Inequalities. Observe that separation

for Capacity Inequalities take into account not only the capacity of the vehicle, but also its container's area. Details on the used cuts can be found in [66] and [76].

Separating Capacity Inequalities is NP-Hard (see [76]), consequently separation is done on Rounded Capacity Inequalities, obtained by replacing $r(S)$ by $k(S)$ in constraints (2.30). To separate different CVRP families of inequalities we used the source code provided by [66].

Following Azevedo et al. [2], to allow better chances of finding more cutting planes at each iteration at the root node, two cyclical approaches based on a separation strategy proposed by [66] are used. At each iteration, the following separation routines can be called: Rounded Capacity Inequalities, Framed Capacity Inequalities, Multistar Inequalities, Strenghtened Comb Inequalities and 2-Edges Extended Hypotour Inequalities. For each separation attempt, every inequality that was found is added to the LP.

Initially, the algorithm tries to separate Rounded Capacity Inequalities. If there is at least one cut violated by less than a certain limit, the algorithm attempts the Framed Capacity separation. If no Framed Capacity Inequality is found, two different approaches are used: if the current iteration is a multiple of five, the algorithm tries to separate Multistar and Strenghtened Comb Inequalities. Otherwise, a circular strategy is put into action: separation of Multistar, Strenghtened Comb and Hypotour occurs every third iteration, e.g., if Multistar Inequalities are found, re-optimization occurs and the algorithm will not try to separate this family again until both Strenghtened Comb and Hypotour Inequalities are found. The order used was Multistar, followed by Strenghtened Comb and then Hypotour Inequalities. It should be noted that separation is pursued only if the algorithm finds at least one Rounded Capacity Inequality violated by a predefined limit. This limit is different for each family of inequalities.

Non-root nodes of the branching tree are treated differently from the root node. First the algorithm calls the routine to separate Capacity Inequalities, then we try to separate Framed Capacity Inequalities, followed by Multistar Inequalities and Strenghtened Comb Inequalities. Unlike our approach for the root node, these procedures are called without any conditions (limits). For each subsequent iteration, only Capacity separation is attempted.

2.4.2 Packing Separation Routine

When none of the Routing Separations constraints are found, we can use the information given by the packing problem to add some extra cuts. Those cuts will also eliminate unfeasible routes, the ones with clients such that their items cannot be packed in a single vehicle. Given a partial solution for the integer linear program (2.28-2.33) described in section 2.3.2, it is possible that not all variables are integer. We propose two methods that can be used to add extra cuts.

In the first method, the Branch-and-Bound is executed until all values are integers, so the solution to the ILP model is a set of routes. For each of these routes, the algorithm searches for a feasible packing. Figure 2.3 illustrates a partial solution where all variables are integers. We will call this method *BranchFirst*.

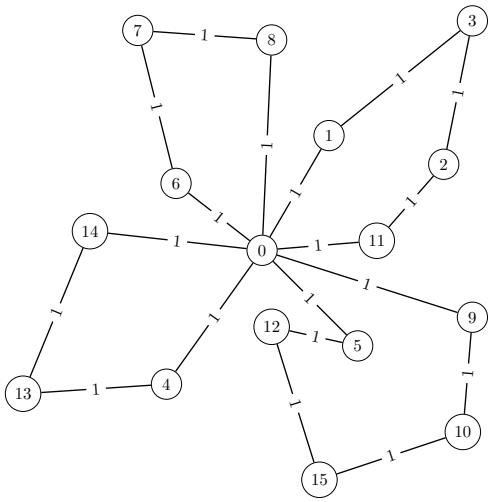


Figura 2.3: Example of a partial solution, packing can be tested for each route.

The second strategy searches for possible cuts even with fractional solutions to the ILP model, as illustrated in Figure 2.4. We define the problem of finding cuts in this partial solution as *Unpackable Path Problem*.

Unpackable Path Problem

The Unpackable Path Problem is the following: given a fractional partial solution to the ILP model, find a path whose customer's items cannot be packed in that specific order. This path can be removed from the solution, by adding a new cut.

Heuristic for Unpackable Path Problem

Initially all edges e whose value of x_e is below a certain value $p > 0.5$ are removed. In the example of figure 2.4 we considered $p = 0.7$, and removed the dashed edges. The edges incident to the depot are also removed. We obtain a set of disjoint paths and isolated vertices, these paths are called sub-routes. Let P be one of these paths, where $\sum_{e \in P} x_e > m - 2$. The algorithm then check if the packing of items associated to the path P is feasible. If it is not, we add the following cut $\sum_{e \in P} x_e \leqslant |P| - 1$, pruning this sub-route from the set of solutions. We will call this strategy *CutFirst*.

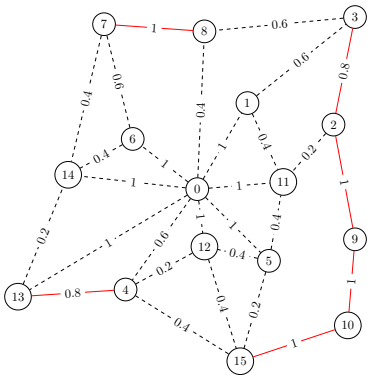


Figura 2.4: Example of a partial solution with fractional variables, packing can also be used to find cuts in this case.

For each route or sub-route of a solution or partial solution, we perform the following steps: first check if it can be found in the hash table. If so, it means that this route's packing was solved previously and its feasibility is known. Otherwise, we try our packing heuristic. The heuristic attempts to pack the items of customers considering the sequence of visit determined by the route. If a feasible packing is found, we update the hash table. If it fails, one of the exact algorithms is called and the hash table is updated with the results found. Thus we avoid multiple calls to identical packing instances. If there is no feasible packing for the corresponding route, a cut is added to the model in order to remove this route from the feasible space.

One can also conclude the unfeasibility of routes or sub-routes by looking for unpackable subsequences. For instance, consider we wish to know if the route (3, 2, 9, 10, 15) is feasible or not, and we already have concluded and stored that (3, 9, 10) is unfeasible, we can conclude that the route (3, 2, 9, 10, 15) is also unfeasible. Obviously, the search for this subsequence may take some time and do not always compensate. Therefore we tested the algorithm with and without this search procedure. Combined with the strategy *CutFirst*, we call this search by *SubSeqSearch*.

2.5 Computational Results

The proposed algorithms were implemented in c++ language, and compiled with g++ 4.6.4 in a computer running Linux. The experiments were run on a 2.93GHz Intel Xeon Cpu. With the exception of the BRKGA metaheuristic, all algorithms runs in a single core. The integer linear programming solver used was the Cplex 12.5.1, and the constraint programming solver was the CP Solver 1.7, both from IBM ILOG.

Following Lysgaard et al.[66] and Azevedo et al. [2] the root node separation strategy tries to separate different families of cuts only if there is at least one Rounded Capacity Inequality violated by less than a certain limit. These limits are: 0.2 for Framed Capacity Inequalities, 0.05 for Multistar Inequalities, 0.1 for Strengthened Comb Inequalities and 0.1 for the 2-Edges Extended Hypotour Inequalities.

2.5.1 Instances

We tested our exact algorithm for the 2L-CVRP on the 60 instances used by Iori et al. [58] which were based on 12 instances for the CVRP provided by Reinelt [81]. For each one of these 12 CVRP instances, Iori et al. [58] considered the complete graph connecting the clients, the weights m_v for each client v and the total capacity M of each vehicle. The distances between each pair of clients is an integer obtained by truncating the corresponding euclidian distances. For each one of these 12 CVRP instances, 5 classes of instances were created by Iori et al. [58], differing on how items for each client were generated. In the first class, each original instance of the CVRP is adapted to the 2L-CVRP in a straightforward way: create a single item of sides length equal to 1 for each client, and a container with $W = H = n$. For the remaining classes it was considered vehicles with $W = 20$ and $H = 40$. The number of items per client is an uniform random value in the interval $[1, r]$, where $r \in \{2, \ldots, 5\}$ is the number of the class. For each item its shape is selected with equal probability among: *Vertical*, *Homogeneous* and *Horizontal*, and its side lengths h and w are randomly generated from the intervals given in Table 2.1. The number of available vehicles in each instance were obtained by Iori et al. [58], heuristically solving a Two-dimensional Bin Packing Problem considering all items, but not considering Unloading Constraints or gathering items of a same customer in the same container. Then K is as the maximum value between this number and the number of vehicles in the original CVRP instance. These instances are available at [26].

Tabela 2.1: Classes for the 2L-CVRP instances

		Vertical		*Homogeneous*		*Horizontal*	
Class	Itens per Client	h	w	h	w	h	w
1	1	1	1	1	1	1	1
2	$[1, 2]$	$\left[\frac{4H}{10}, \frac{9H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{2H}{10}, \frac{5H}{10}\right]$	$\left[\frac{2W}{10}, \frac{5W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{4W}{10}, \frac{9W}{10}\right]$
3	$[1, 3]$	$\left[\frac{3H}{10}, \frac{8H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{2H}{10}, \frac{4H}{10}\right]$	$\left[\frac{2W}{10}, \frac{4W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{3W}{10}, \frac{8W}{10}\right]$
4	$[1, 4]$	$\left[\frac{2H}{10}, \frac{7H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{H}{10}, \frac{4H}{10}\right]$	$\left[\frac{W}{10}, \frac{4W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{2W}{10}, \frac{7W}{10}\right]$
5	$[1, 5]$	$\left[\frac{H}{10}, \frac{6H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{H}{10}, \frac{3H}{10}\right]$	$\left[\frac{W}{10}, \frac{3W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{W}{10}, \frac{6W}{10}\right]$

To test the efficiency of the modifications done in the constraint programming algorithm CP2D, we used the instances provided by Côté et al. [20]. These instances were obtained by first creating extra instances based on 180 2L-CVRP instances reported in [41], by modifying the dimensions of the container according to 5 different types:

- Type 1 : H = 40, W = 20
- Type 2 : H = 32, W = 25
- Type 3 : H = 50, W = 16

- Type 4 : H = 80, W = 14
- Type 5 : H = 130, W = 14

This resulted in 900 2L-CVRP instances, in which they heuristically generate routes of clients saving those that was not proved unfeasible by some Lower Bounds. They end up with a total of 2183 2OPPUL instances. Which we use to test the efficiency of our packing algorithms.

The heuristics presented in section 2.2.4 aim to solve hard instances for which the exact algorithm would take too much time. The instances provided by Côté et al. [20] were easily solved by the exact methods, so we created a new set of instances for the two-dimensional orthogonal packing problem. These instances were generated by running our exact algorithm for the 2L-CVRP, and when the algorithm faces a 2OPPUL that cannot be solved by the CP2D within 1 second, we save this route as a new 2OPPUL instance. We end up with a total of 296 instances. These new instances are available at the our Laboratory website `http://www.loco.ic.unicamp.br`.

We also use the 3L-CVRP instances presented by Gendreau et al. [40] to verify the behaviour of our algorithm in the three-dimensional case.

2.5.2 Efficiency of CP and Discretization Points

The efficiency of using the discretization points and the top-bottom mixfill approach were tested in the instances of [20]. We compared the efficiency of the original CP2D algorithm, its version using discretization points and top-bottom mixfill approach and also the adapted Branch-and-Bound algorithm OneBin that considers unloading constraints. We used a time limit of 600 seconds for the algorithms in the experiments.

We present the results in Table 2.2. Column S_{OB} presents the number of instances solved to optimality by the adapted OneBin algorithm, while column T_{OB} presentes the average time used to solve these instances. Columns S and T indicate respectively the number of instances solved to optimality, and the average time used by the original CP2D algorithm with complete domain. Analogously columns S_{dp} and T_{dp} indicate the number of instances solved to optimality, and the average time taken by CP2D algorithm using discretization points (see Section 2.2.2). Finally columns S_{dp_mix} and T_{dp_mix} indicates respectively the number of instances solved to optimality and the average time used by CP2D algorithm using both discretization points and the top-bottom mixfill approach (see Section 2.2.2). Those combinations of class and type where all the algorithms were not able to solve any instance were omitted.

The number of solved instances increased by 7.4% when using discretization points in the CP2D algorithm, when compared to its original version. And the number of solved instances increased by 8.7% when using both discretization points and the top-bottom mixfill approach. One important note is that the number of solved instances did not decrease in none of the instances types, when using these strategies.

Tabela 2.2: Performance of CP with dicretization points and mixfill

Class	Type	#inst	S_{OB}	T_{OB}	S	T	S_{dp}	T_{dp}	S_{dp_mix}	T_{dp_mix}
1	3	30	27	80.37	29	9.12	30	10.39	30	8.56
1	4	198	64	185.66	185	33.81	190	28.58	190	25.85
1	5	200	3	384.01	68	153.37	79	147.92	81	151.66
2	2	1	1	0.21	1	0.06	1	0.04	1	0.00
2	3	128	122	62.47	128	18.43	128	9.88	128	8.42
2	4	198	89	195.56	187	50.90	194	47.43	194	43.37
2	5	200	5	405.06	55	162.25	61	152.17	63	159.46
3	3	2	2	32.35	2	0.36	2	0.22	2	0.19
3	4	155	30	194.35	144	30.89	148	22.45	148	22.22
3	5	206	2	345.03	59	147.54	75	124.07	75	116.87
4	4	187	2	218.11	68	152.84	85	118.86	92	114.50
4	5	182	0	0.00	1	136.231	3	332.42	4	363.55

2.5.3 Efficiency of metaheuristics

The proposed heuristics to solve the two-dimensional orthogonal packing problem were implemented using the brkgaAPI framework [89], that implements the BRKGA described in section 2.2.4. These heuristics are used to improve the time spent to find a feasible packing that must respects unloading constraints of some given route. Unfortunately, the set of instances provided by Côté et al. [20] does not reflect the routes that we desire to solve in the 2L-CVRP instances considered.

To evaluate the performance of these heuristics, we generated a new set of instances as follows: for each 2L-CVRP instance proposed by Iori et al. [58], we ran the branch-and-cut algorithm for 10 seconds, and for each found route, it is first checked its feasibility with the lower-bounds. If the lower bounds did not prove its unfeasibility, we ran the basic BLDW algorithm presented in section 2.2.3, and if it does not found a packing, we ran the CP2D algorithm with a time limit of 1 second. If feasibility or unfeasibility has not been proved by any of these methods then we store this route in our set of instances. This procedure resulted in a set of 296 instances to the orthogonal packing problem.

We evaluated the performance of the BRKGA heuristics with this new set of instances. Table 2.3 presents the performance of the Heuristic Absolute Position presented in section 2.2.4, and Heuristic Best Corner Point presented in section 2.2.4. The column "Class"indicates the class of the instance, column "#inst"indicates the number of instances of that type and column "avg. it."the average number of items in these instances. Columns S_{cp} and T_{cp} represents respectively the number of feasible instances of that type, and the average time spent by the CP2D algorithm to solve these instances. Columns S_{abs} and T_{abs} represents respectively the number of feasible solutions found by Heuristic Absolute Position and the average time when the best solution was found. Analogously, columns S_{bcp} and T_{bcp} represents respectively the number of feasible solutions found by Heuristic Best Corner Point and the average time when the best solution was found.

Table 2.4 presents the performance of the remaining heuristics based on BRKGA (see Section 2.2.4). For each heuristic he three columns S_{he}, T_{he} and H_{he} indicate respectively, the number of feasible solutions found by heuristic he, the average time when the best solution was found, and the average best fitness function value. The heuristics are

- *bl*: Heuristic Bottom-Left presented in Section 2.2.4
- *bl-lb*: Heuristic Bottom-Left and Left-bottom presented in Section 2.2.4
- t: Heuristic Tetris presented in Section 2.2.4
- $d.t$: Heuristic Double-Layer Tetris presented in Section 2.2.4
- *bl-p*: Heuristic Bottom-Left Penalty presented in Section 2.2.4

Heuristics Better Corner Point and Double-Layer Tetris achieved the best results. For this reason we included only these two in the branch-and-cut algorithm, besides the basic BLDW heuristic.

2.5.4 Comparison of the 2L-CVRP algorithms

Since we did not have acess to the original implementation of Iori et al. [58], we compare our algorithm with a simpler version of the branch-and-cut, disabling heuristics presented in Section 2.4.2 and checking if a routing is feasible, using the hash table, the bottom-left heuristic and the OneBin algorithm limited to 600 second, this is similar to the algorithm proposed by Azevedo et al. [2]. We refer to this simpler version as BNC, while we refer to our improved version as BNC-improved. In BNC-improved we limit the time of CP2D in 2 seconds. Which means that the solution found might not be optimal. We ran our code in a single core of an Intel Xeon 2.93GHz with 8GB of ram.

Table **??** presents the results. First is presented information about the instances: columns Name, Class, number of clients (n), total number of items (M) and number of available vehicles (K). The results of the approach of BNC is presented next: the total time spent by the algorithm

Tabela 2.3: Performance of metaheuristics to the orthogonal packing problem

Class	#inst	avg. it.	S_{cp}	T_{cp}	S_{abs}	T_{abs}	S_{bcp}	T_{bcp}
3	22	13.05	1	2.32	0	0.13	0	0.02
4	67	14.94	9	31.11	0	0.21	2	0.03
5	207	19.84	112	62.19	0	0.39	36	0.09
All	296	18.23	122	50.7	0	0.33	38	0.07

Tabela 2.4: Performance of metaheuristics to the strip packing problem.

Type	S_{bl}	T_{bl}	H_{bl}	$S_{bl\text{-}lb}$	$T_{bl\text{-}lb}$	$H_{bl\text{-}lb}$	S_t	T_t	H_t	$S_{d.t}$	$T_{d.t}$	$H_{d.t}$	$S_{bl\text{-}p}$	$T_{bl\text{-}p}$	$H_{\text{bl-p}}$
3	0	0	53.82	0	0.06	47.05	0	0.16	44.36	0	0.17	43.68	0	0.03	50.18
4	0	0	51.84	1	0.09	47.07	0	0.15	45.03	1	0.22	44.19	0	0.04	50.49
5	4	0.02	49.46	3	0.21	46.79	2	0.27	44.51	17	0.47	43.9	2	0.1	49.67
All	4	0.01	50.32	4	0.17	46.88	2	0.23	44.61	18	0.39	43.95	2	0.08	49.9

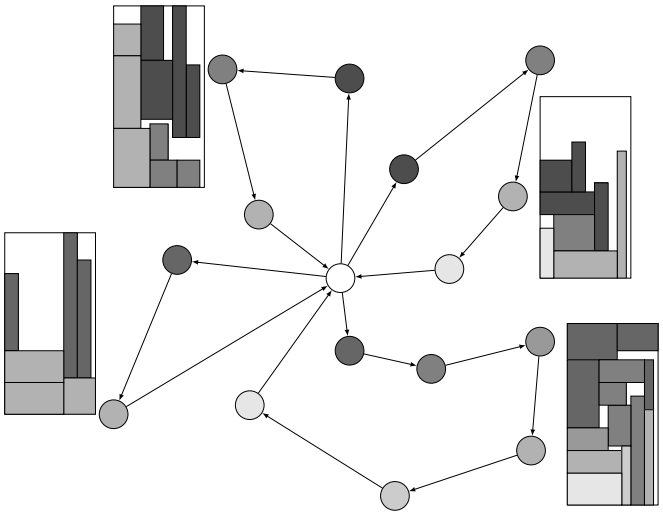


Figura 2.5: Example of solution for 2L-CVRP

($time$), and the solution value (z) of the best feasible solution found. Then in the last columns is presented the results of our algorithm BNC-improved: the total time spent by the algorithm ($time$), and the solution value (z) of the best feasible solution found. We set a time limite of 3600 seconds to both algorithms. Figure 2.5 represents a solution for instance *E016-03m* with $K = 4$.

The results show that, with our improvements, there was a reduction of the time spent in solving packing sub-problems, and then the BNC-improved was able to find feasible solutions to all instances in 3600 seconds, while the BNC could not give feasible solutions for 7 of those instances. The impact of the Rounded Capacity Inequalities and hash tables may be found in [2].

2.5.5 Comparison of the 3L-CVRP algorithms

We compared our algorithm with a simple version of a branch-and-cut algorithm. We ran our code in a single core of an Intel Xeon 2.93GHz with 8GB of ram. We set our time limit to 3600 seconds, and the time for each packing sub-problem was limited to 60 seconds. As we do not consider rotations and fragility of items, we can not compare our algorithm with the tabu search approach presented by Gendreau et al. [40].

Table 2.6 presents the results for the 3L-CVRP. It presents, for each instance, the time spent solving the instance and the value of the solution found. We compared a simple Branch-and-Cut algorithm (BNC), and for our improved Branch-and-Cut (BNC-Improved). The instances where both algorithms did not found an integer solution were removed.

To solve the 3opp problem we used the simple CP formulation presented in Section 2.2.6. We strongly believe that adapting the techniques presented to solve the 2opp to the three-dimensional case will improve the algorithm, increasing the number of solutions found. We only present results for those instance where our algorithm found an integer solution. We can see that the BNC-Improved was able to find solutions to two more instances when compared to

Instance					BNC		BNC-Improved	
Name	Class	n	M	K	Time	z	Time	z
E016-03m	1	15	15	3	1.14	273	1.19	273
	2	15	24	3	21.16	285	10.00	285
	3	15	31	3	85.22	280	49.79	280
	4	15	37	4	23.97	288	0.08	288
	5	15	45	4	600.76	279	0.79	279
E016-05m	1	15	15	5	0.30	329	0.31	329
	2	15	25	5	2.76	342	2.63	342
	3	15	31	5	7.18	347	3.81	347
	4	15	40	5	287.74	336	1.43	336
	5	15	48	5	0.54	329	0.25	329
E021-04m	1	20	20	4	0.46	351	0.49	351
	2	20	29	5	2275.52	396	10.95	396
	3	20	46	5	373.43	387	5.31	387
	4	20	44	5	99.07	374	1.41	374
	5	20	49	5	520.81	369	5.31	369
E021-06m	1	20	20	6	1.48	423	1.35	423
	2	20	32	6	4.15	434	1.23	434
	3	20	43	6	4.74	432	5.45	432
	4	20	50	6	57.29	438	17.18	438
	5	20	62	6	3011.87	423	17.46	423
E022-04g	1	21	21	4	0.05	367	0.05	367
	2	21	31	4	2.17	380	2.13	380
	3	21	37	4	46.73	373	12.93	373
	4	21	41	4	892.87	377	1.19	377
	5	21	57	5	605.29	389	2.32	389
E022-06m	1	21	21	6	0.95	488	1.05	488
	2	21	33	6	4.33	491	8.34	491
	3	21	40	6	35.46	496	13.12	496
	4	21	57	6	141.67	489	3.36	489
	5	21	56	6	4491.52	488	30.87	488
E023-03g	1	22	22	3	0.02	558	0.03	558
	2	22	32	5	3600.01	724	61.75	724
	3	22	41	5	98.88	698	19.15	698
	4	22	51	5	2120.64	714	56.84	714
	5	22	55	6	1035.19	742	0.93	742
E023-05s	1	22	22	5	0.02	657	0.02	657
	2	22	29	5	778.18	720	23.50	720
	3	22	42	5	3600.00	730	26.24	730
	4	22	48	5	1611.09	701	31.64	701
	5	22	52	6	601.37	721	0.07	721
E026-08m	1	25	25	8	0.62	609	0.62	609
	2	25	40	8	0.47	612	3.60	612
	3	25	61	8	1.62	615	20.52	615
	4	25	63	8	95.70	626	40.98	626
	5	25	91	8	3112.99	609	16.14	609
E030-03g	1	29	29	3	1.25	524	1.20	524
	2	29	43	6	3600.00	-	3600.00	687
	3	29	49	6	3600.01	637	1450.30	637
	4	29	72	7	3600.00	-	3600.03	739
	5	29	86	7	3600.00	-	1426.58	706
E033-03n	1	32	32	3	0.15	1909	0.36	1909
	2	32	44	7	3600.00	-	3600.00	2715
	3	32	56	7	3600.02	2854	3219.52	2574
	4	32	78	7	3600.00	-	3603.01	2699
	5	32	102	8	3600.90	2621	1246.30	2672
E036-11h	1	35	35	11	218.15	682	249.64	682
	2	35	56	11	364.63	682	161.58	682
	3	35	74	11	712.04	682	327.16	682
	4	35	93	11	3600.07	691	2481.22	691
	5	35	114	11	3600.00	-	1003.71	682

Tabela 2.5: Comparison between a simpler branch-and-cut algorithm and our BNC-Improved

the BNC. It also found solutions of equal or better quality, when compared to BNC.

Tabela 2.6: Comparison between our BNC-Improved and BNC.

Instance	BNC		BNC-Improved	
Name	$Time$	z	$Time$	z
E016-03m.dat	157.35	293	39.04	293
E016-05m.dat	0.28	329	1.17	329
E021-04m.dat	1007.61	357	227.64	357
E021-06m.dat	393.15	423	36.13	423
E022-04g.dat	127.55	424	26.05	424
E022-06m.dat	273.24	488	73.57	488
E023-03g.dat	599.01	761	132.82	761
E023-05s.dat	1891.84	822	458.47	822
E026-08m.dat	687.59	661	1228.21	661
E030-03g.dat	1450.77	797	3600.50	797
E030-04s.dat	268.35	770	686.90	770
E031-09h.dat	2406.83	611	3600.03	611
E033-04g.dat	3668.41	1583	3606.77	1564
E033-05s.dat	3663.25	1537	3609.41	1537
E036-11h.dat	1644.89	700	3134.11	700
E041-14h.dat	3600.60	-	3600.03	849
E051-05e.dat	3357.87	747	3607.65	747
E101-14s.dat	3627.47	-	3601.23	1541

2.6 Conclusions and Future work

In this paper we present a branch-and-cut algorithm for the vehicle routing problem with unloading constraints. The algorithm uses known separation routines for the vehicle routing problems and an exact packing algorithm to separate infeasible routes and subroutes. The packing algorithm is an adaptation of the two-dimensional packing algorithm presented by Clautiaux et al. [17], to satisfy unloading constraints. This algorithm was improved, by restricting the item packing positions to discretization points and applying the top-bottom mixfill approach. The improved algorithm showed to be more robust and faster than the compared algorithms. Compared to the straighforward adaptation of the algorithm presented by Clautiaux et al. [17], the presented algorithm obtained 8.7% more exact solutions for compared instances, and could solve 190% more instances than the algorithm presented by Azevedo et al. [2]. We also presented several heuristics for the two-dimensional packing problem with unloading constraints, as alternatives to the exact algorithm. For example, in a set of 38 instances where the exact algorithms take more than 1 second each, the Best Corner Point heuristic obtained solutions in 0.07 seconds, on average.

The main algorithm, for the vehicle routing problem with unloading constraints, obtained optimum solutions within significantly smaller computational times for most of the instances presented in the literature. We made a computational comparison of the presented algorithm with the one presented in [2], with computational time limited to 3600 seconds. For all instances solved to the optimallity by the compared algorithm, the presented algorithm also obtained optimum solutions and within 5.8% of the total time used by the compared algorithm. For the 13 instances not proven to be optimum by the compared algorithm, the presented algorithm obtained optimum solutions for 11.

Many practical problems are in fact problem combinations, that are better solved with different approaches. This suggest the development of hybrid algorithms that makes use of different algorithmic methods, each one to deal with more appropriate structures. In this paper we present an hybrid algorithm for a problem that combines routing and packing constraints. The algorithm combines different approaches, as constraint programming, integer linear programming, branch-and-bound and metaheuristics. The computational experiments showed the presented algorithm could obtain improved results, showing the success of the hybridization approach. It does not seems that simple partition of a complex problem into subproblems with different characteristic and the application of specific algorithms for each part will lead to good algorithms for the main problem. The difficult resides in how to divide the main problem in a way to guide combined subproblem solutions in the direction of better solutions for the main problem.

For future improvements of the algorithm, we mention the use of initial primal heuristics, more efficient packing strategies and improved lower bounds. Directions for future research may include the integration of the problem with other constraints, as time windows, packing stability and pickup and delivery constraints. The presented technique can also be applied to other routing problems where packing constraints can be included, such as the shortest paths, traveling salesman, multi-depot and cross-docking vehicle routing problems, among others. The packing algorithms can be easily adapted to solve other packing problems, such as the two- and three-dimensional knapsack and bin packing problems and their variations.

Acknowledgements: This research was partially supported by CAPES, CNPq (Proc. 477692/2012-5,311499/2014-7,306358/2014-0 and FAPESP (Proc. 2011/13382-3).

Capítulo 3

A Bounded Space Algorithm for Online Circle Packings *

ABSTRACT

We study the Online Circle Packing Problem where we need to pack circles that arrive online in square bins with the objective to minimize the number of bins used. An online algorithm is said to have bounded space if at any given time, only a constant number of bins are open, circles are packed only in open bins and once a bin is closed it cannot be reopened. In particular, we present a 2.4394-competitive bounded space algorithm for this problem and a 2.2920 lower bound on the competitive ratio of any online bounded space algorithm for this problem.

KEYWORDS. Circle packing, Online algorithms, Worst-case analysis, Computational geometry

3.1 Introduction

In the *Online Circle Packing Problem*, one has infinitely many square bins and receives a list of circles (given by its radii) in an online fashion. When a circle arrives, it must be packed in a bin, without intersecting other circles or the borders of the bin. Also, after packing the circle, it cannot be moved to another bin or another position in the bin. The objective is to minimize the number of bins used.

We say that an online algorithm A has an asymptotic competitive ratio of α if, for every instance I, $A(I) \leqslant \alpha\,\mathrm{OPT}(I) + C$ where $A(I)$ is the value of the solution produced by algorithm A, $\mathrm{OPT}(I)$ is the value of an optimal offline solution and C is a constant. In this paper we present an online algorithm with asymptotic competitive ratio at most 2.4394. This algorithm has the nice property that it has *bounded space*, that is, at any time there is at most a constant

*This work was partially supported by CNPq (grants 311499/2014-7, 477692/2012-5) and FAPESP (grant 2011/13382-3, 2013/21744-8).

number of open bins. After a bin is closed, it is not opened anymore and, hence, does not receive new circles. Also, we present a 2.2920 lower bound on the competitive ratio of any online bounded space algorithm for this problem.

Previous Works The book of Szabo et al. [88] presents many results regarding finding the maximum common radius of k circles that can be packed in a unit square for several values of k along with other related problems. The website maintained by Specht [86] collects even more results, not only regarding the packing of circles in a unit square but also the packing of circles in a circle, in an isosceles right triangle, in a semicircle, in a circular quadrant and other problems. Some applications of circle packing includes obtaining a maximal coverage of radio towers in a geographical region [88] and construct photo collages [100]. A review on circle packing problems and methodologies can be found in [48].

For the offline circle packing problem, there is an asymptotic polynomial time approximation scheme by Miyazawa et al. [73] when we can augment the bin in one direction which can also be adapted to the circle strip packing problem. Note that, as shown by Demaine et al. [25], it is NP-hard to decide if a set of circles can be packed into a square bin.

This online problem is already studied in the literature when the objects to be packed are squares, rectangles and hyperboxes. Epstein and Van Stee [31] developed a bounded space online algorithm for the d-dimensional online hypercube packing, extended this algorithm for the d-dimensional online hyperbox packing, for the variable-sized d-dimensional bin packing problem and for the online bin packing with resource augmentation. Later on, Epstein and Van Stee [32] presented numerical lower and upper bounds for d-dimensional online bounded space hypercube for $d \in \{2, \ldots, 7\}$. Epstein [30] presented bounded and unbounded space algorithms for the two-dimensional online rectangle packing with orthogonal rotations.

3.2 An Algorithm for Online Circle Packing

We start by presenting an algorithm for the Online Circle Packing Problem. For simplicity, we consider that a bin is a square of side length 1. The algorithm divides the circles into large circles and small circles. Given some positive integer constant M, a circle is said to be *large* if its radius is bigger than $1/M$ and, otherwise, it is said to be *small*.

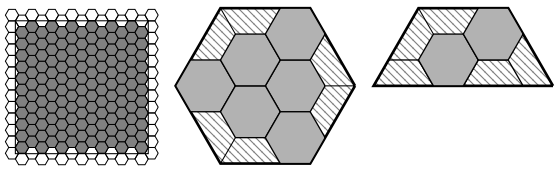
For every positive integer i, let ρ_i^* be the largest value such that i circles of radius ρ_i^* can be packed in a bin. For example, $\rho_1^* = 0.5$ since we can pack a circle of radius 0.5 in a bin but we cannot pack a circle of radius $0.5 + \varepsilon$ in a bin for any $\varepsilon > 0$. As only some few values of ρ_i^* are currently known, we will use the best known lower bound on the value of unknown ρ_i^*, by ρ_i, obtained from the literature [86]. For algorithms that can compute ρ_i, we refer to [88]. Let K be such that $\rho_K > 1/M \geqslant \rho_{K+1}$. We will say that a large circle is of type $1 \leqslant i < K$ if its radius is at most ρ_i and larger than ρ_{i+1} and of type K if its radius is at most ρ_K and larger than $1/M$. We pack large circles of the same type together, packing at most i circles of type i in the same bin.

Let $C > 1$ be a positive integer constant that is a multiple of 3. We say that a small circle

of radius r is of type i (for $M \leqslant i < CM$), subtype k (or, simply that r is of type (i,k)) if $1/(i+1) < C^k r \leqslant 1/i$ where k is the largest integer such that $C^k r \leqslant 1/M$. Small circles are packed using a recursive hexagonal packing defined later.

At a given time, the algorithm maintains at most K bins opened to pack large circles and $(C-1)M$ bins opened to pack small circles, and thus, it has bounded space. Recall that the area of a hexagon of side length ℓ is $3\sqrt{3}\,\ell^2/2$. Also, the radius of the inscribed circle of a hexagon of side length ℓ is $\sqrt{3}\,\ell/2$. That is, it is possible to pack a circle of radius r in a hexagon of side length $2r/\sqrt{3}$.

The algorithm generates three types of sub-bins. For $1 \leqslant i \leqslant K$, a *c-bin* of type i is a circular bin of radius ρ_i (it is used only for large circles). For $M \leqslant i < CM$ and $k \geqslant 0$, an *h-bin* of type (i,k) is a hexagonal bin of side $2/(\sqrt{3}\,C^k i)$ and a *t-bin* of type (i,k) is a trapezoidal bin created by cutting an h-bin of type (i,k) in half with a cut parallel to two of its sides. Notice that h-bins and t-bins are only used to pack small circles. Also, the algorithm will divide a bin into h-bins of type $(i,0)$. This is done by selecting a hexagonal tiling of a bin where there is a hexagon on the left bottom part of the bin with two of its sides parallel to the bottom of the bin and by removing the hexagons that are not properly contained in the bin. See Figure 3.1a. The algorithm also divides h-bins and t-bins in additional sub-bins. In Lemma 3.2.1, we show how this can be done without losing any area of the original sub-bin.



(a) bin (b) h-bin (c) t-bin

Figura 3.1: In (a), a division of a bin in h-bins of type $(i,0)$. In (b), a subdivision of an h-bin of type (i,k) and in (c) a subdivision of a t-bin of type (i,k) in h-bins and t-bins of type $(i,k+1)$ for $C = 3$.

Lemma 3.2.1. For $M \leqslant i < CM$ and $k \geqslant 0$, if C is a multiple of 3 then it is possible to partition an h-bin (or a t-bin) of type (i,k) in h-bins and t-bins of type $(i,k+1)$.

Proof. Consider an h-bin of type (i,k) scaled so that its side length is C and embedded in the plane with its center at the origin, with two sides parallel to the x-axis. Notice that, after the scaling, an h-bin of type $(i,k+1)$ has side length 1. Finally, consider the hexagonal packing of the plane where the hexagons has side length 1 and have two sides parallel to the x-axis and, also, there is a hexagon with its leftmost point at $(0,0)$.

Notice that, because C is a multiple of 3, we have that the leftmost point of the h-bin (at

$(-C, 0)$) is the leftmost point of a hexagon. Thus, the segment which goes from the h-bins' leftmost point to the leftmost point of its top (at $(-C/2, \sqrt{3}\,C/2)$), which has an angle of 60 degrees, either cuts hexagons in half or cuts between hexagons. Again, as C is a multiple of 3, it cuts a hexagon exactly in half at leftmost point of the h-bin's top, ending at the hexagon rightmost point of its top. Now, by reflection symmetry over the x-axis, the same is true for the segment which goes from the h-bins' leftmost point to the leftmost point of its base (at $(-C/2, -\sqrt{3}\,C/2)$). Finally, by rotational symmetry over 120 degrees, the same is true for all edges of the h-bin. See Figure 3.1b.

For a t-bin of type (i, k), the result follows from this observation along with the fact that we will also split the hexagons with y-coordinate 0 horizontally in half. See Figure 3.1c. □

Next, we present the algorithm.

When a large circle c of type i arrives:

1. If there is no empty c-bin of type i, close the current bin of type i, if any, and open a new bin of type i containing i c-bins of type i.
2. Pack c in an empty c-bin of type i.

When a small circle c of type (i, k) arrives:

1. If there is no empty h-bin of type (i, k) or empty sub-bin of type (i, k') with $k' < k$, close the current bin of type i, if any, and open a new bin of type i and divide it in h-bins of type $(i, 0)$.
2. While there is no empty h-bin of type (i, k), let k' be the largest number such that $k' < k$ and there is an empty sub-bin of type (i, k'). If there is an empty t-bin of type (i, k'), then let B be such t-bin, otherwise let B be an empty h-bin of type (i, k'). Subdivide B in type $(i, k' + 1)$ bins.
3. Pack c in an empty h-bin of type (i, k).

Theorem 3.2.2. For every $\varepsilon > 0$, there exists C such that the occupation ratio of a closed bin used for small circles of type i for some $M \leqslant i < CM$ is at least $\left(1 - \frac{5.89}{M}\right) \frac{\pi}{\sqrt{12}} \frac{M^2}{(M+1)^2} - \varepsilon$.

Proof. For a set of circles of area A packed in a bin, we will call the value $1 - A$ by *area loss*. We will bound the area loss due to subdividing the bin in hexagons, due to empty sub-bins and due to packing circles in hexagons. First, notice that when dividing a bin in h-bins of type $(i, 0)$

we lose an area of at most $2 + 1/2$ times the side of the h-bin on the left and the right of the bin plus $1 + 1/2$ times the height of the h-bin on the bottom and the top of the bin. Therefore, the area loss due to the borders of a bin is at most $2(5/2 + 3\sqrt{3}/2)/(M\sqrt{3}) \leqslant 5.89/M$.

We say that a sub-bin is a *sister* of another sub-bin if they were generated by the same subdivision of a sub-bin. An empty t-bin is said to be *good* if it has an empty sister h-bin. Otherwise, we say that such t-bin is *bad*. Notice that when we generate an h-bin of type (i, k), there was no empty h-bin of type (i, k). That is, every empty h-bin of type (i, k) in a closed bin was generated by the same subdivision of a sub-bin. Also, notice that every empty good t-bin of type (i, k) was generated by the same subdivision of a sub-bin, since if two of those t-bins were generated by different subdivisions, then we would have empty h-bins of type (i, k) generated by different subdivisions, contradicting the previous argument.

Since every empty h-bin and every empty good t-bin of same type/subtype is in the same sub-bin, the area loss in level $k > 0$ is at most A_k, where A_k is the area of an h-bin of type (i, k). Summing over all $k \geqslant 1$ (since there is no empty h-bin of subtype 0 when we close a bin) and using the fact that $A_k = A_0/C^{2k}$ and that $A_0 = 2\sqrt{3}/i^2 \leqslant 2\sqrt{3}/M^2$, we have that the area loss from empty h-bins and empty good t-bins is at most $\sum_{k\geqslant 1} A_k = \sum_{k\geqslant 1} A_0/C^{2k} = A_0/(C^2 - 1) \leqslant 2\sqrt{3}/(M^2(C^2 - 1))$.

After we remove empty h-bins and empty good t-bins, we are left only with empty bad t-bins and undivided non-empty h-bins (disregarding all sub-divided sub-bins). Consider a set of bad t-bins contained in a specific sub-bin. Since such t-bins are bad, by definition there is no empty h-bin in such sub-bin. Also, since there are empty t-bins in this sub-bin, there is no sub-divided h-bin in this sub-bin, as the algorithm prioritizes the division of t-bins. That is, in such sub-bin every h-bin is non-empty and undivided. Let h_h and t_h be the number of h-bins and t-bins generated by the subdivision of an h-bin and let h_t and t_t be the number of h-bins and t-bins generated by the subdivision of a t-bin. We conclude that in this sub-bin, the occupation ratio of undivided non-empty h-bins is at least $\min\{2h_h/(2h_h + t_h), 2h_t/(2h_t + t_t)\}$, as the area of an h-bin is twice the area of a t-bin of same type and, by Lemma 3.2.1, the area of a h-bin or a t-bin is the sum of the area of its sub-bins. Also, notice that in a sub-division of an h-bin of type (i, k) the number of h-bins of type $(i, k + 1)$ is at least twice the number of h-bins of type $(i, k + 1)$ in a sub-division of a t-bin of type (i, k), that is, $2h_t \leqslant h_h$. Since the area of an h-bin of type (i, k) is twice the area of a t-bin of type (i, k), we have that $2h_h + t_h = 2(2h_t + t_t)$. From this, we conclude that $2h_t/(2h_t + t_t) = 4h_t/(2h_h + t_h) \leqslant 2h_h/(2h_h + t_h)$. That is, the occupation ratio of undivided non-empty h-bins in sub-bins where there are bad t-bin is at least $2h_h/(2h_h + t_h)$.

Consider now a small circle of radius r of type (i, k) packed in an h-bin. We have that the side length of the h-bin is $2/(\sqrt{3}\,C^k i)$, from where we conclude that the area of the h-bin is $2\sqrt{3}/(C^{2k}i^2)$. Also, notice that $r > 1/(C^k(i + 1))$, and we conclude that the area of the circle is at least $\pi/(C^{2k}(i + 1)^2)$. From this, we have that the occupation ratio is at least $(\pi/(C^{2k}(i + 1)^2))/(2\sqrt{3}/(C^{2k}i^2)) \geqslant \pi M^2/(\sqrt{12}(M + 1)^2)$. So, we have that the occupation ratio of small circles in closed bins is at least

$$\left(1 - \frac{5.89}{M} - \frac{2\sqrt{3}}{M^2(C^2 - 1)}\right) \frac{2h_t}{2h_t + t_t} \frac{\pi}{\sqrt{12}} \frac{M^2}{(M + 1)^2}.$$

But now, notice that as C grows, $2\sqrt{3}/(M^2(C^2-1))$ goes to zero and $2h_t/(2h_t+t_t)$ goes to 1, since for large C we have a small number of t-bins relative to the number of h-bins in a sub-divided sub-bin. This happens because the t-bins are generated only at the borders of a sub-bin. From those observations, we have that, as C goes to infinity, the occupation ratio of a closed bin used for small circles of type i for some $M \leqslant i < CM$ goes to $(1-5.89/M)\pi M^2/(\sqrt{12}(M+1)^2)$. □

3.3 Competitive Ratio Analysis

We compute an upper bound for the asymptotic competitive ratio of the algorithm as well a lower bound for every bounded space online algorithm using a weighting function, as previously done in [31, 60, 91] for other packing problems.

We start by defining a weighting function w. For a circle of radius r, if it is a large circle of type i then its weight is $1/i$ and if it is a small circle of radius r, then its weight is $(\pi r^2)/\alpha$ where $\alpha = \left(1-\frac{5.89}{M}\right)\frac{\pi}{\sqrt{12}}\frac{M^2}{(M+1)^2}$. Also, for a set $\mathcal{C}$ of circles, let $w(\mathcal{C}) = \sum_{c\in\mathcal{C}} w(c)$.

Theorem 3.3.1. The presented algorithm is β-competitive where β is the supremum over the weights of sets of circles that can be packed in a bin.

Proof. Let $\mathcal{C}$ be a set of circles, S be the number of bins used by the algorithm, O be the number of open bins at the end of the algorithm's execution and $\mathrm{OPT}(\mathcal{C})$ be the number of bins used by an optimal offline solution. Notice that, for every closed bin generated by the algorithm, if B is the set of circles of such bin, then $w(B) \geqslant 1$. In fact, a closed bin has either i large circles of type i or small circles occupying at least $\alpha = \left(1-\frac{5.89}{M}\right)\frac{\pi}{\sqrt{12}}\frac{M^2}{(M+1)^2}$ of the area and, for every circle in B, if its area is a then its weight is a/α. From this, we conclude that $S - O \leqslant w(\mathcal{C})$. Also, notice that $w(\mathcal{C})/\beta$ is a lower bound on the value of $\mathrm{OPT}(\mathcal{C})$, since every configuration of an optimal solution has weight at most β. Combining those facts, we have that $S \leqslant w(\mathcal{C}) + O \leqslant \beta\mathrm{OPT}(\mathcal{C}) + O$. Notice that O is at most a constant, from where we conclude that the algorithm is β-competitive. □

To compute a lower bound, we also use weights. A circle c has weight $\omega(c) = 1/i$ if we can pack i copies of c in a bin but we cannot pack $i+1$ copies of c in a bin. As before, for a set $\mathcal{C}$ of circles, let $\omega(\mathcal{C}) = \sum_{c\in\mathcal{C}} \omega(c)$. Also, let $A(c)$ be the area of a circle c and $A(\mathcal{C}) = \sum_{c\in\mathcal{C}} A(c)$.

Theorem 3.3.2. If $\mathcal{C}$ is a set of circles that can be packed in a bin then every bounded space online algorithm has competitive ratio at least $\omega(\mathcal{C}) + \sqrt{12}\,(1 - A(\mathcal{C}))\,/\pi$.

Proof. Let $0 < \varepsilon < 1$ be a constant. We will show that every bounded space online algorithm has competitive ratio at least $\omega(\mathcal{C}) + \sqrt{12}\,(1 - A(\mathcal{C}))\,/\pi - \varepsilon$. For a circle c, let $r(c)$ denote its radius. Also, let $\delta = \pi\varepsilon/(2\sqrt{12})$ and γ be a constant smaller than δ. We will construct a sequence $\{\mathcal{C}_0, \mathcal{C}_1, \ldots\}$ until we obtain a set $\mathcal{C}_k$ such that $1 - A(\mathcal{C}_k) \leqslant \gamma$, where $\mathcal{C}_0 = \mathcal{C}$ and, for $n \geqslant 1$, $\mathcal{C}_n$ is as described below.

For $n \geqslant 1$, let $\ell_n \leqslant \gamma/(4\pi\sum_{c\in\mathcal{C}_{n-1}} r(c) + 4\pi|\mathcal{C}_{n-1}| + (5+3\sqrt{3})/2)$ be constants such that $\ell_n < \ell_{n-1}$ if $n > 1$ and ℓ_1 is smaller than the radius of any circle of $\mathcal{C}_0$. The idea is to use ℓ_n

as the side length of a hexagonal packing used to pack small circles in the empty regions in the packing generated to $\mathcal{C}_{n-1}$. Let $n \geqslant 1$ and suppose that $\mathcal{C}_{n-1}$ can be packed. Fix a packing of $\mathcal{C}_{n-1}$ and consider a hexagonal tilling of the bin with hexagons of side length ℓ_n where there is a hexagon on the left bottom part of the bin. We say that a hexagon is feasible if it is not intersected by the interior of a circle of $\mathcal{C}_{n-1}$ or the border of the bin. Add to the packing of $\mathcal{C}_{n-1}$ circles of radii $\sqrt{3}\,\ell_n/2$ in every feasible hexagon and let $\mathcal{C}_n$ be the union between $\mathcal{C}_{n-1}$ and the new circles packed in hexagons. Notice that, if $\mathcal{C}_{n-1}$ can be packed in a bin, then so do $\mathcal{C}_n$. Notice that the total area of infeasible hexagons that intersect the border of the bin is at most $(5+3\sqrt{3})\ell_n/2$ (as in Theorem 3.2.2). Also, notice that if a hexagon intersects the interior of a circle of radius r centered at a point p then it is properly contained in the circle of radius $r+2\ell_n$ centered at p. Thus, the total area of feasible hexagons is at least

$$\begin{aligned}
&1-(5+3\sqrt{3})\ell_n/2-\sum_{c\in\mathcal{C}_{n-1}}\pi(r(c)+2\ell_n)^2\\
&=1-(5+3\sqrt{3})\ell_n/2-A(\mathcal{C}_{n-1})-4\pi\ell_n\sum_{c\in\mathcal{C}_{n-1}}r(c)-4\pi\ell_n^2|\mathcal{C}_{n-1}|\\
&>1-(5+3\sqrt{3})\ell_n/2-A(\mathcal{C}_{n-1})-4\pi\ell_n\sum_{c\in\mathcal{C}_{n-1}}r(c)-4\pi\ell_n|\mathcal{C}_{n-1}|\\
&\geqslant 1-A(\mathcal{C}_{n-1})-\gamma,
\end{aligned}$$

where the first inequality follows from the fact that $\ell_n \leqslant 1$. Now, since every new circle occupies an area of $\pi/\sqrt{12}$ of the area of the hexagon, we have that $A(\mathcal{C}_n) \geqslant A(\mathcal{C}_{n-1}) + (1 - A(\mathcal{C}_{n-1}) - \gamma)\pi/\sqrt{12}$. It follows, from induction, that $A(\mathcal{C}_n) \geqslant (1-\gamma)[1-(1-\pi/\sqrt{12})^n]$. Now, as $(1-\pi/\sqrt{12})^n$ goes to 0 as n goes to infinity and $\gamma < \delta$, there exist $k \geqslant 0$ such that $A(\mathcal{C}_k) \geqslant (1-\delta)$. Also, it follows from transitivity that $\mathcal{C}_k$ can be packed in a bin and that $\mathcal{C}_0 \subseteq \mathcal{C}_k$.

Consider an instance of the Online Circle Packing Problem composed by the disjoint union of N copies of the set $\mathcal{C}_k$ where the circles are ordered by non-increasing radii. Suppose $\mathcal{C}_0$ has q_i circles of radii at most ρ_i^* and greater than ρ_{i+1}^*. Then, $\omega(\mathcal{C}) = \sum_{i\geqslant 1} q_i/i$. Also, let $t = |\{i\colon q_i > 0\}|$, that is, t is the number of different types of circles in $\mathcal{C}$. For $1 \leqslant i \leqslant k$, let r_i be the number of circles added to $\mathcal{C}_{i-1}$ in order to construct $\mathcal{C}_i$ and let a_i be the area of such a circle. Notice that, by the choice of ℓ_i, a circle in $\mathcal{C}_i \backslash \mathcal{C}_{i-1}$ has radius different from the radius of any other circle not in $\mathcal{C}_i \backslash \mathcal{C}_{i-1}$.

Any online algorithm with bounded space B uses at least $Nq_i/i - B$ bins for every type of circle in $\mathcal{C}_0$. This follows from the fact that circles are ordered and that one bin can hold at most i circles of type i. Also, as the hexagonal packing is optimal for packing circles in the plane, we have that any online algorithm with bounded space B uses at least $Nr_ia_i\sqrt{12}/\pi - B$ bins to pack the circles added to $\mathcal{C}_{i-1}$ in order to construct $\mathcal{C}_i$. Let $N \geqslant 2(t+k)B/\varepsilon$, any

algorithm with bounded space B utilizes at least

$$\begin{aligned}
&\sum_i \left(\frac{Nq_i}{i} - B\right) + \sum_{i=1}^{k} \left(Nr_i a_i \frac{\sqrt{12}}{\pi} - B\right) \\
\geqslant\ & N\left(\omega(\mathcal{C}) + \left(1 - A(\mathcal{C}) - \frac{\pi\varepsilon}{2\sqrt{12}}\right)\frac{\sqrt{12}}{\pi}\right) - (t+k)B \\
=\ & N\left(\omega(\mathcal{C}) + (1 - A(\mathcal{C}))\frac{\sqrt{12}}{\pi}\right) - N\varepsilon/2 - (t+k)B \\
\geqslant\ & N\left(\omega(\mathcal{C}) + (1 - A(\mathcal{C}))\frac{\sqrt{12}}{\pi}\right) - N\varepsilon,
\end{aligned}$$

while an optimal offline solution uses only N bins, from where the result follows. □

3.4 Numerical Results

In order to obtain numerical results for the bounds presented in Sect. 3.3, we combine Integer Linear Programming with Constraint Programming. Constraint Satisfaction Problems (CSP) are defined by a set of variables $X = \{x_1, \ldots, x_n\}$, a finite set $Dom(x_i)$, for each variable x_i, called domain of x_i, with all possible values this variable can assume. Also, a set of constraints restricts the values variables can simultaneously assume. A solution for this problem is an assignment of values to all variables within their domains, that satisfy all constraints. Solutions to a CSP are found by systematically exploring possible assignments. For more information about constraint programming we suggest [82].

An upper bound for the competitive ratio of the algorithm can be given by the following integer linear program:

$$\begin{aligned}
&\text{maximize } y/\alpha + \sum_{i=1}^{K}\sum_{t=1}^{i} t\, w_i\, x_i^t \\
&\text{subject to} \qquad \sum_{i=1}^{t} x_i^t \leqslant 1 \qquad \forall\, 1 \leqslant i \leqslant K \\
&\qquad\qquad y + \sum_{i=1}^{K}\sum_{t=1}^{i} t\, a_i\, x_i^t = 1 \\
&\qquad\qquad\qquad x_i^t \in \{0,1\} \quad \forall\, 1 \leqslant i \leqslant K, \forall\, 1 \leqslant t \leqslant i
\end{aligned}$$

where x_i^t is a binary variable indicating if we have t large circles of type i, y is an upper bound on the area of small circles, a_i is a lower bound on the area of a circle of type i, w_i is the weight of a large circle of type i and $\alpha = \left(1 - \frac{5.89}{M}\right)\frac{\pi}{\sqrt{12}}\frac{M^2}{(M+1)^2}$. We solve this model using a branch-and-cut algorithm where, whenever an integer solution is found, its feasibility is tested using a constraint programing model. If such test fails, we cut out this solution.

In order to test if a given set $\mathcal{C} = (r_1, \ldots, r_n)$ of circles can be packed in a bin using

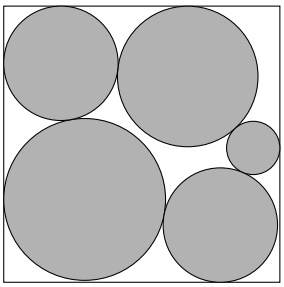


Figura 3.2: A packing with one circle of type 1 (radius 0.292893), one circle of type 2 (radius 0.254333), two circles of type 4 (radius 0.207106) and one circle of type 25 (radius 0.096362). The total weight of these circles is 2.04 and they occupy an area of 0.77139. Thus, we obtain a weight of 0.2520 for the remaining area for a total weight of 2.2920. By choosing $\gamma = 0.9975$, we have that $W = 4157$ and the circles (after scaling by W) can be packed at positions $(1218, 1247)$, $(2770, 3098)$, $(861, 3296)$, $(3261, 861)$, and $(3756, 2022)$, respectively.

constraint programming, we have to do a discretization of the search space. Let $(r'_1, \ldots, r'_n)$ be a set of circles such that $r'_i = \gamma r_i$ for some $0 < \gamma < 1$. Also, let $\underline{r} = \min\{r_i \colon 1 \leqslant i \leqslant n\}$ and δ be a constant such that $\delta \leqslant (1 - \gamma^2)\underline{r}/2$. It follows from simple calculations that if $(r'_1, \ldots, r'_n)$ cannot be packed in a bin using a grid of granularity δ, then $(r_1, \ldots, r_n)$ cannot be packed in the bin (with general positions). Thus, given a $\gamma < 1$, we scale the 1×1 bin to a $W \times W$ bin, such that $W = \lceil 1/\delta \rceil$ and we scale the radius of each circle to $r'_i = \lfloor r_i \, \gamma \, W \rfloor$. If these circles do not have a feasible packing into the $W \times W$ bin, the original circles also do not have a feasible packing into the 1×1 bin. Thus, in the constraint programming model we have, for every circle i, integer variables x_i and y_i that indicates the position where circle i is packed. The domain for each variable is $Dom(x_i) = Dom(y_i) = [r'_i, \ldots, W - r'_i]$. This domain assures that circles do not exceed the limits of the bin. Non-overlapping constraints are also necessary: For each par of circles i, j, $i \neq j$, we add to the model the constraint $(x_i - x_j)^2 + (y_i - y_j)^2 \geqslant (r'_i + r'_j)^2$. Also, we break symmetries by considering that the center of largest circle of the set $\mathcal{C}$ is in the first quadrant, the center of second largest circle is above the diagonal, at least one circle is packed in its left most position, at least one circle is packed in its bottom most position, and for two circles C_i and C_j having the same radius with $i < j$ by considering that C_i is at the left of C_j or below C_j if they have the same x-coordinate. Using this combination of integer programming and constraint programming with $M = 59$ and, thus, $K = 992$, we obtained an upper bound solution with value 2.4394 that is composed by circles of type: 1, 2, 4 (2 circles), 5, 6, 16, 111, 987.

In order to compute the lower bound, we use an integer programming model similar to the one presented before, with an objective function that minimizes the total area of the circles, with a cutoff in the value of the set, which we gradually increase. We use the presented constraint programming method to find a feasible packing of a set $\mathcal{C}$ of circles, where the radius of each circle is scaled to $r'_i = \lceil r_i \, W \rceil$ for a given W. Figure 3.2 presents the lower bound found of value 2.2920.

3.5 Final Remarks

In this paper we present a bounded space algorithm for the Online Circle Packing Problem, which has a competitive ratio of 2.4394. To our knowledge, this is the first algorithm for such problem in the literature. We also present a lower bound of 2.2920 for any bounded space algorithm for such problem. Notice that these bounds can possibly be improved using different techniques. Also, to our knowledge, this is the first paper to use Constrained Programming as heuristic to test the feasibility of packing a set of circles in a square bin. We believe that this could be useful for developing algorithms for other applications such as to solve the circle packing problem in practice. Finally, we believe that the technique used to recursively pack small circles through the usage of hexagons can be of independent interest.

Capítulo 4

Two-dimensional Disjunctively Constrained Knapsack Problem: Heuristic and Exact approaches *

ABSTRACT

This work deals with the 0-1 knapsack problem in its two-dimensional version, considering a conflict graph, where each edge in this graph represents a pair of items that must not be packed together. This problem arises as subproblem of the bin packing problem and in supply chain scenarios. We propose some integer programming formulations that are solved with a branch-and-cut algorithm. The formulation is based on location-allocation variables mixing the one- and two-dimensional versions of this problem. When a candidate solution is found, a feasibility test is performed by a constraint programming algorithm, which verifies if it satisfies the two-dimensional packing constraints. Moreover, bounds and valid cuts are also investigated. A heuristic that iteratively generates a solution using a greedy randomized procedure is also proposed. In order to avoid local optimal solutions, a memory list is used, as well as different packing strategies over a grid of points for the heuristic. The results are extended in order to consider complete shipment of items, where subsets of items have all to be loaded or left out completely. This constraint is applied in many real-life packing problems, such that packing parts of machinery, or when delivering cargo for different clients. Experiments on several instances derived from the literature indicate the competitiveness of some algorithms, which solved 99% of the instances to optimality requiring low runtime.

KEYWORDS. two-dimensional 0-1 knapsack problem, conflict graph, disjunctive constraint, complete shipment, integer programming.

*We would like to thank CNPq, FAPEG and FAPESP (proc. 2011/13382-3 and 2013/21744-8) due to financial support, and François Clautiaux that provided the source code Clautiaux et al. [16].

4.1 Introduction

This work deals with the 0-1 knapsack problem in its two-dimensional version, considering a conflict graph, in which each edge in this graph represents a pair of items that must not be packed together, also referenced as disjunctive constraints. We also consider complete shipment of items, in which items are partitioned in sets and we either pack all items of one such set or none at all. The 0-1 knapsack problem is a classical combinatorial optimization problem that appears in several real-world applications as well as a subproblem of other optimization problems as the bin packing. Surveys on cutting and packing problems can be found in Bentz et al. [9], Lodi et al. [65], Sweeney and Paternoster [87], Wäscher et al. [93]. Some variants of the knapsack problem are the two-dimensional (2D) with the packing in pallets and the cut of wood sheets; and, the three-dimensional, when dealing with containers [12]. Both variants are NP-hard [38].

A practical constraint is related with pairs of items that must not be packed together in the same bin due to various conditions (for example, food and toxic goods), thus we have to select a subset of non-conflicting items that can be packed in the bin [98, 99]. This constraint is also referred to as *disjunctive constraint* and is associated with a conflict graph.

We investigate the 2D 0-1 disjunctively constrained knapsack problem (2D-DCKP). A conflict graph is used, so an edge in this graph represents that its vertices (items) are in conflict, and only one of them may be packed in the bin. The items are rectangular goods with length, width and a non-negative profit (value). The bin is also rectangular. The aim is to pack orthogonally to the sides of the bin a subset of non-conflicting items of maximum value. It is required that items must not overlap each other and have to respect the bin's dimensions. Rotations of items are not allowable.

For the 2D-DCKP, some integer formulations, which are solved with a branch-and-cut algorithm improved with valid bounds and cuts, are proposed. First, an initial formulation is presented, which is then relaxed following a location-allocation model, similar to capacitated facility-location problems [34], where we first search for subsets of non-conflicting items of maximum value, and next verify if these subsets can be allocated in the bin. The latter is performed by searching for a solution to a one-dimensional contiguous bin packing problem, inspired by the model presented by Côté et al. [19]. If no such solution exists, then there is no solution to the corresponding two-dimensional orthogonal packing problem. Otherwise, it is solved a constraint programming based model, which verifies the feasibility of the candidate solution, so cuts based on them are added in the branch-and-bound tree.

A heuristic is also proposed, where its solutions are generated iteratively using a greedy randomized procedure. In this heuristic, a memory list is used to prevent repeated solutions and some strategies to pack items are considered, as well as a diversification criteria that changes the direction of the search and avoids local optima solutions.

We also extend these algorithms in order to consider complete shipment of items. In this case, we consider subsets of items that must be packed or left out completely, and conflict happens between different subsets. The computational results reported here show that the proposed algorithms found optimal solutions very fast for almost all the instances considered for both the

2D-DCKP and with the complete shipment constraint.

4.1.1 Literature review

The 0-1 disjunctively constrained knapsack problem has been first investigated in the one-dimensional version (1D-DCKP). Vance et al. [92] consider it as a sub-problem in a cutting stock problem. Yamada et al. [99] presents a lower bound obtained by a heuristic that generates an initial solution with a greedy procedure, and then improves such a solution with a neighborhood search procedure based on 2-opt operations. These bounds are used in a branch-and-bound exact algorithm capable of solving instances with up to 1,000 items and 10,000 pairs of conflicting items.

Hifi and Michrafy [49] also investigated the 1D-DCKP. They proposed a reactive local search algorithm that combines a memory list to prevent cycling on solutions with a degrading strategy to avoid local optimal solutions. This algorithm solved large-sized instances very quickly. On the other hand, Hifi and Michrafy [50] considered exact approaches combined with heuristics and relaxations of integer programming models in order to solve medium-sized instances. Two of the three exact approaches are two-phase procedures: first, a reduction phase that fix decision variables, and next an integer model is solved with a branch-and-bound algorithm. The last is a modified dichotomous algorithm applied over reduced intervals of search, where dominating and covering constraints are added. Approximation algorithms, including fully polynomial time approximation schemes, were proposed by Pferschy and Schauer [78] for special classes of conflict graphs, in particular for chordal, tree and bounded treewidth graphs.

Other approaches for the 1D-DCKP were proposed in Akeb et al. [1], Hifi and Otmani [51]. Akeb et al. [1] developed local-branching based algorithms. Three versions were proposed: the first one is a rounding procedure, which works on the fractional solution of a given integer model; the second one is a two-phase method that first tries to fix variables of a fractional solution, and next an exact solver is applied in the reduced problem; and, the last one is a modification of the second algorithm considering an intensification procedure. Hifi and Otmani [51] presented two scatter search based heuristics which differ each other in the way that solutions are combined. The first uses a greedy strategy based on the structure and the relative profit per weight-degree values of items, while the second one considers a 3-opt search procedure on the neighborhood of feasible solutions.

Recently Hifi [47] and Hifi et al. [52] proposed heuristics to tackle the 1D-DCKP. In Hifi [47], a hybrid guided neighborhood search is presented, in which initial solutions are built from independent sets and next are converted to a feasible solution for the problem. Then, a descent method and an ant colony optimization method are applied in order to improve the solution. Hifi et al. [52] developed a heuristic based on a iterative rounding search. First, fractional variables of a linear relaxation are rounded to get a feasible solution that is next improved by applying a neighbor search. Besides that, constraints based on cardinality of valid sets and lower bounds on the objective function are considered during the resolution.

For the complete shipment constraint, Bortfeldt and Wäscher [12] conclude that this constraint has been neglected by the literature. To the best of our knowledge, there is only the work

of Eley [29] that considers such constraint with conflicting items, where is proposed a heuristic that generates packing patterns that are next used an integer programming formulation. In this paper we consider the complete shipment constraint in the 2D-DCKP and show how the algorithms proposed for the latter can be easily adapted for the former.

Note that approaches for the 2D-DCKP can also be used to solve the case without conflict graphs, namely the 2D 0-1 knapsack problem (2KP). Moreover, the 2D-DCKP can appear as a subproblem (in branch-and-price algorithms) when solving the bin packing with conflicts which was first studied in the one-dimensional version by Jansen and Ohring [59]. Now, special attention has been given for the 2D version of this bin packing problem. Epstein et al. [33] considered the version with squares and for specific graph classes, bipartite and perfect, they proposed approximation algorithms. For bipartite graphs, their algorithm has an approximation ratio of $2 + \epsilon$ for $\epsilon > 0$, while for perfect conflict graphs the approximation factor is at most 3.2744. The first heuristic for such problem was proposed in Khanafer et al. [62]. It is based on decomposing techniques and is valid for general conflict graphs. The main idea is to compute a tree-decomposition of the complement of G, called compatibility graph $\bar{G}$, and corresponds to find a triangulation of $\bar{G}$. Next, each item is assigned to a cluster (which can be viewed as a reduced instance), however an item may belong to different clusters, so a partitioning problem is solved with greedy heuristics and a tabu list based algorithm. Finally, partial solutions are combined on a unique solution, where items of some bins are redistributed for other ones in order to reduce the total number of used bins.

In the roll of good strategies for 2D cutting and packing problems, Baldacci and Boschetti [4] solved the two-dimensional orthogonal non-guillotine cutting problem, that generalizes the 2KP. First, they computed a good upper bound applying several reductions on the problem, based on area and value, so fixing items in the final solution. Their main algorithm is a two-level method. At the first level, a relaxed version of the problem involving knapsack, dominance and cover inequalities is solved. When an integer solution is achieved, a feasibility test is then performed. The feasibility test asks if a set of rectangular items can be arranged orthogonally in a rectangular bin, and it is referenced in the literature as the NP-complete two-dimensional orthogonal packing problem (2OPP).

Fekete et al. [36] presented a graph-theoretic approach that relied on interval graphs when searching for feasible solutions. They provided a branch-and-bound algorithm, in which many redundancies in the tree search are removed. Clautiaux et al. [15] proposed two algorithms: the first, a branch-and-bound method based on the algorithm in Martello and Vigo [67], in which items are packed at the leftmost-downward feasible point; the second is a two-step branch-and-bound method, in which for all feasible solutions of a relaxed problem, an inner procedure tries to obtain a solution for the original problem. The relaxation of the second method consists in slice each item into strips of equal size. All strips of a given item has to be packed at the same $x-$position. The inner procedure searches for feasible $y-$positions that then allows to solve the original problem.

In Clautiaux et al. [16] there is a branch-and-bound algorithm based on constraint programming, in which the 2OPP is modeled as a constraint-based scheduling model. Constraint

propagation rules for scheduling problems are then derived to tighten the domain of variables and check for packing inconsistencies. Mesyagutov et al. [72] improved the algorithm in Clautiaux et al. [16] using refined set of points, new branching strategies and pruning tests based on LP-based rules. Branching strategies were defined on coordinates of the items, domain of the variables and overlapping relations.

Côté et al. [19] solved the 2OPP version with the addition of unloading constraints, which appears commonly in integrated vehicle routing and loading problems. The feasibility test considered in this paper is performed by first solving a relaxed model of the 2OPP, based on the contiguous bin packing problem [19]. This relaxation may prove that the candidate solution is unfeasible, and if not, the algorithm in Clautiaux et al. [16] is called, which gives an exact answer.

The paper is organized as follows: Section 4.2 describes the problem and discusses how to update item sizes without loss of generality; Our heuristic is detailed in Section 4.3, which details about the memory list and how the items are packed are presented. In Section 4.4 we present some integer formulations for the 2D-DCKP, which are solved with a branch-and-cut approach. Moreover, feasibility tests and valid cuts are also presented in this section. In Section 4.5 we extend the 2D-DCKP and our algorithms to consider the complete shipment constraint. Computational experiments on the algorithms are reported in Section 4.6 showing that the algorithms are very competitive. Finally, conclusions and directions for further works are given in Section 4.7.

4.2 Preliminary Discussion

In this section we formally describe the two-dimensional disjunctively constrained knapsack problem, how we can lift the items size to reduce computation time and how we discretize the bin.

4.2.1 Problem description

Given a bin B of dimensions (L, C), with area A_{bin},and a set V of n items, where each item $i \in V$ has dimensions (l_i, c_i), value (profit) v_i and area a_i, for $i = 1, \ldots, n$, and a conflict graph $G = (V, E)$, where each edge $\{i, j\} \in E$ represents that i and j are conflicting items, the 2D-DCKP is the problem of finding a subset $V^* \subset V$ of items of maximum total profit, and a feasible packing of items V^* in B. A solution of the 2D-DCKP is represented in the Cartesian Plane $\mathbb{R}^2$, with the origin at $(0, 0)$, and the position of an item is given by its bottom left corner point. An instance of the 2D-DCKP is given by $I = (L, C, V, l, c, G)$, we consider that all input data are positive integers.

4.2.2 Lifting item sizes

We adapt the procedure described in Boschetti and Mingozzi [13] to increase the item sizes without losing optimal solutions, in order to improve the quality of the lower and upper bounds. For each item $i \in V$ consider the problem (4.1) which searches for a subset of non-conflicting items with i of maximum sum of their lengths satisfying $L - l_i$.

$$
\begin{aligned}
L_i^* = \quad & \max \sum_{k \in V} l_k \tau_k \\
& \\
subject\ to: & \\
(i) \quad & \sum_{k \in V} l_k \tau_k \leqslant L, \\
& \\
(ii) \quad & \tau_j + \tau_k \leqslant 1, \qquad \forall\ (j,k) \in E, \\
& \\
(iii) \quad & \tau_i = 1, \\
& \\
(iv) \quad & \tau_k \in \{0,1\}, \qquad \forall\ k \in V \backslash \{i\}.
\end{aligned}
\tag{4.1}
$$

Boschetti and Mingozzi [13] show, when there is no disjunctive constraints, that for any feasible solution containing item i there is an equivalent solution for i with new length $l_i + (L - L_i^*)$. After updating the size of i, the items in the optimal solution with value L_i^* will not have their lengths modified due to constraint (4.1.i). Trying to update as many item sizes as possible, we sort the set of items in non-increasing order of length and update items in this order without updating items used in solutions of previous increased items. Similar task can be performed for updating item widths. From now on, we consider that the instance was preprocessed using the above procedure.

4.2.3 Finding Independent Sets

A competitive branch-and-bound algorithm to compute the maximum valuable clique in a graph was proposed in Niskanen and Ostergard [77]. We adapted this algorithm in order to compute such clique while considering the items' areas. The algorithm $cliquer$, with parameter H and A, finds a clique K of H with maximum value and such that $\sum_{k \in K} a_k \leqslant A$. Thus, one can find a candidate solution to 2D-DCKP by computing $cliquer(\bar{G}, A_{bin})$, where $\bar{G}$ represents the complement of graph G. Notice that if this candidate solution can be packed in the bin, then it is indeed optimal. This algorithm is shown to be faster than solving 1D-DCKP solely with an integer programming solver. This algorithm is used used to find valid cuts as showed in section 4.4.3.

4.2.4 Packing items

In order to arrange a set of items, we discretize the bin in a grid of points, and pack items at the left-bottom most position of the grid. A position is feasible if an item can be arranged there generating a packing without overlap between any pair of items and respecting the bin's dimensions.

In accordance with Herz [45], there is no loss of generality for considering the *discretization points* for the grid of a two-dimensional packing problem. Observe that we can use the discretization points in the 2D-DCKP, since conflict graphs do not impose any restriction on how to pack items, but that some items must not be packed together in the same bin.

A discretization point of the length (resp., width) is a non-negative integer $d \leqslant L$ (resp., $e \leqslant C$) that is a non-negative binary combination of the sizes in $l = (l_1, \ldots, l_n)$ (resp., $c = (c_1, \ldots, c_n)$). The set of discretization points in the length (resp., width) is denoted by P (resp., Q). Note that what we call discretization points, are in fact one-dimensional coordinates.

For guillotinable and L-packing patterns, an improvement of the discretization points was proposed by Scheithauer and Terno [85] and Queiroz et al. [80], respectively, resulting in the set of *reduced raster points*. Reduced raster points are computed from P and Q as:

$$\begin{aligned} \tilde{P} &:= \{\langle L - r \rangle_P \mid r \in P\}, \quad \text{where } \langle s \rangle_P = \max\{t \in P \mid t \leqslant s\}; \\ \tilde{Q} &:= \{\langle C - u \rangle_Q \mid u \in Q\}, \quad \text{where } \langle a \rangle_Q = \max\{b \in Q \mid b \leqslant a\}, \end{aligned} \tag{4.2}$$

where $\tilde{P}$ (from P) and $\tilde{Q}$ (from Q) are the set of reduced raster points of the length and width, respectively. Queiroz et al. [79] presented the algorithm RRP to compute the reduced raster points. First, discretization points are computed using a dynamic programming and next the reduced raster points are obtained from equation (4.2).

Based on Birgin et al. [10] and Queiroz et al. [80], it is believed that there is no loss of generality when considering the set of reduced raster points for any packing problem, although no demonstration of this statement has been given in the literature. Nonetheless, in our integer formulations we use the discretization points, thus guaranteeing that these indeed find an optimal solution and we use the reduced raster points only in the heuristic proposed.

4.3 A Greedy Randomized Heuristic

In the next subsections we describe a heuristic, which we call of GR. First, we give an overview and next discuss how it works considering routines for packing and maintain a memory list. A routine to repack items is discussed, which aims to construct different packings, that is, with the same profit, but different layouts.

4.3.1 Heuristic Overview

In the GR there is an algorithm called PHASE ONE, which receives a solution of 2D-DCKP for instance I and tries to add more rectangles to it. We wrap this algorithm in another algorithm

called TWO PHASE, where we are given a solution and we repeatedly remove an item (accordingly to some criteria), reorganize the items inside the bin (using an algorithm called REPACK) and then we call the algorithm PHASE ONE in order to improve this solution. In order to avoid local minima, we use a *memory list*, which forbids an item to be packed in the same position as it was before removal by algorithm TWO PHASE for some iterations. Finally, we also wrap algorithm TWO PHASE in algorithm GR, where we repeatedly either consider an empty solution, discarding the solution from the previous iteration, or the solution from the previous iteration (after a diversification), which is latter improved using algorithm TWO PHASE.

During the algorithm, a memory list LP is used to keep a list of forbidden items and grid point combination. For an item i arranged at point p, selected to belong to LP, we consider the pair $\{degree_G(i), LP_i\}$ for which: $degree_G(i)$ states that i cannot be arranged in any solution for at least the degree of i at G; and, LP_i contains the list of blocked points, now including p, where i cannot be arranged.

During execution of GR, for an item i selected to be packed, first we check if $degree_G(i)$ is zero, if not, such value is decremented by one, otherwise next we take in consideration the penalized points of LP_i to arrange i at the grid. Follows that LP and LP_i (for $i = 1, \ldots, n$) are empty at each iteration of the GR's main loop.

4.3.2 Heuristic GR

For a new iteration, to construct a new solution, GR tries an acceptance probability function in order to start from an empty solution, or from the current solution after applying a diversification strategy. The best solution is updated if necessary. Algorithm 1 resumes the GR heuristic.

The diversification is applied if a random value picked off $[1, 120]$ is less than:

$$prob = \exp(-\tfrac{\Delta}{0{,}5 \times T}) \times 100. \tag{4.3}$$

Equation (4.3) follows the acceptance function used in the Simulated Annealing algorithm [63], for which we consider: $\Delta = profit(bestS) - profit(S)$, where $bestS$ is the best solution ever found; $T = MAX \times n^2$, which is decreased of 5% at each iteration.

4.3.3 Constructing a solution in two phases

The routine which constructs a solution iterates in two phases: the first one selects items using a greedy randomized procedure, and the second selects one item to add in LP.

In the first phase, items are selected from Lists of Candidates (LC) to generate a solution S. Each LC has a subset of items that are chosen at random from the input instance. So, for each LC, we select an item i with the largest ratio value/area, the smallest degree in G and that do not have any conflict with other items in S.

Once we select i, the packing routine is applied observing the memory list. In other words, if $i \in LP$, we search for the left-bottom most feasible point r not in LP_i to arrange i. If no such r exists, then i is not packed. The detailed routine is presented at Algorithm 2.

Algorithm 1: GR

Input : Instance $I = (B, V, G)$ of the 2D-DCKP; percentage q; sets $\tilde{P}$ and $\tilde{Q}$ of reduced raster points; value X and MAX of iterations.
Output: Solution for the 2D-DCKP.

1.1 $S \leftarrow \varnothing$; $bestS \leftarrow \varnothing$; $LP \leftarrow \varnothing$.
1.2 $rd \leftarrow 0$; $prob \leftarrow 0$; $T \leftarrow MAX \times n^2$.
1.3 **for** $c \leftarrow 1$ **to** MAX **do**
1.4 **if** $rd \geqslant prob$ **then**
1.5 $S \leftarrow \varnothing$.
1.6 **else**
1.7 Diversification: Add $\lfloor |S|/2 \rfloor$ items chosen from S at random to LP.
1.8 $S \leftarrow$ Solution of the **Two-phase** for input $(I, LP, S, q, \tilde{P}, \tilde{Q}, X)$.
1.9 **if** $value(S) > value(bestS)$ **then**
1.10 $bestS \leftarrow S$.
1.11 $rd \leftarrow$ a random value in the interval $[1, 120]$.
1.12 $\Delta \leftarrow profit(bestS) - profit(S)$.
1.13 $prob \leftarrow \exp(-\frac{\Delta}{0.5 \times T}) \times 100$.
1.14 $LP \leftarrow \varnothing$; $T \leftarrow T \times 0.95$.
1.15 **return** $bestS$.

Algorithm 2: PHASE ONE

Input : Instance $I = (n, B, V, G)$ of the 2D-DCKP; memory list LP; solution S; percentage q; sets $\tilde{P}$ and $\tilde{Q}$ of reduced raster points.
Output: Solution with value (possibly) improved.

2.1 $L \leftarrow V \backslash S$.
2.2 **while** $L \neq \varnothing$ **do**
2.3 $flag \leftarrow false$; *bestVA* $\leftarrow 0$; $bestD \leftarrow n$.
2.4 $LC \leftarrow$ choose at random $q\%$ items of L.
2.5 **foreach** $i \in LC$ **do**
2.6 **if** *bestVA* $> v_i$ **OR** $bestD < degree_G(i)$ **OR** $\exists\ (k, j) \in E$ *for* $k \in S$ **then**
2.7 $LC \leftarrow LC - \{i\}$.
2.8 **else**
2.9 *bestVA* $\leftarrow v_i$; $item \leftarrow i$; $bestD \leftarrow degree_G(i)$; $flag \leftarrow true$.
2.10 $L \leftarrow L - \{i\}$.
2.11 **if** $flag$ **then**
2.12 Pack $item$ in S, if it is possible.
2.13 **return** S.

For a solution S generated on phase one, now phase two selects an item j that meets the criteria cr below and adds it in the memory list. Next, it is applied the routine to repack $S - \{j\}$.

- cr = 1: item with smallest *(value/area)* $\times$ $(\min\{degree_G\})$;
- cr = 2: item with largest *(value/area)* $\times$ $(\max\{degree_G\})$;
- cr = 3: item with smallest *(value/area)*;
- cr = 4: item with largest *(value/area)*;
- cr = 5: item with smallest *value/(area* $\times$ $\max\{degree_G\})$;
- cr = 6: item with $\max\{degree_G\}$;
- cr = 7: item with largest *(value/area)* $\times$ $(\max\{degree_G\})$.

Algorithm 3 summarizes the two-phase procedure that aims to construct iteratively good solutions for the 2D-DCKP. Note that each criteria cr is applied consecutively following the main loop of lines $3.2 - 3.12$.

Algorithm 3: TWO PHASE

Input : Instance $I = (n, B, V, G)$ of the 2D-DCKP; memory list LP; solution S; percentage q; sets $\tilde{P}$ and $\tilde{Q}$ of reduced raster points; maximum number of iterations X.
Output: Solution for the 2D-DCKP.

3.1 $cr \leftarrow 1$; $type \leftarrow 1$; $bestS \leftarrow \varnothing$.
3.2 **for** $c \leftarrow 1$ **to** X **do**
3.3 $(j, p) \leftarrow$ apply cr in S and return the item j and point p where it is arranged.
3.4 **if** *exists* j **then**
3.5 $LP \leftarrow LP \cup \{j\}$; $LP_j \leftarrow LP_j \cup \{p\}$; $S \leftarrow S - \{j\}$.
3.6 $S \leftarrow Repack(I, type, S, \tilde{P}, \tilde{Q})$.
3.7 $S \leftarrow$ **Phase One** for the input $(I, LP, S, q, \tilde{P}, \tilde{Q})$.
3.8 **if** $value(S) > value(bestS)$ **then**
3.9 $bestS \leftarrow S$.
3.10 **if** $c \geqslant \frac{type}{5} \times X$ **then**
3.11 $type \leftarrow type + 1$.
3.12 $cr \leftarrow c \mod 7$.
3.13 **return** $bestS$.

4.3.4 Repacking items

Algorithm TWO PHASE uses the variable $type$ to indicate how to sort items of the current solution S before it applies a routine to repack them. Note that only the layout may change, contrary to the solution profit that remains unchanged. Clearly, this new layout opens possibilities to arrange another items out of S. There are five orders under consideration, namely:

- type = 1: sort into non-increasing order of area. Ties are broken placing items with largest length first;
- type = 2: sort into non-increasing order of profit. Ties are broken sorting into non-increasing order of area;
- type = 3: sort into non-increasing order of profit/area. Ties are broken placing items of largest profit first;
- type = 4: choose the order of each item at random;
- type = 5: the order is the same as in the input instance.

Algorithm 4 tries to arrange each item, following the $type$ sorting. An item is arranged into the bin searching for the left-bottom most feasible point on the grid $\tilde{P} \times \tilde{Q}$. This search starts at the length direction, that is, from coordinates $\{0 \rightarrow L\}_{\tilde{P}}$ and, continues level-by-level, that is, $\{0 \rightarrow C\}_{\tilde{Q}}$. If exists an item that cannot be arranged, then the initial solution S is returned. Otherwise, a solution with a possibly new layout is returned. Observe that the memory list is not considered here.

Algorithm 4: Repack

Input : Instance $I = (n, B, V, G)$ of the 2D-DCKP; $type$ to sort the items; solution S; sets $\tilde{P}$ and $\tilde{Q}$ of reduced raster points.
Output: Solution with possibly a new layout.

4.1 Copy all items from S to S'.
4.2 Sort items in S' accordingly to $type$.
4.3 **foreach** *item* $i \in S'$ *following the sorting order* **do**
4.4 Pack i at the first point p in $\tilde{P} \times \tilde{Q}$. If there is no such a point, **return** S.
4.5 **return** S'.

4.4 Integer Formulations for the 2D-DCKP

In this section, we present integer programming formulations for the 2D-DCKP. We start with a formulation that considers the packing position of items in a discretized bin. Then, we present another formulation that now does not define the packing position of the items, but instead uses feasibility tests in order to consider only feasible solutions. For this, we solve a problem relaxed from the 2OPP, so called contiguous bin packing problem, and when necessary a constraint programming algorithm.

4.4.1 First model

We consider next an integer formulation for the 2D-DCKP supported by the discretization points, in order to guarantee that the optimal solution can be found. The formulation uses y_i to represent whether item i is in/out the bin, while x_{ip} indicates whether item i is arranged at point p of the grid.

Therefore, let $\mathcal{P} = P \times Q$ be the set of all points of the grid in which the set $\mathcal{P}_i$ denotes the points where item i can be packed. Moreover, we denote by $\mathcal{D}_{ip}$ the set of points q, so item i packed at q covers point p. This set does not contain the points q in which i packed at result in points p on the right and on the top border of i. The integer formulation is as follows.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} v_i \, y_i, \\
& \\
\textit{subject to}: & \\
(i) \quad & y_i + y_j \leqslant 1, \qquad \forall \, \{i,j\} \in E, \\
& \\
(ii) \quad & \sum_{i=1}^{n} \sum_{q \in \mathcal{D}_{ip}} x_{iq} \leqslant 1, \qquad \forall \, p \in \mathcal{P}, \\
& \\
(iii) \quad & \sum_{p \in \mathcal{P}_i} x_{ip} = y_i, \qquad i = 1, \ldots, n, \\
& \\
(iv) \quad & y_i, \, x_{ip} \in \{0,1\}, \qquad i = 1, \ldots, n; \; \forall \, p \in \mathcal{P}_i.
\end{aligned}
\tag{4.4}
$$

In formulation (4.4), the objective function aims to maximize the total value within the bin, and constraints: (i) ensure that disjunctive constraints must be respected; (ii) ensure that no overlapping between pairs of items occur, that is, each point p of the grid has to be covered at most once by some item; (iii) connect variables x_{ip}, for all $p \in \mathcal{P}_i$, with y_i, for each item i. Finally, constraints (iv) ensure that all variables are binary.

The integer formulation (4.4) has pseudo-polynomial size in the worst case, since the number of points depends of the bin and items' dimensions. Some preliminaries tests show that its linear relaxation solution has several variables with a fractional value requiring much CPU time to close the optimality gap, especially due to constraints (4.4.ii) that are hard to solve.

4.4.2 Location-allocation based model

We now consider a relaxed version of formulation (4.4), where we do not prespecify the location of the items within the bin, but we consider only subsets of items that can, in fact, be packed.

The resulting formulation is given in eq. (4.5).

$$\max \quad \sum_{i=1}^{n} v_i \, y_i,$$

$$\begin{aligned} &subject\ to: \\ &(i) \quad y_i + y_j \leqslant 1, \qquad \forall \{i,j\} \in E, \\ &(ii) \quad \sum_{i \in H} y_i \leqslant |H| - 1, \qquad H \in \mathcal{H}, \\ &(iii) \quad \sum_{i=1}^{n} a_i \, y_i \leqslant A_{bin}, \\ &(iv) \quad y_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{4.5}$$

where $\mathcal{H} = \{H$: set of items that cannot be all packed together in the bin$\}$. Observe that constraints (4.5.ii) impose a two-dimensional orthogonal packing feasibility test for items in H, namely the 2OPP. Constraint (4.5.iii) avoids set of items for which the sum of area is larger than the bin's area. We call formulation (4.5) without constraints (4.5.ii) as R2F. Therefore, a feasibility test has to be executed whenever (an integer) solution of R2F is reached to verify violated constraints of type (4.5.ii).

Feasibility tests

When solving the R2F, and it reaches an integer solution, we have to verify if the solution is indeed feasible. Let H be the set of items in this candidate solution from R2F, so we need to find a packing of H in B or prove that such packing does not exist. To this purpose we use the constraint programming algorithm presented in Clautiaux et al. [16] reducing the domain of variables with discretization points [54].

Clautiaux et al. [16] considered, for each item $i \in H$, variables X_i and Y_i denoting a feasible point $p = (X_i, Y_i)$ where item i can be packed. Their domains are given by $X_i \in [0, \ldots, L - l_i]$ and $Y_i \in [0, \ldots, C - c_i]$, for each i. With the assumption that all input data are integers, it follows now that $X_i \in \{0, 1, \ldots, L - l_i\}$ and $Y_i \in \{0, 1, \ldots, C - c_i\}$. A *basic constraint programming model* requires, for each pair $\{i, j\}$ of items in H, only non-overlapping constraints:

$$[X_i + l_i \leqslant X_j] \; or \; [X_j + l_j \leqslant X_i] \; or \; [Y_i + c_i \leqslant Y_j] \; or \; [Y_j + c_j \leqslant Y_i]. \tag{4.6}$$

Although the basic model allows to solve the problem, it is very inefficient in practice. Clautiaux et al. [16] improved this model considering a non-preemptive continuous-scheduling problem in which $A^L = \{A_1^L, A_2^L, \ldots, A_n^L\}$ is a set of activities (all items in H) related with dimension L of the bin. Each activity A_i^L has duration l_i, consumes c_i of resource capacity C,

and can occur in interval $[s_i^L,\ e_i^L) = [0, L - l_i]$, where s_i^L and e_i^L represent, respectively, the earliest start time and the latest end time for such activity. A solution for this problem requires the start time $start_i^L$ of each activity A_i^L while satisfying resource constraints. In other words, at any point time $t \in [0, \ldots, L]$, the sum of resource consumed by activities with $start_i^L \leqslant t < start_i^L + l_i$ has to be less than or equal to capacity C. The problem is non-preemptive, that is, each activity must not be interrupted during its execution, and continuous, that is, the same resource (in this case C) has to supply all activities at time t. Similarly, it is defined a second non-preemptive continuous-scheduling problem for the set $A^C = \{A_1^C, A_2^C, \ldots, A_n^C\}$ related with dimension C. It is easy to see that these two scheduling problems can be connected to the basic constraint programming model using constraints $[start_i^L = X_i]$ and $[start_i^C = Y_i]$, for each item i, allowing to solve the 2OPP.

In the algorithm, the branching in each node u occurs at non-scheduled (non-fixed) variables. First, among all non-fixed variables $start_i^L$, the one with minimum earliest start time is chosen. Two descendant nodes are created: u_1 in which $start_{i_{min}}^L$ is fixed at its lower bound; and, u_2 where the lower bound for the $start_{i_{min}}^L$ domain is increased. Lower bounds based on dual-feasible functions, energetic reasoning concepts of cumulative-scheduling problems, and solutions of subset-sum problems are used to strengthen this algorithm.

As previously commented, we do not need to consider all integer positions to pack items, but, without loss of generality, only those ones over the discretization points. Let P_i^H (resp. Q_i^H) be the set of discretization points in the dimension L (resp. C) as defined in section 4, but as non-negative binary combination only of those items in $H \backslash \{i\}$. Then, the domain of variables X_i and Y_i, consequently, $start_i^L$ and $start_i^C$, for each item i, is assumed to be the respectively the sets of discretization points P_i^H and Q_i^H. We call this algorithm of CP2 and its runtime time is limited by a given time limit.

Contiguous bin packing as a feasibility test

In order to prevent unnecessary calls of the CP2 algorithm, a feasibility test is performed for each candidate solution in the 2OPP. This test consists of solving a integer linear formulation based on the contiguous bin packing problem [19], which we call CBP. It receives the set of items H and the bin B, and returns $true$ if the instance is feasible, and $false$ otherwise. If it returns $false$, then the respective instance of 2OPP is indeed infeasible, and then the cut is added to the R2F. On the other hand, if it returns $true$ we have to call the CP2 to find the packing or definitively proves its infeasibility.

Let $P^H = \bigcup_{i \in H} P_i^H$ be the set of all discretization points on dimension L for all items in H, and $P_i^H(t)$ be the set of all discretization points t', such that if item i is packed at t', then it covers t. Formally $P_i^H(t) = \{t' \in P_i^H : [t - l_i + 1]^+ \leqslant t' \leqslant t$. The formulation of the CBP is

as follow:

$$\begin{aligned}
&(i) \quad \sum_{i \in H} \sum_{t' \in P_i^H(t)} l_i \, y_{it'} \leqslant L, \qquad t \in P^H, \\
&(ii) \quad \sum_{t' \in P_i^H} y_{it'} = 1, \qquad i \in H, \\
&(iv) \quad y_{it} \in \{0,1\}, \qquad i \in H, \, t \in P^H.
\end{aligned} \tag{4.7}$$

For each item i and discretization point t, y_{it} is a decision variable that is 1 if item i is packed at coordinate t. As it is a feasibility test, the integer program has no objective function. Constraints (4.7.i) prevent that at any given coordinate t the sum of items that covers that coordinate is less than L. Constraints (4.7.ii) assure that all items in H are packed.

4.4.3 Bounds and valid cuts

Valid upper bounds for the 2D-DCKP can be obtained solving the 1D-DCKP considering the items area. This corresponds to search for the most valuable clique in $\bar{G}$ that respects bin's area.

Lower and upper bounds Let $\tilde{S}$ be a solution returned by $cliquer(\bar{G}, A_{bin})$, where $\tilde{S}$ is the set of items of maximum value in $\bar{G}$ that respects the bin's area. Next, we apply CP2 with $\tilde{S}$ as input to verify if it is feasibly. If yes, then we stop, since the optimal solution was reached. Otherwise, note that $\sum_{j \in \tilde{S}} v_j$ is a valid upper bound for the maximum value $\sum_{i \in V} v_i y_i$ that can be packed in any optimal solution.

When exploring a node u of the branch-and-bound tree, we can add some valid cuts over the tree. First, consider the solution sol_u of the linear relaxation at node u. For variables fixed at zero or one in sol_u, let V'_u be the set of all items fixed at one, while $\tilde{V}_u$ has those fixed at zero. We only consider variables as fixed when the lower and upper bounds are equal, that is, their gap on bounds are closed.

Valid cuts from node u With the sets V'_u and $\tilde{V}_u$, let $\bar{V}$ be the set of items in $V \backslash (V'_u \cup \tilde{V}_u)$ without conflict with any item in V'_u and let $\bar{A} = A_{bin} - \sum_{j \in V'_u} a_j$ be the remaining area of the bin considering that items in V'_u are packed within the bin.

Given the conflict graph $G_{\bar{V}}$ of all items in $\bar{V}$, let $\bar{G}_{\bar{V}}$ be its complement. For $\tilde{S}$ a solution of $cliquer(\bar{G}_{\bar{V}}, \, \bar{A})$, apply CP2 in $V'_u \cup \tilde{S}$ to verify its feasibility. If it is feasibly, a valid cut is $\sum_{j \in V} v_j y_j \geqslant \sum_{j \in \{V'_u \cup \tilde{S}\}} v_j$. Otherwise, if it is proven to be infeasible, then all items in $V'_u \cup \tilde{S}$ cannot be packed together, that is $\sum_{j \in \{V'_u \cup \tilde{S}\}} y_j \leqslant |V'_u \cup \tilde{S}| - 1$. Note that if $\sum_{j \in \{V'_u \cup \tilde{S}\}} v_j$ is less than the value of the best solution ever computed, then node u can be pruned.

It is worth to mention that for several nodes in the branch-and-bound tree, the sets V'_u and $\tilde{V}_u$ may be repeated, so we use a hash list to avoid unnecessary computation. Also, when $\bar{G}_{\bar{V}}$ has one vertex or no edges the verification occurs directly, without calling $cliquer$.

4.5 Complete Shipment of Items

We now consider a extension of the 2D-DCKP, where the set of items V is partitioned in sets $V_1, V_2, \ldots, V_{\mathcal{K}}$ and we are constrained to, for every $1 \leqslant i \leqslant \mathcal{K}$, either pack all items or no items of V_i, that is, if an item of V_i is packed for some $1 \leqslant i \leqslant \mathcal{K}$, then every item of V_i has to be packed too. This constraint is called *Complete Shipment of Items* in the literature and, thus, we will call set V_i as a *complete shipment set*.

Without loss of generality, we consider that there is no conflict between two items inside the same V_i as, otherwise, we cannot pack V_i and thus V_i can be removed from the instance. Also, notice that if item $s \in V_i$ and item $t \in V_j$ have conflict, then no item of V_i can be packed with an item of V_j. Thus, we consider that the vertex set of the conflict graph G is $\{V_1, \ldots, V_{\mathcal{K}}\}$, that is, conflicts happen for complete shipment sets instead of individual items.

In what follows, we discuss briefly how to adapt our approaches in order to consider the complete shipment constraint. All of these adaptation follows almost directly from the fact that we are considering complete shipment of items and that conflicts happen between sets of items.

Lifting item sizes In order to adapt the procedure of lifting item sizes (as in Section 4.2.2), we have to consider a slightly different integer programming formulation. For an item i, let $css(i)$ be the complete shipment set V_j such that $i \in V_j$, and for a complete shipment set V_j, let A_j be the sum of the areas of the items in V_j. Also, for an item i, let S_i be the set of non-conflicting items regarding i different than i, that is, $S_i = \{j \in V : (css(i), css(j)) \notin E\} \backslash \{i\}$. We can lift item sizes using the following formulation:

$$
\begin{aligned}
L_i^* = \quad & \max \sum_{k \in V} l_k \tau_k, \\
& \\
\textit{subject to}: & \\
(i) \quad & \sum_{k \in V} l_k \tau_k \leqslant L, \\
& \\
(ii) \quad & \tau_k \leqslant y_{css(k)}, \qquad \forall\, k \in V, \\
& \\
(iii) \quad & y_j + y_k \leqslant 1, \qquad \forall\ \{j,k\}\ \in E, \\
& \\
(iv) \quad & \sum_{j=1}^{\mathcal{K}} A_j y_j \leqslant A_{bin}, \\
& \\
(v) \quad & \tau_i = 1, \\
& \\
(vi) \quad & \tau_j \in \{0,1\}, \qquad \forall\, j \in V \backslash \{i\}, \\
& \\
(vii) \quad & y_j \in \{0,1\}, \qquad \forall\, j \in \{1, \ldots, \mathcal{K}\}.
\end{aligned}
\tag{4.8}
$$

Adapting the heuristic In order to adapt the heuristic GR to consider complete shipment, we only consider that, whenever the algorithm decides to remove an item from the packing, then we remove the whole complete shipment set of that item and, whenever the algorithm decides to add an item to a packing, we add the whole complete shipment set of that item.

Adapting the formulations It is easy to adapt the formulations from Section 4.4 to consider complete shipment, since the conflicts happens for sets of items. Instead of considering a variable y_i for every item i, we consider a variable y_i for every complete shipment set i, for $i = 1, 2, \ldots, \mathcal{K}$. The modifications on formulation 4.5 can be seen below. The integer formulation 4.4 can be adapted in the same way.

$$\max \quad \sum_{i=1}^{\mathcal{K}} y_i \sum_{j \in V_i} v_j,$$

$$\begin{aligned} &subject\ to: \\ &(i) \quad y_i + y_j \leqslant 1, \qquad \forall \{i,j\} \in E, \\ &(ii) \quad \sum_{i \in H} y_i \leqslant |H| - 1, \qquad H \in \mathcal{H}, \\ &(iii \quad \sum_{i=1}^{\mathcal{K}} A_i\, y_i \leqslant A_{bin}, \\ &(iv) \quad y_i \in \{0,1\}, \qquad i = 1, \ldots, \mathcal{K}. \end{aligned} \tag{4.9}$$

Adapting bounds and valid cuts It follows directly from the observation above that we now pack complete shipment sets instead of items and that conflicts happens between complete shipment sets.

4.6 Computational Experiments

The algorithms were implemented in the C/C++ programming language and all computational tests were performed in a computer with Intel® Core^{TM} i7-2600 3.4 GHz processor with 8 GB of RAM memory and *Linux* operating system. We used the standard framework provided by the ILOG CPLEX 12.5 Callable Library to solve the integer formulations. The algorithm CP2 used the framework provided by the ILOG CP optimizer 12.5 and ILOG CP 1.7. The total CPU time was limited to 3600 seconds for each instance when using CPLEX, in which each call to CP2 was limited to 1000 seconds.

The parameters adopted for the heuristic were obtained after some calibration tests. In such tests we looked for parameters that allowed to obtain good quality solutions. Then, the resulting

parameters were the following: $q = 30\%n$; $MAX = 5000$; and, $X = 1000$. We also imposed a time limit of 3600 seconds for the heuristic when solving each instance.

The computational tests were carried out using seven sets of instances, totaling 54 ones, defined in the literature of two-dimensional cutting and packing problems. They are:

- *cgcut*: 3 instances from Christofides and Whitlock [14];
- *gcut*: 13 instances from Beasley [5];
- *m*: 5 instances from Morábito et al. [74];
- *mw*: 5 instances from Hifi [46];
- *ngcut*: 12 instances from Beasley [6];
- *okp*: 5 instances from Fekete et al. [36];
- *uw*: 11 instances from Fayard et al. [35];

For each of these instances, we consider only one copy of each item (that is, we discard the multiplicity of each item if it exists) in order to generate three classes of conflict graphs. Each class is associated with one of the following densities for the conflict graph, $\{10\%, 17\%, 25\%\}$, where the number of conflicts in the graph corresponds to a percentage, which is the given density, of $\frac{n(n-1)}{2}$, with n as the number of items in the instance. The values assumed for the density are in accordance with the literature on the knapsack problem with conflict graph [49, 99]. The pairs of conflicting items were chosen at random.

All instances are available upon request to the authors and some of their details are given in Table 4.1. This table has the following columns: instance name; number of items; size of the bin; and, for each density there are: number of items that have the length and width updated, and number of conflicting items in the instance.

Observing Table 4.1, we note that the percentage of items with sizes updated increases accordingly to the number of conflicts grows. Then, if there are many conflicts, then there are few items that can be arranged together (into the bin), so more other items can update their sizes or even their sizes can increase significantly. For the instances under consideration, when the density is of 10%, there are 38% of the items with length updated (and 35.05% ones with width updated). If the number of conflicts in the graph increases, for example, with the density of 25%, the percentage of items with length updated grows to 42.79% (and to 44.47% for the width), so representing an elevation of 12.36% (and 26.87%) of items with length (and width) improved.

4.6.1 Results

As there is no other work about the 2D-DCKP, we consider three different approaches to solve it that are based on the resolution of the integer formulations discussed in Section 4.4. The approaches are called as F1, F2 and F3, and all ones use the grid discretized over the discretization points including the CP2 algorithm. Therefore:

Tabela 4.1: Information of the instances under consideration.

Instance	n	(L, C)	Density of 10%		Density of 17%		Density of 25%	
			#items updated	#conflicts	#items updated	#conflicts	#items updated	#Conflicts
cgcut1	7	(15, 10)	3 | 1	2	2 | 0	3	1 | 0	5
cgcut2	10	(40, 70)	3 | 3	4	2 | 2	7	2 | 3	11
cgcut3	20	(40, 70)	6 | 0	19	9 | 3	32	8 | 3	47
ngcut1	5	(10, 10)	0 | 2	1	2 | 2	1	1 | 2	2
ngcut2	7	(10, 10)	0 | 1	2	0 | 2	3	1 | 3	5
ngcut3	10	(10, 10)	1 | 0	4	1 | 2	7	2 | 1	11
ngcut4	5	(15, 10)	0 | 1	1	0 | 2	1	0 | 2	2
ngcut5	7	(15, 10)	5 | 1	2	3 | 1	3	5 | 2	5
ngcut6	10	(15, 10)	0 | 0	4	0 | 0	7	1 | 1	11
ngcut7	5	(20, 20)	1 | 2	1	1 | 2	1	1 | 2	2
ngcut8	7	(20, 20)	1 | 1	2	1 | 0	3	1 | 3	5
ngcut9	10	(20, 20)	2 | 0	4	2 | 0	7	4 | 0	11
ngcut10	5	(30, 30)	3 | 2	1	3 | 1	1	3 | 2	2
ngcut11	7	(30, 30)	4 | 1	2	4 | 1	3	4 | 4	5
ngcut12	10	(30, 30)	0 | 3	4	0 | 3	7	2 | 6	11
gcut1	10	(250, 250)	3 | 5	4	4 | 6	7	5 | 5	11
gcut2	20	(250, 250)	15 | 7	19	13 | 9	32	14 | 9	47
gcut3	30	(250, 250)	12 | 12	43	15 | 14	73	16 | 18	108
gcut4	50	(250, 250)	14 | 17	122	18 | 18	208	23 | 18	306
gcut5	10	(500, 500)	8 | 4	4	8 | 4	7	7 | 6	11
gcut6	20	(500, 500)	16 | 11	19	16 | 13	32	15 | 12	47
gcut7	30	(500, 500)	22 | 17	43	21 | 17	73	20 | 16	108
gcut8	50	(500, 500)	26 | 19	122	25 | 18	208	27 | 19	306
gcut9	10	(1000, 1000)	6 | 6	4	6 | 6	7	7 | 6	11
gcut10	20	(1000, 1000)	14 | 12	19	13 | 12	32	14 | 12	47
gcut11	30	(1000, 1000)	24 | 16	43	24 | 15	73	23 | 15	108
gcut12	50	(1000, 1000)	31 | 33	122	30 | 34	208	31 | 36	306
gcut13	32	(3000, 3000)	0 | 0	49	0 | 2	84	6 | 4	124
m1-1	10	(100, 156)	3 | 3	4	4 | 4	7	3 | 5	11
m2-1	10	(253, 294)	3 | 6	4	2 | 6	7	5 | 7	11
m3-1	10	(318, 473)	5 | 4	4	6 | 6	7	6 | 4	11
m4-1	10	(501, 556)	7 | 4	4	6 | 5	7	6 | 6	11
m5-1	10	(750, 806)	6 | 4	4	6 | 5	7	6 | 6	11
mw1	10	(100, 156)	3 | 3	4	4 | 4	7	3 | 5	11
mw2	10	(253, 294)	3 | 6	4	2 | 6	7	5 | 7	11
mw3	10	(318, 473)	5 | 4	4	6 | 6	7	6 | 4	11
mw4	10	(501, 556)	6 | 4	4	6 | 5	7	6 | 6	11
mw5	10	(750, 806)	6 | 4	4	6 | 5	7	6 | 6	11
okp1	15	(100, 100)	1 | 5	10	1 | 4	17	5 | 6	26
okp2	30	(100, 100)	0 | 5	43	0 | 8	73	0 | 6	108
okp3	30	(100, 100)	0 | 10	43	0 | 11	73	0 | 14	108
okp4	33	(100, 100)	0 | 5	52	0 | 5	89	0 | 9	132
okp5	29	(100, 100)	0 | 5	40	0 | 5	69	1 | 6	101
uw1	25	(500, 500)	12 | 15	30	12 | 14	51	13 | 14	75
uw2	35	(560, 750)	10 | 20	59	13 | 22	101	14 | 19	148
uw3	35	(700, 650)	18 | 16	59	19 | 18	101	18 | 17	148
uw4	45	(1245, 1015)	16 | 11	99	17 | 14	168	21 | 13	247
uw5	35	(1100, 1450)	21 | 22	59	18 | 21	101	17 | 22	148
uw6	47	(1750, 1542)	21 | 24	108	24 | 25	183	28 | 29	270
uw7	50	(2250, 1875)	26 | 22	122	23 | 29	208	27 | 27	306
uw8	55	(2645, 2763)	28 | 23	148	27 | 26	371	32 | 33	252
uw9	45	(3000, 3250)	30 | 25	99	29 | 29	168	29 | 30	247
uw10	60	(3500, 3650)	25 | 31	177	29 | 35	300	35 | 33	442
uw11	30	(555, 632)	3 | 1	43	3 | 3	73	5 | 6	108
#Impr.	-	-	38% | 35.05%	-	38.48% | 39.01%	-	42.79% | 44.37%	-

- F1 represents formulation (4.4) purely. F1 does not consider CBP, CP2, and neither the bounds and valid cuts;
- F2 considers formulation R2F with CP2, bounds and valid cuts (but without CBP);
- F3 considers formulation R2F with CBP called before each call to CP2, and with the bounds and valid cuts. In fact, F3 has all the improvements considered in this paper.

Approaches F1, F2 and F3 are also compared with the heuristic GR. For that, we apply the heuristic ten times (for ten different seeds) on each instance and then the best and the worst solution found are presented. For the best solution, we present the iteration where it was obtained and the CPU time spent in seconds to obtain it. The results below show that the heuristic is quite competitive in terms of computing the optimal solution, although approach F3 is by far the best one for solving the instances to optimality.

We present in Tables 4.2, 4.3 and 4.4 the results for the instances with density equal to 10%, 17% and 25%, respectively. Each row of these tables has: name of the instance; output of F1: value of the solution and total CPU time (in seconds); outputs of F2 and F3: value of the solution, CPU time (in seconds) of all calls to CP2, and total CPU time (in seconds) required to solve the instance, which also includes the time of CP2; output of GR: value of the best solution over the ten executions, iteration (of MAX) where the best solution was computed, average CPU time (in seconds) for the ten runs, and the value of the worst solution observing the ten runs. If a solution is proven to be optimal, then an asterisk appears with the respective value. In the case of the heuristic, we mark with "+".

Table 4.2 considers the case with conflict graph's density equal to 10%. All the instances were solved to optimality with approaches F2 and F3, except for $gcut13$, for which the time limit imposed was reached. In terms of solution quality, F3 has the best performance (note that the solution of $gcut13$ computed with F3 is better than those of F1, F2 and GR). With this in mind, the difference in percentage (namely, GAP) comparing with F3 was, respectively, on average: of 0.89%, for F1; of 0.13%, for F2; and, of 0.74%, for the heuristic.

For the total CPU time in Table 4.2, F3 had the best result too. On average, its time was of 120.71 seconds against 146.41 seconds for F2, 1353.28 seconds for F1, and 1816.75 seconds for GR. On the other hand, the strategy of calling CBP prior to CP2 shows clearly that CP2 can be time consuming, since the time of CP2 for F2 was of 93.12 seconds, on average, and it was reduced to 69.12 seconds for F3. We can observe that the CPU time of GR was the worse even if comparing with F1. However, this is justified by the large number of iterations that we considered for the heuristic. Note that GR obtained its best solution very quickly, around the ten first iterations, for 39 out of 54 instances, namely for 72% of the instances. Moreover, for GR, the worst solution computed over the ten executions coincided with the best solution value for 39 instances and the overall GR's GAP is lower than that of F1 (0.74% against 0.89% of F1).

In therms of non-optimal solutions and comparing with F3 in Table 4.2, we point out that: F2 did not solve only $gcut13$ that had a GAP of 0.13%; F1 did not solve 9 out of 54 instances, where the worst GAP was of 18.94% for $gcut13$; and, GR did not solve 12 instances, in which

the worst GAP was of 8.56% for $gcut13$. Observe that the worst GAP returned with GR is even better than the worst one for F1, although F1 solved more instances to optimality than GR.

Tabela 4.2: Comparing F1, F2, F3, and GR for the density equal to 10%.

Instance	F1		F2			F3			GR heuristic			
	value	time	value	t. CP2	time	value	t. CP2	time	b. value	b. iter.	avg. time	w. value
cgut1	139*	0.02	139*	0.00	0.00	139*	0.00	0.00	139+	1	80.25	139+
cgut2	1804*	1.14	1804*	0.00	0.00	1804*	0.00	0.00	1804+	1	677.75	1804+
cgut3	1620*	10.81	1620*	0.00	1.00	1620*	0.01	1.26	1620+	2	1562.54	1620+
ngcut1	141*	0.00	141*	0.00	0.00	141*	0.00	0.00	141+	1	61.82	141+
ngcut2	138*	0.04	138*	0.00	0.00	138*	0.00	0.17	138+	1	208.20	138+
ngcut3	194*	0.06	194*	0.00	0.02	194*	0.01	0.21	194+	1	205.38	194+
ngcut4	193*	0.00	193*	0.00	0.00	193*	0.00	0.00	193+	1	53.52	193+
ngcut5	227*	0.00	227*	0.00	0.00	227*	0.00	0.00	227+	1	60.21	227+
ngcut6	238*	0.26	238*	0.01	0.01	238*	0.00	0.24	238+	12	328.22	238+
ngcut7	370*	0.00	370*	0.00	0.00	370*	0.00	0.00	370+	1	120.30	370+
ngcut8	635*	0.00	635*	0.00	0.00	635*	0.00	0.00	635+	1	194.63	635+
ngcut9	804*	0.26	804*	0.00	0.00	804*	0.00	0.13	804+	1	537.42	804+
ngcut10	763*	0.00	763*	0.00	0.00	763*	0.00	0.00	763+	1	47.70	763+
ngcut11	1114*	0.06	1114*	0.00	0.00	1114*	0.00	0.14	1114+	1	202.77	1114+
ngcut12	1302*	0.34	1302*	0.00	0.00	1302*	0.00	0.13	1302+	2	862.44	1302+
gcut1	48368*	0.15	48368*	0.00	0.01	48368*	0.00	0.10	48368+	1	112.41	48368+
gcut2	59563*	53.79	59563*	0.02	0.10	59563*	0.00	1.11	59563+	1	1410.70	59563+
gcut3	60080*	82.47	60080*	0.45	25.73	60080*	0.00	28.71	60080+	46	3600.00	60080+
gcut4	61380	3600.00	61380*	25.51	1957.07	61380*	0.00	1221.21	61380+	359	3600.00	61380+
gcut5	195582*	0.75	195582*	0.00	0.01	195582*	0.00	0.23	195582+	1	223.37	195582+
gcut6	236305*	1.54	236305*	0.02	0.13	236305*	0.00	0.78	236305+	1	999.40	236305+
gcut7	236483*	10.54	236483*	0.29	1.82	236483*	0.00	7.06	236483+	1	2256.54	236483+
gcut8	240772	3600.00	245758*	8.14	651.03	245758*	0.00	551.00	245758+	221	3600.00	243466+
gcut9	924503*	0.12	924503*	0.01	0.01	924503*	0.00	0.03	924503+	1	214.68	924503+
gcut10	937349*	7.32	937349*	0.01	0.07	937349*	0.00	0.52	937349+	1	778.47	937349+
gcut11	951831*	35.30	951831*	0.28	4.85	951831*	0.00	11.88	951831+	15	3600.00	951831+
gcut12	976877	3600.00	976877*	1.82	78.07	976877*	0.00	46.50	976877+	531	3600.00	976877+
gcut13	6511040	3600.00	7464396	3600.00	3600.00	8032214	3600.00	3600.00	7344924	1	3600.00	6511040
m1	13899*	14.18	13899*	0.00	0.01	13899*	0.00	0.27	13899+	1	566.25	13899+
m2	62169*	174.92	62169*	0.02	0.02	62169*	0.00	0.89	62169+	2	1042.15	62169+
m3	133085*	22.70	133085*	0.01	0.01	133085*	0.35	0.44	133085+	1	783.68	133085+
m4	233361*	2.25	233361*	0.00	0.01	233361*	0.00	0.39	233361+	1	366.53	233361+
m5	538994*	154.18	538994*	0.00	0.01	538994*	0.00	0.35	538994+	1	751.81	538994+
mw1	3240*	14.33	3240*	0.00	0.00	3240*	0.00	0.31	3240+	1	654.10	3240+
mw2	15433*	31.49	15433*	0.02	0.02	15433*	0.00	1.05	15433+	44	1187.93	15433+
mw3	30359*	22.22	30359*	0.01	0.01	30359*	0.00	0.50	30359+	1	835.73	30359+
mw4	48732*	153.34	48732*	0.00	0.00	48732*	0.00	0.67	48732+	3	704.76	48732+
mw5	143961*	282.65	143961*	0.00	0.00	143961*	0.00	0.34	143961+	1	813.09	143961+
okp1	22048	3600.00	22048*	0.02	0.02	22048*	0.00	2.72	22048+	1	3600.00	22048+
okp2	20892	3600.00	20892*	0.00	0.00	20892*	0.00	17.59	19710	51	3600.00	19019
okp3	23196	3600.00	23196*	2.45	25.87	23196*	0.00	15.12	22801	57	3600.00	22203
okp4	24771	3600.00	25474*	1.54	63.23	25474*	0.00	41.89	25474+	6	3600.00	24771
okp5	20334	3600.00	20344*	6.67	110.62	20344*	0.66	57.98	20334	12	3600.00	19797
uw1	3973	3600.00	3973*	0.02	0.04	3973*	0.00	0.11	3973+	48	3600.00	3874
uw2	5019	3600.00	5019*	5.01	2.58	5019*	0.09	3.40	4928	57	3600.00	4753
uw3	3481	3600.00	3481*	0.02	0.13	3481*	0.00	0.21	3481+	4	3600.00	3481+
uw4	5594	3600.00	5594*	3.73	7.40	5594*	0.03	3.90	5499	2	3600.00	5202
uw5	4291	3600.00	4291*	0.62	0.64	4291*	0.02	2.41	4291+	41	3600.00	4291+
uw6	5040	3600.00	5040*	22.34	22.60	5040*	0.21	27.18	4847	52	3600.00	4593
uw7	5502	3600.00	5648*	1.82	2.29	5648*	0.01	3.98	5571	1	3600.00	5502
uw8	5024	3600.00	5600*	284.61	285.26	5600*	39.71	89.43	5215	2	3600.00	5024
uw9	4056	3600.00	4113*	5.60	6.07	4113*	0.06	5.73	4056	15	3600.00	3906
uw10	5281	3600.00	5680*	148.07	149.78	5680*	32.42	93.71	5454	1	3600.00	5281
uw11	5367	3600.00	5545*	909.21	909.23	5545*	59.21	676.82	5367	9	3600.00	5123

As reported in Table 4.3, for the density of 17%, F3 again had the best results for the solution value, time of CP2 and total CPU time.

Tabela 4.3: Comparing F1, F2, F3, and GR for the density equal to 17%.

Instance	F1		F2			F3			Heuristic			
	value	time	value	t. CP2	time	value	t. CP2	time	b. value	b. iter.	b. time	w. value
cgut1	137*	0.01	137*	0.00	0.00	137*	0.00	0.00	137+	1	147.34	137+
cgut2	1744*	0.85	1744*	0.00	0.00	1744*	0.00	0.00	1744+	1	664.96	1744+
cgut3	1500*	6.11	1500*	0.00	0.02	1500*	0.00	0.08	1500+	2	1141.15	1500+
ngcut1	118*	0.00	118*	0.00	0.00	118*	0.00	0.00	118+	1	44.89	118+
ngcut2	130*	0.02	130*	0.00	0.01	130*	0.00	0.07	130+	1	164.28	130+
ngcut3	185*	0.08	185*	0.00	0.01	185*	0.00	0.19	185+	1	202.35	185+
ngcut4	173*	0.00	173*	0.00	0.00	173*	0.00	0.00	173+	1	56.18	173+
ngcut5	251*	0.00	251*	0.00	0.00	251*	0.00	0.00	251+	1	61.32	251+
ngcut6	250*	0.14	250*	0.00	0.00	250*	0.00	0.15	250+	1	331.14	250+
ngcut7	348*	0.00	348*	0.00	0.00	348*	0.00	0.00	348+	1	89.58	348+
ngcut8	610*	0.00	610*	0.00	0.00	610*	0.00	0.00	610+	1	192.90	610+
ngcut9	712*	0.18	712*	0.01	0.01	712*	0.00	0.17	712+	1	459.36	712+
ngcut10	998*	0.00	998*	0.00	0.00	998*	0.00	0.00	998+	1	47.65	998+
ngcut11	1077*	0.02	1077*	0.00	0.00	1077*	0.00	0.00	1077+	1	196.32	1077+
ngcut12	1272*	0.60	1272*	0.00	0.00	1272*	0.00	0.00	1272+	1	726.11	1272+
gcut1	48368*	0.10	48368*	0.00	0.01	48368*	0.00	0.06	48368+	1	115.96	48368+
gcut2	58540*	38.52	58540*	0.04	0.12	58540*	0.00	1.25	58540+	1	1376.67	58540+
gcut3	59827*	73.03	59827*	0.09	1.66	59827*	0.01	17.60	59827+	1	3600.00	59827+
gcut4	60738	3600.00	60942*	4.81	185.03	60942*	0.00	138.92	60942+	47	3600.00	60942+
gcut5	192907*	0.33	192907*	0.01	0.01	192907*	0.00	0.11	192907+	1	220.68	192907+
gcut6	227555*	5.58	227555*	0.03	0.09	227555*	0.00	0.54	227555+	1	776.78	227555+
gcut7	240143*	16.45	240143*	0.16	0.71	240143*	0.00	3.49	240143+	3	2078.69	240143+
gcut8	241388	3600.00	243917*	4.00	195.91	243917*	0.00	131.94	243805	316	3600.00	242656
gcut9	835406*	0.12	835406*	0.01	0.01	835406*	0.00	0.06	835406+	1	213.57	835406+
gcut10	937349*	12.14	937349*	0.01	0.04	937349*	0.00	0.28	937349+	1	731.14	937349+
gcut11	957270*	23.63	957270*	0.10	0.98	957270*	0.00	4.06	957270+	68	2865.94	957270+
gcut12	964648*	1215.70	964648*	0.79	29.68	964648*	0.00	27.23	964648+	4	3600.00	964648+
gcut13	7712154	3600.00	7712154*	0.01	0.01	7712154*	0.00	0.00	6638044	9	3600.00	5957868
m1	13061*	3.60	13061*	0.00	0.01	13061*	0.00	0.40	13061+	3	526.74	13061+
m2	54434*	26.73	54434*	0.00	0.00	54434*	0.00	0.00	54434+	1	996.65	54434+
m3	133802*	0.37	133802*	0.00	0.00	133802*	0.00	0.00	133802+	2	513.36	133802+
m4	220120*	1.03	220120*	0.00	0.01	220120*	0.00	0.11	220120+	1	355.77	220120+
m5	508345*	21.27	508345*	0.00	0.01	508345*	0.00	0.39	508345+	2	757.88	508345+
mw1	2948*	3.58	2948*	0.00	0.00	2948*	0.00	0.20	2948+	3	493.11	2948+
mw2	13074*	21.14	13074*	0.00	0.00	13074*	0.00	0.00	13074+	1	1020.58	13074+
mw3	29899*	0.37	29899*	0.00	0.00	29899*	0.00	0.22	29899+	5	424.35	29899+
mw4	45415*	19.31	45415*	0.01	0.01	45415*	0.00	1.02	45415+	1	517.03	45415+
mw5	126990*	56.67	126990*	0.01	0.01	126990*	0.00	0.41	126990+	1	600.67	126990+
okp1	17763	3600.00	17763*	0.00	0.06	17763*	0.00	0.14	17763+	1	3600.00	17763+
okp2	20484	3600.00	20484*	1.05	17.50	20484*	0.00	22.31	19965	1	3600.00	19404
okp3	22531	3600.00	22531*	0.28	3.25	22531*	0.01	11.56	22531+	4	3600.00	22061
okp4	24695	3600.00	24695*	0.30	2.48	24695*	0.00	10.08	24099	25	3600.00	23850
okp5	19766	3600.00	19766*	0.36	1.48	19766*	0.01	4.81	19536	1	3600.00	19130
uw1	3258	3600.00	3258*	0.01	0.03	3258*	0.00	0.19	3249	1	3600.00	3249
uw2	4784	3600.00	4784*	2.11	2.18	4784*	0.02	11.92	4729	2	3600.00	4628
uw3	3222	3600.00	3222*	0.05	0.17	3222*	0.00	0.23	3222+	9	3600.00	3222+
uw4	5634	3600.00	5634*	3.97	4.09	5634*	0.03	9.12	5403	17	3600.00	5068
uw5	4217	3600.00	4217*	0.04	0.08	4217*	0.00	0.20	4217+	45	3600.00	4160
uw6	4708	3600.00	4708*	0.64	0.74	4708*	0.00	4.33	4529	93	3600.00	4418
uw7	5305	3600.00	5305*	0.23	0.30	5305*	0.00	2.17	5037	11	3600.00	4855
uw8	4750	3600.00	4750*	1.55	1.63	4750*	0.06	4.12	4699	1	3600.00	4410
uw9	4167	3600.00	4198*	7.91	8.71	4198*	0.14	22.09	4167	53	3600.00	3995
uw10	4878	3600.00	5201*	19.46	19.99	5201*	0.24	29.02	5013	12	3600.00	4695
uw11	5088	3600.00	5717	3600.00	3600.00	5717	3600.00	3600.00	5287	9	3600.00	5088

Comparing F1, F2, and GR with F3 in Table 4.3, the difference on average in terms of solution value was of 0.38%, 0.0%, and 0.88%. Note that F2 and F3 returned the same solutions for all instances, while F1 was better than the heuristic. Once again, F2 and F3 solved all the instances to optimality, except for one, see $uw11$. And, F1 could not prove optimality of 19 instances, although for some of them (14 ones) it obtained the same optimal value, but without prove optimality due to the time limit that was reached. The worst solution computed with F1 was for $uw11$ with difference of 11%, while GR had problems with 14 instances (no optimal solution for them), in which the worst difference was of 13.93% for $gcu13$.

Observing the total CPU time in Table 4.3, it was on average of 75.21 seconds for F3, of 75.51 seconds for F2, of 1295.32 seconds for F1, and of 1753.91 seconds for GR. Note that if considering F3 and F2 instead of F1, there is a substantial reduction on the CPU time and this is even better if comparing with GR. The time of CP2 was very close for both F3 (of 66.67 seconds on average) and F2 (of 67.55 seconds on average), but it still show that CBP is useful. It is important mentioning that the average CPU time and CP2 time of both F2 and F3 increased due to instance $uw11$, which reached the time limit of one hour.

Table 4.4 reports the same conclusions of those cases with 10% and 17% as density for the conflict graph. Once again F3 and F2 solved all the instances to optimality, while F1 had a difference on percentage with F3 of 0.19%, on average, and GR had a difference of 0.50%, on average, comparing with F3. We note that F1 computed for 52 out of 54 instances a solution equal to optimal one, however for 20 out of these 52 instances it did not prove optimality, since the time limit was reached. On the other hand, GR computed optimal solutions for 45 instances and both F1 and GR computed the worst worst solution for the instance $uw10$ with a difference of 5.60% compared to F3.

Observing the total CPU time of all approaches in Table 4.4, they required 5.42 seconds for F3, 8.01 seconds for F2, 1337.41 seconds for F1, and 1639.44 seconds for GR. In therms of CP2 usage, F3 required 0.44 seconds, while F2 required 8.01 seconds. All these results are average ones. Although the CPU time of GR was the worst one compared with the other approaches, it was capable of reaching optimal solutions very soon for 38 out of 54 instances (before the tenth iteration) as well as its worst solutions coincided with its best solutions over the ten executions for 41 instances.

Comparing the results in Tables 4.2 to 4.4, we observe that the solution value and total CPU time decreased accordingly the density of the conflict graph increased. On average if comparing the results for the density of 10% with that of 25%, the value of the solution decreased of 22.59%, while the CPU time decreased 95.51% both for F3, which presented the overall best results. Moreover, the size of the cliques in $\bar{G}$ became small when the number of conflicts in G increased, consequently the solutions had not so many items and check the feasibility was quite fast, as noted in the time of CP2 that was very low on average. On the other hand, to find such cliques was not an easy task, but cliquer was able to perform that very well especially after applied the strategies to improve the original instance.

Regarding the computational time required by GR, note that it arranges items over a grid of reduced raster points. Although these number of points is small compared to the discretization

Tabela 4.4: Comparing F1, F2, F3, and GR for the density equal to 25%.

Instance	F1		F2			F3			Heuristic			
	value	time	value	t. CP2	time	value	t. CP2	time	b. value	b. iter.	b. time	w. value
cgut1	109*	0.02	109*	0.00	0.00	109*	0.00	0.00	109+	1	105.55	109+
cgut2	1537*	1.21	1537*	0.00	0.00	1537*	0.00	0.00	1537+	1	669.54	1537+
cgut3	1400*	4.89	1400*	0.00	0.02	1400*	0.00	0.38	1400+	1	784.42	1400+
ngcut1	118*	0.00	118*	0.00	0.00	118*	0.00	0.00	118+	1	52.48	118+
ngcut2	148*	0.00	148*	0.00	0.00	148*	0.00	0.02	148+	1	93.77	148+
ngcut3	183*	0.02	183*	0.00	0.00	183*	0.00	0.00	183+	1	162.97	183+
ngcut4	146*	0.00	146*	0.00	0.00	146*	0.00	0.00	146+	1	51.07	146+
ngcut5	233*	0.00	233*	0.00	0.00	233*	0.00	0.00	233+	1	55.65	233+
ngcut6	225*	0.02	225*	0.00	0.00	225*	0.00	0.00	225+	1	223.62	225+
ngcut7	370*	0.00	370*	0.00	0.00	370*	0.00	0.00	370+	1	86.31	370+
ngcut8	358*	0.00	358*	0.00	0.00	358*	0.00	0.00	358+	1	131.93	358+
ngcut9	681*	0.24	681*	0.00	0.00	681*	0.00	0.03	681+	1	279.39	681+
ngcut10	763*	0.00	763*	0.00	0.00	763*	0.00	0.00	763+	1	50.39	763+
ngcut11	872*	0.01	872*	0.00	0.00	872*	0.00	0.00	872+	1	93.42	872+
ngcut12	1179*	0.12	1179*	0.00	0.00	1179*	0.00	0.06	1179+	1	399.58	1179+
gcut1	46516*	0.12	46516*	0.00	0.01	46516*	0.00	0.08	46516+	1	94.82	46516+
gcut2	58705*	42.18	58705*	0.02	0.04	58705*	0.00	0.27	58705+	5	941.97	58705+
gcut3	59390*	77.69	59390*	0.06	0.42	59390*	0.01	2.72	59390+	1	3012.25	59390+
gcut4	60388	3600.00	60388*	0.00	61.19	60388*	0.00	26.92	60388+	62	3600.00	60388+
gcut5	181982*	0.11	181982*	0.01	0.01	181982*	0.01	0.06	181982+	1	101.95	181982+
gcut6	225229*	14.13	225229*	0.00	0.07	225229*	0.00	0.51	225229+	1	613.40	225229+
gcut7	234804*	12.79	234804*	0.06	0.14	234804*	0.00	0.83	234804+	2	1600.47	234804+
gcut8	243838	3600.00	243838*	1.36	52.45	243838*	0.00	40.34	243838+	3	3600.00	243838+
gcut9	819514*	0.13	819514*	0.00	0.01	819514*	0.00	0.04	819514+	1	143.03	819514+
gcut10	899316*	9.71	899316*	0.01	0.05	899316*	0.00	0.47	899316+	1	536.10	899316+
gcut11	957270*	23.13	957270*	0.02	0.35	957270*	0.00	0.92	957270+	12	2413.63	957270+
gcut12	964648	3600.00	964648*	0.37	37.82	964648*	0.02	16.49	964648+	2	3600.00	964648+
gcut13	5064537	3600.00	5064537*	0.00	0.00	5064537*	0.00	0.00	5025828	1	3600.00	4875352
m1	13384*	1.22	13384*	0.00	0.00	13384*	0.00	0.00	13384+	1	447.96	13384+
m2	52777*	2.51	52777*	0.00	0.00	52777*	0.00	0.00	52777+	1	649.03	52777+
m3	108619*	3.40	108619*	0.00	0.00	108619*	0.00	0.00	108619+	1	330.32	108619+
m4	238934*	0.36	238934*	0.00	0.00	238934*	0.00	0.10	238934+	1	286.87	238934+
m5	520844*	5.08	520844*	0.00	0.00	520844*	0.00	0.00	520844+	1	531.40	520844+
mw1	3039*	1.23	3039*	0.00	0.00	3039*	0.00	0.00	3039+	1	433.97	3039+
mw2	12813*	3.95	12813*	0.00	0.00	12813*	0.00	0.00	12813+	2	620.85	12813+
mw3	25317*	3.28	25317*	0.00	0.00	25317*	0.00	0.00	25317+	1	288.87	25317+
mw4	46458*	7.06	46458*	0.00	0.00	46458*	0.00	0.00	46458+	1	425.51	46458+
mw5	126304*	5.16	126304*	0.00	0.00	126304*	0.00	0.00	126304+	1	451.36	126304+
okp1	16422	3600.00	16422*	0.00	0.00	16422*	0.00	0.16	16422+	1	3600.00	16422+
okp2	19545	3600.00	19545*	0.17	1.62	19545*	0.00	3.22	19545+	14	3600.00	18854
okp3	22531	3600.00	22531*	0.07	0.42	22531*	0.00	1.76	22531+	66	3600.00	22073
okp4	23264	3600.00	23264*	0.04	0.19	23264*	0.00	1.08	23150	33	3600.00	22870
okp5	18215	3600.00	18215*	0.19	0.42	18215*	0.00	2.02	18215+	3	3600.00	18215+
uw1	3360	3600.00	3360*	0.00	0.02	3360*	0.00	0.04	3360+	2	2965.92	3360+
uw2	4571	3600.00	4571*	0.00	0.02	4571*	0.00	0.00	4571+	148	3600.00	4486
uw3	3297	3600.00	3297*	0.02	0.04	3297*	0.00	0.07	3297+	29	3600.00	3297+
uw4	4522	3600.00	4522*	0.31	0.45	4522*	0.01	1.22	4348	87	3600.00	4170
uw5	3701	3600.00	3701*	0.01	0.03	3701*	0.00	0.10	3667	9	3600.00	3545
uw6	4229	3600.00	4229*	0.58	0.78	4229*	0.04	4.10	4229+	127	3600.00	4092
uw7	4989	3600.00	4989*	0.15	0.21	4989*	0.00	1.87	4945	135	3600.00	4581
uw8	4396	3600.00	4396*	0.20	0.33	4396*	0.01	1.99	4178	47	3600.00	3965
uw9	3694	3600.00	3694*	0.02	0.10	3694*	0.00	0.12	3530	52	3600.00	3453
uw10	4453	3600.00	4717*	0.61	0.78	4717*	0.03	3.27	4453	36	3600.00	4232
uw11	4768	3600.00	5015*	175.30	274.88	5015*	23.81	181.66	4768	14	3600.00	4709

points as described in the literature [79], this seems not to be a good benefit for the heuristic, since the number of points still remain large and search for a feasible point may require some relevant CPU time. Maybe the usage of a set containing only the feasible points on the contour of the packing [84], instead to observe all points in the grid lead to a better performance.

It is worth mentioning that the approaches here proposed can also be used to solve the 2KP, although it is not the focus of this paper. Specific algorithms for the 2KP can be found for example in Baldacci and Boschetti [4]. Moreover, disjunctive constraints may not allow some classes of valid inequalities and bounds of the 2KP, as those based on knapsack formulations, conservative scales [8], dual-feasible functions [18] and 1D bar relaxations [7], can be directly applied on the 2D-DCKP. And, the price of adapting such strategies may not comply with the expected quality in the final solution.

4.6.2 Results for Complete Shipment

In order to generate instances for this case, for each of the instances discussed previously, we organize at random items on groups, with one, two, or at most three items per group, due to the number of items in instances. Next, the conflicts were generated observing these groups instead of individual items.

Although we mentioned in Section 4.5 how to adapt the integer formulations, bounds, valid cuts and the heuristic for dealing with complete shipment, we do not provide the results here for all the approaches like presented before. Instead of this, we only present the results for an approach similar to F3 (the best one), namely F4, which consists of formulation (4.9) with CBP called before each call to CP2.

The results for F4 are in Table 4.5 that has the following columns: instance name; number of groups that the instance has; output of F4 for the density equal to 10%, which are: value of the solution, total CPU time (in seconds) considering all calls of CP2, and the total CPU time (in seconds) to solve the instance, for a time limit of 3600 seconds. The same outputs are considered for the density equal to 17% and 25%.

Accordingly to the results in Table 4.5, F3 solved all the instances to optimality for the density of 25%, while for the density of 10% only one instance (see $gcut13$) was not proved optimal, and two instances (see $gcut13$ and $uw11$) for the density of 17%. The total CPU time was, in seconds, of 85.57 for the density of 10%, of 133.36 for the density of 17%, and of 0.02 for density of 25%. Note that it had an expressive reduction of 99% if comparing for the density of 10% with that of 25%.

Comparing the results of F3 in Tables 4.2 to 4.4 for the 2D-DCKP without complete shipment constraint with those in Table 4.5, we noted that the solution value decreased of 11.12% and 8.82 for the densities of 10% and 17%, respectively, whereas it increased of 11.97% for the density of 25%. These evidence that the number of conflicts can grow if considering the complete shipment constraint, since the conflicts now occur between groups of items, which results on more conflicts in the graph.

Tabela 4.5: Results considering the complete shipment constraint for F4.

Instance	#groups	density of 10%			density of 17%			density of 25%		
		value	t. CP2	time	value	t. CP2	time	value	t. CP2	time
cgcut1	3	131*	0.00	0.00	131*	0.00	0.00	131*	0.00	0.00
cgcut2	4	1977*	0.00	0.00	1848*	0.00	0.00	1522*	0.00	0.00
cgcut3	7	880*	0.00	0.00	880*	0.00	0.00	880*	0.00	0.00
ngcut1	2	63*	0.00	0.00	63*	0.00	0.00	63*	0.00	0.00
ngcut2	3	158*	0.00	0.00	158*	0.00	0.00	85*	0.00	0.00
ngcut3	4	166*	0.00	0.00	154*	0.00	0.00	154*	0.00	0.00
ngcut4	2	109*	0.00	0.00	109*	0.00	0.00	109*	0.00	0.00
ngcut5	3	248*	0.00	0.00	248*	0.00	0.00	248*	0.00	0.00
ngcut6	4	255*	0.00	0.00	212*	0.00	0.00	212*	0.00	0.00
ngcut7	2	217*	0.00	0.00	217*	0.00	0.00	217*	0.00	0.00
ngcut8	3	557*	0.00	0.00	557*	0.00	0.00	557*	0.00	0.00
ngcut9	4	649*	0.00	0.00	649*	0.00	0.00	649*	0.00	0.00
ngcut10	2	719*	0.00	0.00	719*	0.00	0.00	719*	0.00	0.00
ngcut11	3	941*	0.00	0.00	941*	0.00	0.00	782*	0.00	0.00
ngcut12	4	1383*	0.00	0.00	1310*	0.00	0.00	1256*	0.00	0.00
gcut1	4	21388*	0.00	0.00	21388*	0.00	0.00	21388*	0.00	0.00
gcut2	7	39513*	0.00	0.00	39513*	0.00	0.00	39513*	0.00	0.00
gcut3	11	51158*	0.00	0.00	45662*	0.00	0.01	45662*	0.00	0.01
gcut4	17	51948*	0.01	0.09	51948*	0.00	0.03	51948*	0.00	0.01
gcut5	4	189856*	0.00	0.00	189856*	0.00	0.00	189856*	0.00	0.00
gcut6	7	173682*	0.00	0.00	173682*	0.00	0.00	173682*	0.00	0.00
gcut7	11	177596*	0.00	0.01	177596*	0.00	0.00	177596*	0.00	0.01
gcut8	17	155680*	0.00	0.12	155680*	0.00	0.06	155680*	0.00	0.03
gcut9	4	653793*	0.00	0.00	653793*	0.00	0.00	653793*	0.00	0.00
gcut10	7	659814*	0.00	0.00	659814*	0.00	0.00	659814*	0.00	0.00
gcut11	11	687962*	0.01	0.02	609516*	0.00	0.01	687962*	0.00	0.00
gcut12	17	717188*	0.01	0.03	717188*	0.00	0.03	717188*	0.00	0.02
gcut13	11	7968636	3600.00	3600.00	7951597	3600.00	3600.00	7704177*	0.00	0.01
m1	4	12261*	0.00	0.00	12071*	0.00	0.00	12071*	0.00	0.00
m2	4	63090*	0.00	0.00	63090*	0.00	0.00	63090*	0.00	0.00
m3	4	108094*	0.00	0.00	108094*	0.00	0.00	104694*	0.00	0.00
m4	4	219026*	0.00	0.00	194854*	0.00	0.00	194854*	0.00	0.00
m5	4	477248*	0.00	0.00	467707*	0.00	0.00	467707*	0.00	0.00
mw1	4	2796*	0.00	0.00	2582*	0.00	0.00	2582*	0.00	0.00
mw2	4	15010*	0.00	0.00	15010*	0.00	0.00	15010*	0.00	0.00
mw3	4	22559*	0.00	0.00	24569*	0.00	0.00	22559*	0.00	0.00
mw4	4	42117*	0.00	0.00	41602*	0.00	0.00	41602*	0.00	0.00
mw5	4	116447*	0.00	0.00	116447*	0.00	0.00	116447*	0.00	0.00
okp1	6	20611*	0.00	0.00	15578*	0.00	0.00	16289*	0.00	0.00
okp2	11	18947*	0.01	0.04	18947*	0.00	0.02	18083*	0.01	0.04
okp3	11	18989*	0.00	0.03	22017*	0.00	0.02	18989*	0.00	0.02
okp4	12	19622*	0.02	0.06	19051*	0.01	0.03	18029*	0.01	0.03
okp5	10	17083*	0.01	0.05	17027*	0.06	0.38	17083*	0.00	0.03
uw1	9	2847*	0.00	0.01	2847*	0.00	0.00	2847*	0.00	0.00
uw2	11	3612*	0.34	0.98	3612*	0.04	0.15	3612*	0.01	0.04
uw3	9	2399*	0.00	0.02	2399*	0.00	0.02	2399*	0.00	0.00
uw4	13	3879*	0.01	0.13	3879*	0.02	0.12	3879*	0.00	0.02
uw5	17	2433*	0.00	0.03	2433*	0.00	0.02	2433*	0.00	0.01
uw6	13	3373*	0.03	0.22	3373*	0.00	0.04	2956*	0.00	0.03
uw7	17	3768*	0.11	0.33	3768*	0.01	0.19	3768*	0.00	0.03
uw8	19	3016*	1.77	6.15	3016*	0.04	0.25	3016*	0.01	0.09
uw9	21	3147*	0.02	0.18	3147*	0.01	0.07	3053*	0.00	0.05
uw10	19	3665*	0.31	1.48	3277*	0.11	0.39	3277*	0.01	0.10
uw11	9	5859*	771.26	1011.26	5119	3600.00	3600.00	4424*	0.12	0.60

4.7 Conclusions

This paper investigates the 0-1 knapsack problem in its two-dimensional version and considering a conflict graph that is associated to pairs of conflicting items. A heuristic, with a greedy randomized routine, a memory list and a diversification strategy to escape from local optima, was proposed. It allows the heuristic reaches optimal solutions for 79.39% of the instances, despite of its considerable CPU time that was of 1774.85 seconds on average. Naturally, this time can be reduced by a factor of ten, since for 95.67% of all instances the best solution was reached before the 100th iteration.

Among the approaches that consider integer formulations, those based on the location-allocation model are also decisive, namely F2, F3 and F4. They allow the difficulty of the problem be scaled by solving a feasibility problem, on contrary of the first model that seems to have considerable variables with fractional values in its linear relaxation, so it (F1) could not prove optimality of 35.18% of the instances due to the limit imposed.

The usage of valid cuts over cliques in the compatibility graph allows to prune nodes in which solutions never reached the lower bound. The combination of these procedures during the resolution of the integer formulations allowed to solve 98.76%, 98.76%, and 98.14% of the instances to optimality requiring low CPU time with F2, F3, and F4, respectively. Special attention has to be given to CBP, since it helped to decrease the number of calls to CP2, consequently decreasing the time spent with CP2 during the feasibility tests.

Future work will consider the development of accurate upper bounds and other valid inequalities observing the structure of the conflict graph. Also, we expect to design a local search routine for the heuristic and new diversification strategies to escape of local optima solutions.

Capítulo 5

Conclusões e Propostas Futuras

Nesta tese apresentamos um novo algoritmo branch-and-cut para o problema do roteamento de veículos capacitados com restrições de descarregamento, o algoritmo usa conhecidas e novas rotinas de separação. Formulações em programação por restrições para o problema de empacotamento foram melhoradas e adaptadas para considerar as restrições de descarregamento. Além disso algumas novas heurísticas para o problema de empacotamento foram apresentadas. O algoritmo se mostrou competitivo, obtendo resultados melhores que a literatura na maioria dos casos.

Também apresentamos um novo algoritmo de espaço limitado para o problema online de empacotamento de círculos, com uma razão de competitividade de 2.4394. Também apresentamos um limitante inferior para qualquer algoritmo de 2.2920. Uma nova modelagem e discretização foram propostas para o problema de decisão de um único contêiner.

Por fim apresentamos um novo algoritmo branch-and-cut exato e uma nova heurística para o problema da mochila bidimensional com conflitos, e com carregamento completo. O algoritmo se mostrou muito mais eficiente que antigas formulações em instâncias adaptadas da literatura.

Muitos problemas práticos são de fato combinações de vários problemas, sendo cada um melhor resolvido com diferentes técnicas. Isso sugere que o desenvolvimento de algoritmos híbridos que fazem uso delas pode ser bastante promissor.

Não parece simples dividir qualquer problema complexo em subproblemas com diferentes características e aplicar técnicas específicas em cada parte para obter um bom resultado global. A dificuldade se dá justamente em como dividir o problema de forma a guiar cada parte em direção de boas soluções para o problema principal.

Os algoritmos propostos nesta tese podem ainda ser testados em diferentes instâncias reais ou da literatura, que evidenciem com maior precisão as vantagens e as limitações dos mesmos. Ainda é possível usar estratégias e formulações parecidas nos problemas considerados e em variantes, em particular, pode ser interessante combinar a formulação (4.5) com a (4.6) em um único modelo. Também é possível que o algoritmo apresentado no Capítulo 3 possa ser generalizado para outros problemas e formas.

Referências Bibliográficas

[1] H. Akeb, M. Hifi, and M. E. O. A. Mounir. Local branching-based algorithms for the disjunctively constrained knapsack problem. *Computers & Industrial Engineering*, 60: 811–820, 2011.

[2] B. L. P. Azevedo, P. Hokama, F. K. Miyazawa, and E. C. Xavier. A branch-and-cut approach for the vehicle routing problem with two-dimensional loading constraints. In *Proc. of the XLI Simpósio Brasileiro de Pesquisa Operacional*, page 12 pgs, Porto Seguro, BA, Brazil, September 1-4 2009.

[3] B. Baker, E. Coffman, and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[4] R. Baldacci and M. A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 183:1136–1149, 2007.

[5] J. E. Beasley. Bounds for two-dimensional cutting. *Journal of the Operationsl Research Society*, 36(1):71–74, 1985.

[6] J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36(4):297–306, 1985.

[7] G. Belov, V. M. Kartak, H. Rohling, and G. Scheithauer. One-dimensional relaxations and lp bounds for orthogonal packing. *International Transactions in Operational Research*, 16(6):745–766, 2009.

[8] G. Belov, V. M. Kartak, H. Rohling, and G. Scheithauer. Conservative scales in packing problems. *OR Spectrum*, 35(2):505–542, 2013.

[9] C. Bentz, D. Cornaz, and B. Ries. Packing and covering with linear programming: A survey. *European Journal of Operational Research*, 227(3):409–422, 2013.

[10] E. G. Birgin, R. D. Lobato, and R. Morabito. An effective recursive partitioning approach for the packing of identical rectangles in a rectangle. *Journal of the Operational Research Society*, 61(2):306–320, 2010.

[11] E. Bischoff and M. Ratcliff. Issues in the development of approaches to container loading. *Omega*, 23(4):377 – 390, 1995. ISSN 0305-0483.

[12] A. Bortfeldt and G. Wäscher. Constraints in container loading - a state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013.

[13] M. Boschetti and A. Mingozzi. A heuristic algorithm for orthogonal packing problem. part i: New lower bounds for the oriented case. *4OR*, 1(1):27–42, 2003.

[14] N. Christofides and C. Whitlock. An algorithm for two dimensional cutting problems. *Ops. Res.*, 25:30–44, 1977.

[15] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183:1196–1211, 2007.

[16] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers & Operations Research*, 35: 944–959, 2008.

[17] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers and Operations Research*, 35 (3):944 – 959, 2008.

[18] F. Clautiaux, C. Alves, and J. V. de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 179:317–342, 2010.

[19] J.-F. Côté, M. Gendrau, and . J.-Y. Potvin. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research*, 62(5): 1126–1141, 2014.

[20] J.-F. Côté, M. Gendreau, and J.-Y. Potvin. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research*, 62(5): 1126–1141, 2014.

[21] J. L. M. da Silveira and E. C. Xavier. Pickup and delivery problem with two dimensional loading/unloading constraints. In *5th International Conference in Computational Logistics, ICCL 2014*, volume 8760 of *Lecture Notes in Computer Science*, pages 31–46, 2014.

[22] J. L. M. da Silveira, F. K. Miyazawa, and E. C. Xavier. Heuristics for the strip packing problem with unloading constraints. *Computers & OR*, 40(4):991–1003, 2013.

[23] J. L. M. da Silveira, E. C. Xavier, and F. K. Miyazawa. A note on a two dimensional knapsack problem with unloading constraints. *RAIRO - Theor. Inf. and Applic.*, 47(4): 315–324, 2013.

[24] J. L. M. da Silveira, F. K. Miyazawa, and E. C. Xavier. Two-dimensional strip packing with unloading constraints. *Discrete Applied Mathematics*, 164, Part 2(0):512 – 521, 2014.

[25] E. D. Demaine, S. P. Fekete, and R. J. Lang. Circle packing for origami design is hard. In *Fifth International Meeting of Origami Science, Mathematics, and Education*, pages 609–626, 2010.

[26] U. di Bologna. Operations research group. `http://www.or.deis.unibo.it/`. Accessed: 2014-07-11.

[27] O. Dominguez, A. A. Juan, B. Barrios, J. Faulin, and A. Agustin. Using biased randomization for solving the two-dimensional loading vehicle routing problem with heterogeneous fleet. *Annals of Operations Research*, 236(2):383–404, 2016.

[28] C. Duhamel, P. Lacomme, A. Quilliot, and H. Toussaint. A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem. *Computers & Operations Research*, 38(3):617–640, 2011.

[29] M. Eley. A bottleneck assignment approach to the multiple container loading problem. *OR Spectrum*, 25(1):45–60, 2003.

[30] L. Epstein. Two-dimensional online bin packing with rotation. *Theoretical Computer Science*, 411(31-33):2899–2911, June 2010.

[31] L. Epstein and R. Van Stee. Optimal online algorithms for multidimensional packing problems. *SIAM Journal on Computing*, 35(2):431–448, 2005.

[32] L. Epstein and R. van Stee. Bounds for online bounded space hypercube packing. *Discrete Optimization*, 4(2):185–197, June 2007.

[33] L. Epstein, A. Levin, and R. V. Stee. Two-dimensional packing with conflicts. *Acta Informatica*, 45(3):155–75, 2008.

[34] R. Z. Farahani, N. Asgari, N. Heidari, M. Hosseininia, and M. Goh. Covering problems in facility location: A review. *Computers & Industrial Engineering*, 62:368–407, 2012.

[35] D. Fayard, M. Hifi, and V. Zissimopoulos. An efficient approach for large-scale two-dimensional guillotine cutting stock problems. *Journal of the Operational Research Society*, 49(12):1270–1277, 1998.

[36] S. P. Fekete, J. Schepers, and J. C. van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 2007.

[37] G. Fuellerer, K. Doernera, R. Hartla, and M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers and Operations Research*, 36 (3):655–673, 2009.

[38] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, 1979.

[39] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report, Carnegie-Mellon University, Pittsburgh, PA, 1979.

[40] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40:342–350, 2006.

[41] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1):4–18, 2008.

[42] J. F. Gonçalves and M. G. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

[43] J. F. Gonçalves and M. G. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500–510, 2013.

[44] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

[45] J. C. Herz. A recursive computational procedure for two-dimensional stock-cutting. *IBM J. Res. Dev.*, pages 462–469, 1972.

[46] M. Hifi. Exact algorithms for large-scale unconstrained two and three staged cutting problems. *Computational Optimization and Applications*, 18:63–88, 2001.

[47] M. Hifi. An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem. *Engineering Optimization*, 46:1109–1122, 2014.

[48] M. Hifi and R. Hallah. A Literature Review on Circle and Sphere Packing Problems: Models and Methodologies. *Advances in Operations Research*, 2009(4):1–22, July 2009.

[49] M. Hifi and M. Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57:718–726, 2006.

[50] M. Hifi and M. Michrafy. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & Operations Research*, 34:2657–2673, 2007.

[51] M. Hifi and N. Otmani. An algorithm for the disjunctively constrained knapsack problem. *International Journal of Operational Research*, 13:22–43, 2012.

[52] M. Hifi, S. Saleh, and L. Wu. A hybrid guided neighborhood search for the disjunctively constrained knapsack problem. *Cogent Engineering*, 2(1):1068969, 2015.

[53] P. Hokama, F. K. Miyazawa, and R. C. Schouery. A bounded space algorithm for online circle packing. *Information Processing Letters*, 116(5):337–342, 2016.

[54] P. Hokama, F. K. Miyazawa, and E. C. Xavier. A branch-and-cut approach for the vehicle routing problem with loading constraints. *Expert Systems with Applications*, 47:1–13, 2016.

[55] P. Hokama, F. K. Miyazawa, and E. C. Xavier. A branch-and-cut approach for the vehicle routing problem with loading constraints. *Expert Systems with Applications*, 47:1–13, 2016.

[56] J. N. Hooker. *Integrated Methods for Optimization (International Series in Operations Research & Management Science, Vol. 170)*. Springer, 2011. ISBN 1461418992.

[57] M. Iori and S. Martello. An annotated bibliography of combined routing and loading problems. *Yugoslav Journal of Operations Research*, 23(3), 2013.

[58] M. Iori, J. Salazar-González, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constrains. *Transportation Science*, 41(2):253–264, 2007.

[59] K. Jansen and S. Ohring. Approximation algorithms for time constrained scheduling. *Information and Computation*, 132:85–108, 1997.

[60] D. S. Johnson. *Near-optimal Bin Packing Algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[61] L. Junqueira, J. F. Oliveira, M. A. Carravilla, and R. Morabito. An optimization model for the vehicle routing problem with practical three-dimensional loading constraints. *International Transactions in Operational Research*, 20(5). ISSN 1475-3995.

[62] A. Khanafer, F. Clautiaux, and E.-G. Talbi. Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers & Operations Research*, 39:54–63, 2012.

[63] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[64] J. Li, P. M. Pardalos, H. Sun, J. Pei, and Y. Zhang. Iterated local search embedded adaptive neighborhood selection approach for the multi-depot vehicle routing problem with simultaneous deliveries and pickups. *Expert Systems with Applications*, 42(7):3551 – 3561, 2015. ISSN 0957-4174.

[65] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[66] L. A. Lysgaard J. and E. R. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2003.

[67] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[68] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.

[69] S. Martello, D. Pisinger, D. Vigo, E. D. Boef, and J. Korst. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Trans. Math. Softw.*, 33, March 2007. ISSN 0098-3500.

[70] M. Mesyagutov, G. Scheithauer, and G. Belov. {LP} bounds in various constraint programming approaches for orthogonal packing. *Computers and Operations Research*, 39 (10):2425 – 2438, 2012. ISSN 0305-0548.

[71] M. Mesyagutov, G. Scheithauer, and G. Belov. New constraint programming approaches for 3d orthogonal packing. Technical Report MATH-NM-01-2012, Technische Universität Dresden, Dresden, Germany, January 2012.

[72] M. Mesyagutov, G. Scheithauer, and G. Belov. Lp bounds in various constraint programming approaches for orthogonal packing. *Computers & Operations Research*, 39(10): 2425–2438, 2012.

[73] F. K. Miyazawa, L. L. C. Pedrosa, R. C. S. Schouery, M. Sviridenko, and Y. Wakabayashi. Polynomial-time approximation schemes for circle packing problems. In *Proceedings of the 22th Annual European Symposium on Algorithms*, pages 713–724, 2014.

[74] R. Morábito, M. Arenales, and V. F. Arcaro. An and-or-graph approach for two-dimensional cutting problems. *European J. Operational Research*, 58:263–271, 1992.

[75] V. W. Morais, G. R. Mateus, and T. F. Noronha. Iterated local search heuristics for the vehicle routing problem with cross-docking. *Expert Systems with Applications*, 41(16): 7495 – 7506, 2014. ISSN 0957-4174.

[76] D. Naddef and G. Rinaldi. Branch-and-cut algorithms for the capacitated vrp. toth p. vigo d., eds. the vehicle routing problem. *SIAM Monographs on Discrete Mathematics and Applications*, 9:53–84, 2002.

[77] S. Niskanen and P. R. J. Ostergard. Cliquer user's guide, version 1.0. Technical Report T48, Communications Laboratory, Helsinki University of Technology, Espoo, Finland, 2003.

[78] U. Pferschy and J. Schauer. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13(2):233–249, 2009.

[79] T. A. Queiroz, F. Miyazawa, Y. Wakabayashi, and E. Xavier. Algorithms for 3D guillotine cutting problems: unbounded knapsack, cutting stock and strip packing. *Computers & Operations Research*, 39:200–212, 2012.

[80] T. A. Queiroz, F. K. Miyazawa, and Y. Wakabayashi. On the l-approach for generating unconstrained two-dimensional non-guillotine cutting patterns. *4OR*, 13:199–219, 2015.

[81] G. Reinelt. *TSPLIB - a traveling salesman problem library*. Report. Inst. für Mathematik, 1990. URL `http://books.google.com.br/books?id=4ZgLHAAACAAJ`.

[82] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. ISBN 0444527265.

[83] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A. Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*, pages 10–20. Springer Berlin / Heidelberg.

[84] G. Scheithauer. Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operativa*, 27:3–34, 1997.

[85] G. Scheithauer and J. Terno. The G4-heuristic for the pallet loading problem. *Journal of the Operational Research Society*, 47:511–522, 1996.

[86] E. Specht. Packomania, Mar. 2015. URL `http://www.packomania.com`.

[87] P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: a categorized, application-oriented research bibliography. *J. Operational Research Society*, 43(7):691 – 706, 1992.

[88] P. G. Szabó, M. C. Markót, T. Csendes, E. Specht, L. G. Casado, and I. García. *New Approaches to Circle Packing in a Square. With Program Codes*, volume 6 of *Springer Optimization and Its Applications*. Springer Science & Business Media, 2007.

[89] R. F. Toso and M. G. C. Resende. brkgaAPI: A c++ application programming interface for biased random-key genetic algorithms. `http://www2.research.att.com/~mgcr/src/brkgaAPI/`. Accessed: 2014-07-11.

[90] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2002.

[91] J. D. Ullman. The Performance of a Memory Allocation Algorithm. Technical Report 100, Princeton University, Dept. of Electrical Engineering, Computer Sciences Laboratory, 1971.

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

[93] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[94] L. Wei, Z. Zhang, and A. Lim. An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *Computational Intelligence Magazine, IEEE*, 9(4):18–30, 2014.

[95] L. A. Wolsey. *Integer programming*. Wiley-Interscience, New York, NY, USA, 1998. ISBN 0-471-28366-5.

[96] L. A. Wolsey. *Integer Programming*. Wiley-Interscience publication, 1998.

[97] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley-Interscience. ISBN 0471359432.

[98] T. Yamada and S. Kataoka. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. In *Presented at EURO 2001: Rotterdam, The Netherlands*, 2001.

[99] T. Yamada, S. Kataoka, and K. Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43:2864–2870, 2002.

[100] Z. Yu, L. Lu, Y. Guo, R. Fan, M. Liu, and W. Wang. Content-Aware Photo Collage Using Circle Packing. *IEEE Transactions on Visualization and Computer Graphics*, 20 (2):182–195, 2014.

[101] T. C. Zachariadis E. and K. C. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195 (3):729–743, 2009.