Soluções Heurísticas para o Problema de Bin-Packing 2D com Tamanho Variado

Eduardo Augusto Militão Fernandes[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

Projeto Orientado em Computação II

eduardofernandes@dcc.ufmg.br

Abstract. *This work addresses the 2D Bin-Packing Problem with Varied Size and heuristic solutions for it. An extensive literature review on the problem is carried out, and the state-of-the-art literature algorithm for its variation with guillotine restriction is reproduced in order to validate the reported results and build a baseline to serve as a comparison with a proposed improvement for this algorithm and to enable the addition of a new restriction to the problem that meets an industry demand for its practical application.*

Resumo. *Este trabalho aborda o Problema de Bin-Packing 2D com Tamanho Variado e soluções heurísticas para ele. Uma extensiva revisão literária sobre o problema é realizada, e o algoritmo estado-da-arte da literatura para sua variação com restrição de guilhotina é reproduzido a fim de validar os resultados reportados e construir um baseline para servir de comparação com uma melhoria proposta para este algoritmo e para possibilitar a adição de uma nova restrição ao problema que atenda uma demanda da indústria para sua aplicação prática.*

1. Introdução

O Bin-Packing Problem (BPP) se trata do problema de, dadas caixas unidimensionais e diversos itens de comprimentos diferentes, determinar quais itens devem ser incluídos em cada caixa a fim de empacotar todos os itens e minimizar a quantidade de caixas utilizadas. Tal problema é famosamente NP-Difícil, o que significa que uma solução para o BPP é também uma solução para todos os problema na classe NP, e, indo além, não se conhece um algoritmo de complexidade polinomial para resolvê-lo. Dessa forma, o Bin-Packing Problem é um dos problemas clássicos mais estudados na computação, com diversos algoritmos heurísticos já tendo sido propostos para encontrar boas soluções, porém não necessariamente ótimas, de forma rápida para ele.

Nesse contexto, o Bin-Packing Problem 2D com Tamanho Variado (2DVSBPP) se trata de uma variação do BPP original, agora com caixas e itens bidimensionais retangulares, sendo que tais caixas possuem tamanhos diferentes, e o objetivo é empacotar todos os itens minimizando a área total gasta para tal (ou, da mesma forma, maximizando a utilização da área total gasta). Mais formalmente, dados N itens retangulares de larguras W_i e alturas H_i, com $i \in [1, N]$, e M caixas retangulares de larguras W_j e alturas H_j, com $j \in [1, M]$, deseja-se definir a posição de cada item em exatamente uma caixa tal

que os itens se encontrem inteiramente dentro de sua caixa, não há sobreposição de itens e as laterais de um item estejam paralelas às laterais de sua respectiva caixa, de forma a minimizar a soma das áreas das caixas utilizadas.

Por ser uma generalização do BPP, o 2DVSBPP também está na classe de problemas NP-Difíceis, e métodos para resolvê-lo se tornam ainda mais complexos e elaborados, dado o número exponencialmente maior de combinações possíveis de itens com caixas devido à adição de uma dimensão a mais. Além disso, o 2DVSBPP é de grande relevância para a indústria, tendo como aplicação imediata a determinação do layout de corte de chapas metálicas a fim de minimizar o desperdício de material. Assim, outra maneira de descrever o problema (que será utilizada ao longo do trabalho) é de determinar uma sequência de cortes verticais e horizontais no material base a fim de obter itens retangulares a partir dele. Torna-se, então, de alto interesse estudar este problema a fim de elaborar soluções mais eficientes para ele, dada sua relevância teórica para a computação assim como econômica, devido a sua aplicabilidade industrial.

1.1. Objetivos Gerais

Os objetivos deste trabalho são três principais: realizar uma vasta revisão da literatura para o Bin-Packing Problem 2D, com foco na sua versão com Tamanho Variado e outras possíveis restrições adicionais; reproduzir e verificar o algoritmo e os resultados obtidos e divulgados do atual estado-da-arte para este problema; propor uma melhoria a tal algoritmo, a fim de melhorar os seus resultados - levando em consideração não apenas o aproveitamento da área total gasta como também a praticidade real das soluções geradas - e possibilitar a introdução de uma nova restrição ao problema a fim de aumentar sua aplicabilidade na indústria.

Primeiramente, a revisão da literatura permitiu um melhor conhecimento sobre a nomenclatura, semântica e formalizações a respeito do BPP, além de um entendimento das soluções já existentes para o problema, quais são as principais estratégias e técnicas empregadas para resolvê-lo e para onde está caminhando a comunidade científica neste setor.

Já a reprodução e verificação do atual estado-da-arte é fundamental para o processo científico, dado que a reprodutibilidade de um *paper* é um dos principais critérios para medir sua qualidade. Além disso, deve-se ter certeza que os resultados divulgados são legítimos a fim de determinar o atual ponto das soluções para um problema.

Finalmente, como parte da contribuição deste trabalho para a comunidade científica, foi desenvolvida uma melhoria ao algoritmo reproduzido, com o objetivo de melhorar os atuais resultados para os casos de teste existentes para o problema e torná-lo mais aplicável situações do mundo real.

1.2. Objetivos Específicos

O projeto proposto foi dividido em duas etapas (POC I e POC II).

Assim, para a primeira etapa, foi realizada uma extensa revisão da literatura sobre o BPP 2D e suas variações, focando não apenas nos artigos mais recentes como também nos primeiros trabalhos que começaram a abordar a versão bidimensional deste problema. Além disso, também foi implementado por completo o algoritmo do estado-da-arte para

o 2DVSBPP e executá-lo nas instâncias disponíveis na literatura, a fim de verificar seus resultados.

Para a segunda etapa, o principal objetivo foi propor uma melhoria ao algoritmo estado-da-arte para o problema com o objetivo de obter soluções que não apenas otimizem a principal função-objetivo - aproveitamento da área total utilizada no empacotamento - mas que também sejam mais aplicáveis no mundo real, ou seja, otimizando também a facilidade de utilizar tais soluções na indústria. Este novo algoritmo foi executado na mesma máquina que o algoritmo reproduzido da literatura, de forma a tornar a comparação entre eles a mais justa possível. Finalmente, também foi introduzida a adição de uma nova restrição ao problema, alterando o algoritmo de forma a lidar com esta, a fim de torná-lo ainda mais aplicável industrialmente.

1.3. Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte maneira: a seção 2 apresenta a revisão da literatura realizada para o 2DVSBPP. A Seção 3 apresenta a metodologia utilizada no trabalho, especificando o processo de coleta de instâncias de teste, a implementação do algoritmo de baseline, proposição de uma melhoria para o algoritmo, adição de uma nova restrição ao problema e os resultados correspondentes obtidos. Finalmente, a seção 4 conclui o trabalho.

2. Revisão da Literatura

Em [Lodi et al. 1999], os autores propõem uma notação para definir 4 diferentes variações do Bin Packing Problem 2D:

- $2BP|O|G$: os itens não podem ser rotacionados e devem ser posicionados de forma a permitir corte por guilhotina
- $2BP|R|G$: os itens podem ser rotacionados e devem ser posicionados de forma a permitir corte por guilhotina
- $2BP|O|F$: os itens não podem ser rotacionados e a forma de corte é livre
- $2BP|R|F$: os itens podem ser rotacionados e a forma de corte é livre

Um corte por guilhotina é um corte reto, vertical ou horizontal, que deve ser feito de um lado a outro da caixa. Assim, a restrição de corte por guilhotina implica em achar uma configuração de disposição dos itens de forma a permitir que tais itens sejam sejam extraídos através apenas de cortes deste tipo. Tal restrição é muito comum em aplicações industriais. Ainda neste trabalho, uma heurística diferente é apresentada para cada uma das 4 variações do problema, e um algoritmo baseado em Busca Tabu que permite usar as 4 heurísticas para resolver a variação correspondente foi também apresentado, e se tornou o estado-da-arte da época.

No trabalho de [Pisinger and Sigurd 2005], é apresentada uma solução exata para o 2DVSBPP, minimizando o custo das caixas. Os autores utilizam uma formulação de Programação Inteira (PI) para o problema que, segundo o texto, é muito difícil de ser resolvida por solvers padrões. Em seguida, apresentam 4 *lower-bounds* para o problema, e então implementam um algoritmo de "branch-and-price" que utiliza tais bounds para resolver o problema otimamente. Um novo conjunto de instâncias para o problema, com 5 tamanhos de bins e quantidade de itens igual a 20, 40, 60, 80 e 100, também é proposto. É mostrado que um dos lower-bounds apresentados, que faz uso da decomposição de

Dantzig-Wolfe, é extremamente preciso, capaz de fornecer um bound de 3.89% na média no nó raiz da árvore do branch-and-cut. Tal lower-bound é encontrado através de uma MIP.

Em [Ortmann et al. 2010] é abordado o $2DVSBP|O|F$, ou seja, o $2BP|O|F$ porém com itens e caixas de tamanhos variados. Os autores apresentam um algoritmo two-phase para o problema. A primeira etapa consiste em resolver consecutivos problemas de Strip Packing, usando como largura da strip a largura da caixa atual, até que todos os itens tenham sido empacotados. Na segunda etapa, um processo de reempacotamento é realizado em cada caixa, na esperança de reempacotar todos os seus itens em uma caixa de área menor, através de sucessivas tentativas em caixas cada vez maiores. O algoritmo é testado em instâncias da literatura e novas instâncias propostas, cujo processo de geração é detalhadamente descrito.

Já em [Liu et al. 2011], os autores apresentam dois algoritmos baseados em Programação Dinâmica para resolver o 2DVSBPP com número de cada tamanho de bin limitado. O primeiro algoritmo, H1, resolve apenas o 2DVSBPP sem cortes de guilhotina e com restrição de orientação. Já o segundo, H2, é capaz de incorporar ou não cada uma dessas restrições. A DP não resolve o problema de forma ótima, uma vez que o número de estados necessários é exponencial. Dessa forma, é utilizada uma DP que resolve o problema de forma aproximada, ao unir vários estados em apenas 1, perdendo assim informação mas ganhando tempo e espaço. Na média, o algoritmo H1 é capaz de obter soluções consideravelmente melhores que H2, quando as restrições que H2 pode incorporar não estão em questão.

[Alvarez-Valdés et al. 2013] ataca o 2D e 3D Variable Sized and Variable Cost Bin Packing Problem. Os autores propõem um algoritmo de duas fases para resolver o problema. A primeira fase consiste de um GRASP, que gerará um conjunto de soluções ótimas locais. Já a segunda fase é um algoritmo de Path Relinking, responsável por combinar as soluções da primeira fase em uma solução melhor.

Já em [Wei et al. 2013], é proposta uma solução que faz uso do paradigma de dividir-e-conquistar, Busca Tabu e Busca Binária para resolver o $2DVSBP|O|F$. O algoritmo foi testado em diversas instâncias da literatura e é atualmente o estado-da-arte para esta versão do problema.

[Hong et al. 2014] aborda o $2DVSBP|O|G$ e apresenta uma heurística baseada em Backtracking na escolha do próximo tamanho de caixa a ser utilizado, chamada de BTVS. Para cada tamanho escolhido, um algoritmo semi-guloso é utilizado para tentar preencher a caixa com a maior área de itens possível. Este algoritmo considera diferentes ordenações dos itens. O algoritmo semi-guloso, por sua vez, faz uso de dois algoritmos gulosos, um semi-guloso e outro guloso, descritos no artigo, e no final é escolhido a melhor solução retornada por um dos dois algoritmos. Os autores também apresentam uma heurística baseada em Simulated Annealing, onde a função de vizinhança consiste simplesmente em selecionar dois itens aleatórios e trocar a ordem deles, projetada para melhorar a solução encontrada pelo BTVS.

Em [Bogue et al. 2021], os autores propõem o Problema de Corte Guilhotinado Bidimensional em Três Estágios com Restrições de Precedência em Lote, que é semelhante ao 2DVSBPP, porém com a restrição adicional de que as soluções podem ter no

máximo profundidade igual a três - ou seja, devem se tratar de um conjunto de cortes verticais, seguido de um conjunto de horizontais, seguido por um conjunto final de cortes verticais - e a ordem dos itens sendo obtidos obedeça uma sequência pré-definida. Tais restrições são motivadas por demandas da indústria de cortes de vidro. São apresentadas então três heurísticas para tal problema, sendo que uma faz uso de uma subrotina de um algoritmo de programação dinâmica, e são realizados experimentos em instâncias disponíveis publicamente.

Em [Polyakovskiy and M'Hallah 2022], os autores apresentam uma *matheuristic* para a solução do $2DVSBP|O|G$. Neste algoritmo, soluções viáveis são sucessivamente construídas ao resolver programações inteiras iterativamente, resolvendo sub-problemas de cada vez. 3 PIs diferentes são apresentadas, cada uma melhor porém mais custosa que a outra. O algoritmo supera o então estado da arte, tanto no problema em questão quanto no problema de 2D Single Size BPP. Neste último, o procedimento foi capaz de provar a otimalidade da solução em 415 das 500 instâncias testadas.

[Gardeyn and Wauters 2022] propõe uma solução para o $2DVSBP|*|G$, ou seja, a restrição de orientação pode ser aplicada ou não. Os autores usam uma árvore enraizada para representar uma solução, onde nós folhas são itens / espaços vazios, e os nós internos definem em qual direção os nós filhos são cortados (vertical ou horizontal). O algoritmo proposto é baseado no paradigma de Ruin-and-Recreate, e experimentos computacionais mostram que o algoritmo é o novo estado-da-arte para o $2DVSBP| * |G$, e iguala ao estado-da-arte no $2DBP|*|G$, embora apresente tempos de execução consideravelmente maiores que os concorrentes.

Finalmente, [Mezghani et al. 2023] realiza uma vasta revisão bibliográfica do Bin Packing Problem original, expondo sua notação formal, aplicações e, principalmente, propõe uma nota notação para catalogar todas as variações e restrições do BPP. Além disso, ainda apresenta estatísticas que ilustram como o problema tem sido mais ou menos pesquisado com o passar dos anos.

3. Metodologia e Contribuição

3.1. Coleta de instâncias de teste

A partir da revisão bibliográfica, foi decidido coletar e utilizar as instâncias de teste propostas em [Pisinger and Sigurd 2005], [Ortmann et al. 2010] e [Hopper and Turton 2002], para um total de 855 instâncias. O primeiro conjunto contém 500 instâncias separadas em 10 grupos, nomeadamente MC_B1, até MC_B10. O segundo conjunto consiste em 340 instâncias separadas em dois grupos de 170 cada, onde o primeiro, Nice, contém instâncias de itens de tamanho homogêneo, enquanto o segundo, Path, contém itens de tamanhos e dimensões vastamente diferentes e extremas. Já o último conjunto contém 15 instâncias divididas em 3 grupos: M1, M2 e M3.

3.2. Implementação de baseline da literatura

O algoritmo escolhido para reprodução e implementação foi o proposto em [Gardeyn and Wauters 2022], para o $2DVSBP| * |G$, por se tratar de trabalho extremamente recente, ser o atual estado da arte para este problema e ser bem especificado no artigo original. Toda a descrição do algoritmo apresentada pelos autores foi seguida à risca, e as partes do algoritmo não detalhadas no artigo original foram implementadas

com o foco em minimizar a complexidade de tempo e de memória do algoritmo, embora mantendo o comportamento esperado.

O algoritmo é baseado em uma heurística *goal-driven* de *ruin and recreate*. Uma árvore enraizada representa uma solução, onde nós folhas são itens / espaços vazios, e os nós internos definem em qual direção os nós filhos são cortados. Mais especificamente, se um nó interno representa cortes em uma determinada direção, o seu nó filho que também é interno representa cortes na direção oposta à de seu pai (ou seja, um nó interno representa uma sequência de cortes na mesma direção). O procedimento usa dois valores para guiar a busca por soluções melhores: área total das caixas utilizados e área total dos itens não empacotados. Isso porque o procedimento lida com soluções inválidas ao invés de descartá-las. A fase de Ruin consiste em excluir nós (de itens ou internos) aleatórios da árvore da solução até que um certo número máximo de nós tenha sido excluído e enquanto a área da solução modificada for maior que o limite de área. A fase de Recreate tenta inserir cada item não empacotado na solução, podendo ou não abrir novas caixas para tal. Para cada item, este é inserido no melhor espaço vazio para ele, considerando ambas orientações para ele e para o corte que o gerará. Entretanto, para introduzir um elemento estocástico, há uma probabilidade de uma das opções ser ignorada, podendo levar a inserção em um espaço sub-ótimo. Finalmente, para decidir se a nova solução gerada será aceita ou não, é feito uso da metaheurística Late-Acceptance Hill-Climbing. Nessa metaheurística, uma solução nova é aceita ou não se, e somente se, ela é melhor que a solução atual há X iterações atrás. Algorithm 1 apresenta uma descrição geral a alto nível do algoritmo.

Algorithm 1 Gardeyn and Wauters 2021 - Ruin and Recreate Heuristic

$S, S^*, S' \leftarrow \emptyset$
while Há tempo restante **do**
    $S' \leftarrow RuinSolution(S)$ *//remover aleatoriamente sub-árvores da solução*
    $S' \leftarrow RecreateSolution(S')$ *//selecionar itens para preencher espaços vazios*
    $S'\_val \leftarrow EvaluateSolution(S')$
    if $EvaluateSolution(S') < EvaluateSolution(S)$ **then**
        $S \leftarrow S'$
        if $SolutionIsFeasible(S)$ **then**
            if $EvaluateSolution(S') < EvaluateSolution(S*)$ **then**
                $S^* \leftarrow S'$
            end if
        end if
    end if
end while

3.2.1. Resultados

Nossa implementação do algoritmo foi executada para todas as 855 instâncias descritas na seção 3.1. O código foi desenvolvido em C++11 e compilado com o GCC v9.4.0. Os testes foram executados em uma máquina Ubuntu 18.04 com um Intel Core i7 980 e 24GB de memória RAM. No artigo original, cada instância foi executada por 10 minutos

	Sol Value Mean (%)			
Class Name	**Ours (10m)**	**Ours (25m)**	**Gardeyn**	**Diff (G - O(25m))**
M1	98,38	98,38	98,38	0,00
M2	96,53	97,43	97,87	0,44
M3	97,98	98,26	98,85	0,59
Average	97,63	98,02	98,37	0,34

Tabela 1. Resultados obtidos vs originais nas instâncias de [Hopper and Turton 2002] no $2VSBP|R|G$

	Sol Value Mean (%)			
Class Name	**Ours (10m)**	**Ours (25m)**	**Gardeyn**	**Diff (G - O(25m))**
Nice25i	91,70	91,89	96,82	4,93
Nice50i	90,31	91,25	92,97	1,72
Nice100i	87,41	88,76	94,49	5,73
Nice200i	89,38	90,64	94,21	3,57
Nice300i	88,24	89,99	93,10	3,11
Nice400i	89,89	91,51	93,74	2,23
Nice500i	88,29	90,22	93,54	3,32
Path25i	91,72	91,89	98,88	6,99
Path50i	94,34	95,27	96,49	1,22
Path100i	92,85	93,89	97,17	3,28
Path200i	92,58	93,73	96,56	2,83
Path300i	93,29	94,15	95,85	1,70
Path400i	93,47	94,44	97,00	2,56
Path500i	91,60	94,08	96,56	2,48
Average	91,08	92,27	95,53	3,26

Tabela 2. Resultados obtidos vs originais nas instâncias de [Ortmann et al. 2010] no $2VSBP|R|G$

com 8 threads. Como não foi implementado multi-threading neste trabalho e para analisar o impacto do tempo de execução na performance do algoritmo, foi decidido executar a nossa versão por 25 minutos, reportando então o resultado final e o resultado obtido nos 10 primeiros minutos de cada execução.

As Tabelas 1, 2 e 3 exibem os resultados obtidos nas instâncias de [Hopper and Turton 2002], [Ortmann et al. 2010] e [Pisinger and Sigurd 2005], respectivamente. Cada linha apresenta o valor médio do aproveitamento, em porcentagem, obtido em cada subconjunto de instância, para o nosso algoritmo com 10 e 25 minutos de execução, o resultado reportado em [Gardeyn and Wauters 2022] e a diferença entre o resultado do artigo original e da nossa execução com 25 minutos. A linha final apresenta as médias de cada coluna. O aproveitamento é definido como a soma da área de todos os itens dividida pela soma da área de todas as caixas utilizadas na solução encontrada. Logo, seu valor máximo é de 1 (100%).

Através da análise dos resultados, pode-ser perceber que os resultados obtidos pelo nossa implementação se aproximam dos resultados reportados no artigo original, e, portanto, trata-se de uma reprodução fiel do algoritmo original, além de uma validação positiva de tais resultados reportados em [Gardeyn and Wauters 2022]. Esta implementação,

	Sol Value Mean (%)			
Class Name	**Ours (10m)**	**Ours (25m)**	**Gardeyn**	**Diff (G - O(25m))**
MB_C1	94,03	95,19	97,20	2,01
MB_C2	96,45	96,65	96,75	0,10
MB_C3	90,64	92,09	95,32	3,23
MB_C4	94,39	95,33	96,54	1,21
MB_C5	88,87	90,11	93,69	3,58
MB_C6	91,94	92,93	94,68	1,75
MB_C7	90,30	91,46	95,01	3,55
MB_C8	89,74	91,18	95,25	4,07
MB_C9	75,22	76,51	79,99	3,48
MB_C10	91,84	93,27	96,01	2,74
Average	90,34	91,47	94,04	2,57

Tabela 3. Resultados obtidos vs originais nas instâncias de [Pisinger and Sigurd 2005] no $2VSBP|R|G$

então, serviu de baseline para as etapas seguintes do trabalho, uma vez que foi possível comparar a performance entre o algoritmo o modificações propostas mantendo constante o programador e a máquina utilizada para as execuções.

3.3. Melhoria proposta ao algoritmo baseline

Embora o objetivo principal do 2DVSBPP e, por consequência, do algoritmo de Gardeyn, seja maximizar o percentual de área utilizada das caixas pela solução, em aplicações práticas esta não é a única prioridade levada em conta para construir boas soluções para o problema. Mais especificamente, em aplicações industriais onde o 2DVSBPP é utilizado para definir uma sequência de cortes guilhotinados em chapas de metal/vidro a fim de se obter pedaços menores, um dos critérios de uma solução factível é o número de cortes a serem feitos ou, em cenários mais particulares, o número de trocas entre uma sequência de cortes em uma orientação (por exemplo, vertical) para uma sequência de cortes na outra orientação (por exemplo, horizontal), como abordade em [Bogue et al. 2021]. Neste cenário, este último critério pode ser medido como a profundidade máxima dentre as árvores que representam uma solução construída pelo algoritmo baseline, uma vez que, como já descrito, um nó interno de uma árvore de solução representa uma sequência de cortes em uma mesma direção, e os nós imediatamente filhos deste representam cortes na direção oposta.

Dessa forma, foi decidido medir a profundidade máxima das soluções geradas pela nossa reprodução do algoritmo descrito acima. Os resultados podem ser conferidos nas Tabelas 4, 5, 6.

Percebeu-se, então, que nas instâncias de [Pisinger and Sigurd 2005] e, principalmente, de [Ortmann et al. 2010], muitas das soluções encontradas tiveram profundidades consideradas impraticáveis pelos padrões da indústria (em [Bogue et al. 2021], o requisito da indústria é de profundidade máxima de 3). Tal resultado motivou a elaboração de uma modificação ao algoritmo original, que não apenas visasse melhorar o aproveitamento da área utilizada na solução, mas que, principalmente, focasse em gerar soluções de profundidades menores, menos variáveis e sem valores extremos (em certas instâncias, o método original gerou soluções com profundidade acima de 20).

	Deepest Level			
Class Name	**Mean**	**Max**	**Min**	**Std**
Nice25i	4,90	7	3	1,21
Nice50i	6,60	12	5	1,71
Nice100i	7,56	13	5	2,14
Nice200i	8,68	16	6	2,14
Nice300i	9,84	18	6	2,82
Nice400i	10,28	17	7	2,61
Nice500i	11,24	25	8	3,53
Path25i	5,35	10	3	1,69
Path50i	7,20	10	5	1,41
Path100i	8,52	12	7	1,48
Path200i	10,64	14	7	2,06
Path300i	11,72	16	9	1,65
Path400i	11,96	19	8	2,73
Path500i	12,76	19	9	2,20
Average	9,09	14,86	6,29	2,10

Tabela 4. Profundidade máxima das soluções obtidas nas instâncias de [Ortmann et al. 2010] no $2VSBP|R|G$

	Deepest Level			
Class Name	**Mean**	**Max**	**Min**	**Std**
MB_C1	4,56	7	2	1,32
MB_C2	5,70	8	4	1,02
MB_C3	4,70	6	4	0,61
MB_C4	6,62	9	4	1,05
MB_C5	4,96	6	4	0,70
MB_C6	7,12	10	6	1,12
MB_C7	4,88	7	4	0,80
MB_C8	4,80	6	3	0,76
MB_C9	4,16	6	3	0,74
MB_C10	5,68	7	4	0,71
Average	5,32	7,20	3,80	0,88

Tabela 5. Profundidade máxima das soluções obtidas nas instâncias de [Pisinger and Sigurd 2005] no $2VSBP|R|G$

	Deepest Level			
Class Name	**Mean**	**Max**	**Min**	**Std**
M1	6,00	7	5	0,71
M2	5,00	5	5	0,00
M3	5,40	7	4	1,14
Average	5,47	6,33	4,67	0,62

Tabela 6. Profundidade máxima das soluções obtidas nas instâncias de [Hopper and Turton 2002] no $2VSBP|R|G$

3.3.1. Descrição do algoritmo proposto

Ao visualizar as soluções geradas pelo reprodução do algoritmo original, percebeu-se que, muitas vezes, a alta profundidade da solução estava concentrada em algumas regiões específicas da caixa onde muitos itens eram agrupados com direções de corte que alternavam com alta frequência. Dessa forma, a solução proposta para mitigar este problema, de maneira a gerar soluções de menos profundidade mas que mantivessem - e, possivelmente aumentassem - o aproveitamento das caixas, é de utilizar um algoritmo de Programação Dinâmica (DP) para, dado um espaço vazia na caixa, escolher um conjunto de itens a serem dispostos lado a lado neste espaço de forma a maximar a soma da área dos itens selecionados. Tal DP é equivalente à resolução do Knapsack Problem, onde devem ser escolhidos itens com peso p_i e valor v_i a fim de maximar a soma de seus valores sem que a soma de seus pesos ultrapasse um limite pré-definido. Neste caso, v_i é a área do item i e p_i é ou a largura ou a altura do item, dependendo de como este for disposto na área.

Mais formalmente, seja I o conjunto de itens ainda não adicionados à solução tais que cada item caberia individualmente no espaço vazio a ser preenchido, H a altura do espaço e W sua largura. Seja também h_i a altura do item $i \in I$ e w_i a largura de $i \in I$. Se $V > H$, então os itens serão dispostos de cima para baixo no espaço vazio. Caso contrário, da esquerda para a direita, lado a lado. Sem perda de generalização, considerando que $W > H$, os itens são escolhidos segundo a seguinte equação de recorrência:

$$F(k,l) = \begin{cases} 0, & \text{se } k = 0. \\ -\infty, & \text{se } l < 0. \\ max\{F(k-1,l), F(k-1,l-w_k) + (w_k \cdot h_k)\}, & \text{se } w_k \leq h_k \cup w_k > H. \\ max\{F(k-1,l), F(k-1,l-h_k) + (w_k \cdot h_k)\}, & \text{se } h_k \leq w_k \cup h_k > H. \end{cases} \quad (1)$$

A chamada inicial da equação é $F(\|I\|, W)$. A complexidade de tempo e de memória assintóticas deste algoritmo são iguais a $O(\|I\| \cdot W)$, uma vez que são computados e armazenados os valores de $F(k,l)$ para todo par de valores ($k \in \{1...\|I\|\}, l \in \{1...W\}$).

Assim, a fim de preservar o funcionamento central do algoritmo original e manter seu caráter estocástico, decidiu-se introduzir tal preenchimento por DP como parte da etapa de *Recreate* do algoritmo original. Dessa forma, após cada inserção de um item durante a execução desta etapa, um dos novos espaços vazios gerados por tal inserção (selecionado aleatoriamente com iguais chances) é preenchido com o novo método proposto com probabilidade $p = 1 - \gamma^d$, onde d é a profundidade do nó correspondente ao espaço vazio em questão e $\gamma \in \mathbb{R}$ é um parâmetro entre 0 e 1. Ou seja, quanto mais profundo, maior a chance do novo método ser invocado. Após experimentação, foi decidido utilizar $\gamma = 0.6$. Além disso, após a definição pela DP de quais itens devem ser inseridos, estes são cortados do mais alto para o mais baixo, da esquerda para a direita, e a direção de seus cortes é definida utilizando o mesmo critério do artigo original.

3.3.2. Resultados

Uma vez que soluções com profundidade excessivas foram observadas em instâncias de teste dos conjuntos de [Ortmann et al. 2010] e de [Pisinger and Sigurd 2005], o algoritmo

	Sol Value Mean (%)			Deepest Level Mean		
Class Name	**Gardeyn**	**Proposed**	**Diff (G - P)**	**Gardeyn**	**Proposed**	**Diff (G - P)**
Nice25i	91,62	92,24	-0,61	4,90	5,20	-0,30
Nice50i	91,25	90,92	0,32	6,60	6,04	0,56
Nice100i	88,54	88,27	0,27	7,56	6,68	0,88
Nice200i	90,60	90,41	0,19	8,68	8,20	0,48
Nice300i	89,88	90,04	-0,16	9,84	8,72	1,12
Nice400i	91,14	91,68	-0,54	10,28	8,72	1,56
Nice500i	90,18	91,31	-1,13	11,24	8,84	2,40
Path25i	92,20	92,89	-0,69	5,35	5,40	-0,05
Path50i	94,80	94,81	-0,01	7,20	7,72	-0,52
Path100i	93,80	93,21	0,58	8,52	8,88	-0,36
Path200i	94,79	92,94	1,85	10,64	10,52	0,12
Path300i	94,11	93,45	0,66	11,72	10,40	1,32
Path400i	94,59	94,11	0,48	11,96	10,76	1,20
Path500i	93,80	91,99	1,81	12,76	11,24	1,52
Average	92,24	92,02	0,22	9,09	8,38	0,71

Tabela 7. Resultados obtidos vs originais nas instâncias de [Ortmann et al. 2010] no $2VSBO|R|G$

com a modificação proposta foi executado nestas instâncias, na mesma máquina e mesmos parâmetros descritos na seção 3.2.1, por 25 minutos cada. Os resultados estão dispostos na Tabelas 7 e 8. Além disso, a fim de visualizar a diferença na variação da profundidade das soluções encontradas dentro de um mesmo conjunto de instâncias, foram gerados *boxplots* com a distribuição de tais profundidade, disponíveis nas Figuras 1 e 2.

Pode-se perceber que tanto nas instâncias de Ortmann quanto de Pisinger, não foi observada diferença significativa entre o percentual de aproveitamento das soluções geradas pelos dois métodos. Todavia, ao observar o boxplot da distribuição das profundidades das soluções, é evidente que, principalmente nos conjuntos de instâncias mais desafiadores do dataset de Ortmann - Nice300i, Nice400i, Nice500i, Path300i, Path400i e Path500i -, as soluções encontradas pelo algoritmo proposto possuem profundidades menores, menos variadas e sem valores extremos como os exibidos pelo algoritmo original. Tal diferença, porém, não está presente no conjunto de Pisinger, onde os resultados exibidos pelos dois métodos são extremamente próximos.

3.4. Adição de restrição adicional ao 2DVSBPP

Finalmente, com o objetivo de obter maior controle sobre a profundidade das soluções encontradas, decidiu-se implementar ao algoritmo com a modificação proposta uma restrição rígida sobre a profundidade máxima da solução gerada, parametrizada como *input* ao algoritmo. Tal restrição foi implementada de forma a minimizar o impacto sobre a qualidade das soluções encontradas sem a restrição. Assim, o método de preenchimento por DP passa a poder ser executado apenas em espaços vazios cujo nó correspondente possui profundidade inferior à máxima menos 1, a fim de impedir a escolha de um item cujo corte aumente a profundidade da solução para além do permitido. Em outras etapas do algoritmo, na seleção da ordem dos cortes a serem adicionados e nos itens a serem inseridos, também são restringidas tais escolhas para evitar a quebra da restrição.

	Sol Value Mean (%)			Deepest Level Mean		
Class Name	**Gardeyn**	**Proposed**	**Diff (G - P)**	**Gardeyn**	**Proposed**	**Diff (G - P)**
MB_C1	95,18	95,43	-0,25	4,56	4,62	-0,06
MB_C2	96,66	96,70	-0,03	5,70	5,46	0,24
MB_C3	91,97	92,30	-0,33	4,70	4,78	-0,08
MB_C4	95,23	95,35	-0,12	6,62	6,38	0,24
MB_C5	89,98	90,48	-0,51	4,96	5,20	-0,24
MB_C6	92,67	92,52	0,15	7,12	7,08	0,04
MB_C7	91,40	90,95	0,45	4,88	4,98	-0,10
MB_C8	91,04	91,11	-0,07	4,80	4,84	-0,04
MB_C9	76,60	76,41	0,19	4,16	4,10	0,06
MB_C10	93,22	93,44	-0,22	5,68	5,72	-0,04
Average	91,39	91,47	-0,07	5,32	5,32	0,00

Tabela 8. Resultados obtidos vs originais nas instâncias de [Pisinger and Sigurd 2005] no $2DVSBP|R|G$

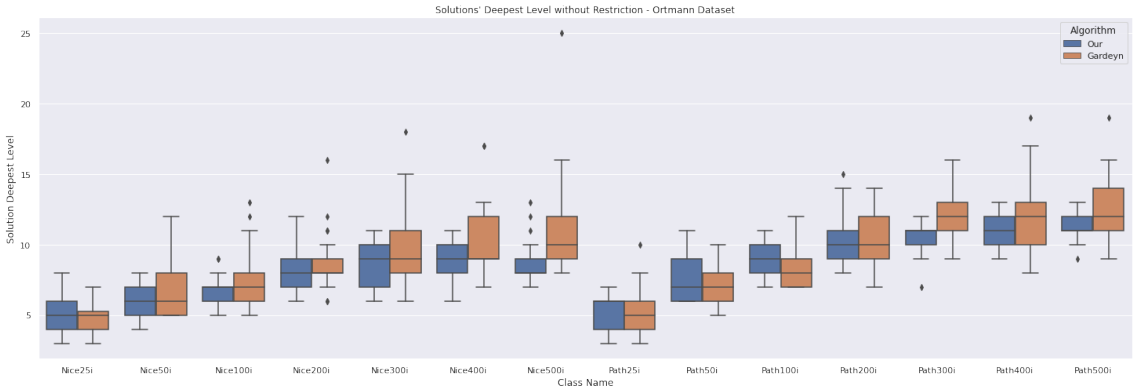

Figura 1. Diferença da profundidade das soluções entre o algoritmo original e a modificação proposta, nas instâncias de [Ortmann et al. 2010]

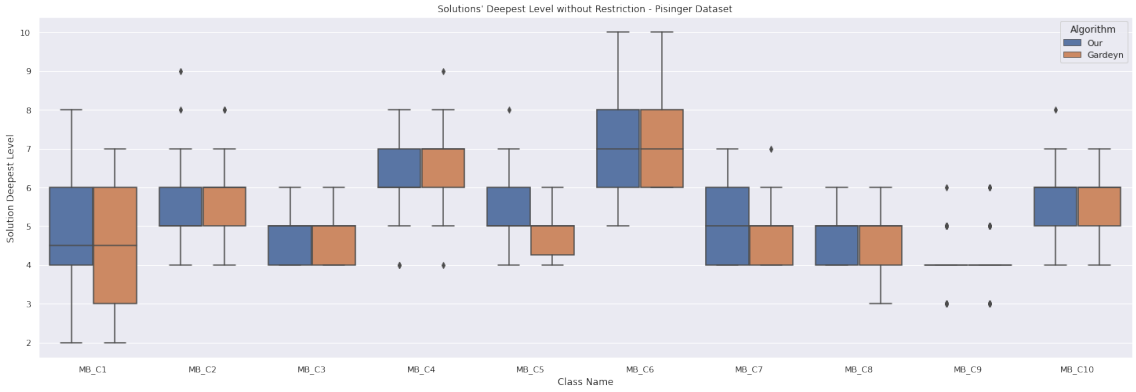

Figura 2. Diferença da profundidade das soluções entre o algoritmo original e a modificação proposta, nas instâncias de [Pisinger and Sigurd 2005]

3.4.1. Resultados

O algoritmo proposto com a restrição rígida de profundidade máxima foi executado para as instâncias de [Ortmann et al. 2010] e de [Pisinger and Sigurd 2005], na mesma

	Sol Value Mean (%)		
Class Name	**Without Rest.**	**With Rest.**	**Diff (WO - W)**
Nice25i	92,24	91,61	0,62
Nice50i	90,92	89,49	1,43
Nice100i	88,27	87,48	0,80
Nice200i	90,41	89,95	0,46
Nice300i	90,04	89,46	0,57
Nice400i	91,68	91,12	0,57
Nice500i	91,31	90,24	1,07
Path25i	92,89	92,01	0,88
Path50i	94,81	94,35	0,46
Path100i	93,21	92,72	0,49
Path200i	92,94	92,42	0,52
Path300i	93,45	92,86	0,59
Path400i	94,11	93,29	0,82
Path500i	91,99	91,08	0,91
Average	92,02	91,29	0,73

Tabela 9. Resultados obtidos vs originais nas instâncias de [Ortmann et al. 2010] no $2DVSBP|R|G$

máquina e parâmetros descritos na Seção 3.2.1, com mesmo valor de $gamma$ citado na Seção 3.3.1, e profundidade máxima de 7. Os resultados comparando a performance do algoritmo com e sem a restrição estão nas Tabelas 9 e 10. Além disso, as Figuras 3.4.1 e 3.4.1 exibem a distribuição das profundidades máximas das soluções do algoritmo proposto com e sem a nova restrição.

Percebe-se, portanto, que foi possível adicionar a nova restrição impactando minimamente a qualidade das soluções encontradas, uma vez que a diferença média entre o aproveitamento da área das soluções sem restrição e com a restrição é de menos de 1%. Mesmo nos conjuntos mais desafiadores do dataset de Ortmann, em que as soluções originais são bastante profundas, o algoritmo com restrição foi capaz de encontrar soluções de qualidade praticamente iguais às do sem a restrição.

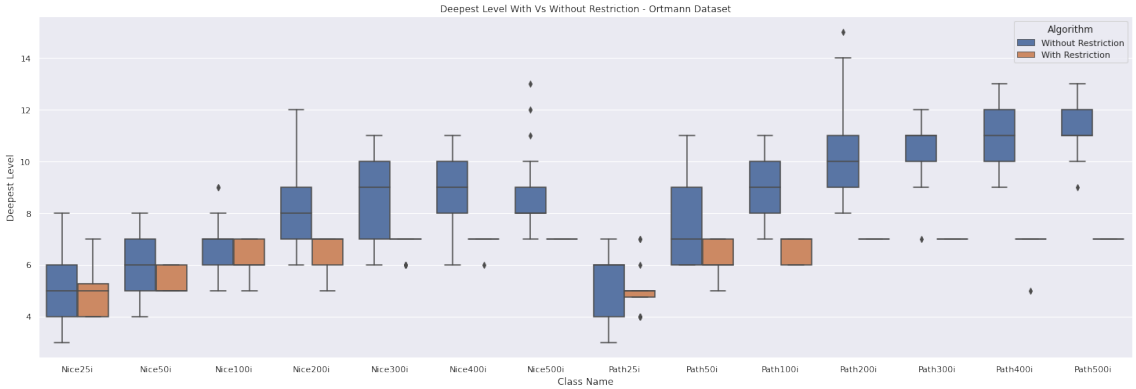

Figura 3. Diferença da profundidade das soluções entre o algoritmo proposto sem restrição e com restrição, nas instâncias de [Ortmann et al. 2010]

	Sol Value Mean (%)		
Class Name	**Without Rest.**	**With Rest.**	**Diff (WO - W)**
MB_C1	95,43	94,74	0,69
MB_C2	96,70	96,62	0,08
MB_C3	92,30	91,45	0,85
MB_C4	95,35	94,56	0,78
MB_C5	90,48	89,63	0,85
MB_C6	92,52	92,05	0,47
MB_C7	90,95	89,61	1,34
MB_C8	91,11	88,84	2,27
MB_C9	76,41	75,70	0,71
MB_C10	93,44	92,53	0,91
Average	91,47	90,57	0,89

Tabela 10. Resultados obtidos vs originais nas instâncias de [Pisinger and Sigurd 2005] no $2DVSBP|R|G$

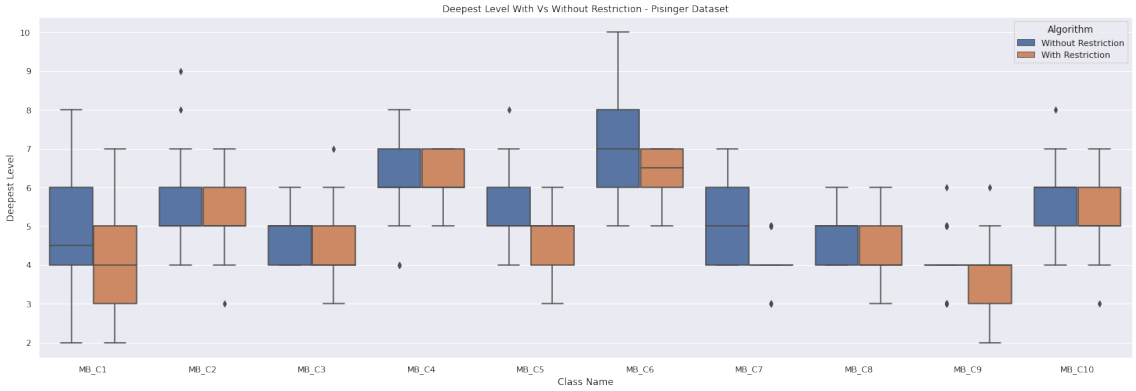

Figura 4. Diferença da profundidade das soluções entre o algoritmo proposto sem restrição e com restrição, nas instâncias de [Pisinger and Sigurd 2005]

4. Conclusão

Este trabalho apresenta uma extensa revisão da literatura sobre o 2D Bin Packing Problem, com foco em sua versão de tamanho variado, a re-implementação do algoritmo estado-da-arte para o 2D Bin Packing Problem com restrição de cortes de guilhotina, a proposta de uma modificação a este algoritmo a fim de melhorar a praticidade real das soluções encontradas por ele e a adição de uma restrição adicional ao 2DVSBPP e, correspondentemente, ao algoritmo proposto a fim de lidar com esta.

Através dos resultados obtidos pelo código reproduzido, foi possível verificar os resultados reportados no artigo do algoritmo original. Além disso, tal reprodução serviu de baseline para comparar a modificação proposta a partir deste a fim de melhorar ainda mais seus resultados e torná-los mais práticos para o mundo real, uma vez que foram mantidas constantes variáveis como o conhecimento e experiência do programador e o ambiente de execução.

Ademais, a proposta de uma modificação ao algoritmo original de forma a utilizar

uma DP para escolher de forma mais otimizada os itens a preencherem espaços vazios nas caixas de maneira a reduzir a profundidade final da solução - e, portanto, torná-la mais prática em aplicações industriais - tornou tal método mais estável sob este critério, ou seja, capaz de encontrar soluções sem valores extremos de profundidade máxima e menor variação de profundidade entre instâncias semelhantes. Finalmente, a adição de uma restrição rígida para a profundidade máxima das soluções geradas pelo algoritmo deu mais controle sobre o algoritmo e tornou-o ainda mais aplicável em situações práticas que exigem tal restringibilidade.

Referências

Alvarez-Valdés, R., Parreño, F., and Tamarit, J. M. (2013). A grasp/path relinking algorithm for two-and three-dimensional multiple bin-size bin packing problems. *Computers & Operations Research*, 40(12):3081–3090.

Bogue, E. T., Guimarães, M. V., Noronha, T. F., Pereira, A. H., Carvalho, I. A., and Urrutia, S. (2021). The two-dimensional guillotine cutting stock problem with stack constraints. In *2021 XLVII Latin American Computing Conference (CLEI)*, pages 1–9. IEEE.

Gardeyn, J. and Wauters, T. (2022). A goal-driven ruin and recreate heuristic for the 2d variable-sized bin packing problem with guillotine constraints. *European Journal of Operational Research*, 301(2):432–444.

Hong, S., Zhang, D., Lau, H. C., Zeng, X., and Si, Y.-W. (2014). A hybrid heuristic algorithm for the 2d variable-sized bin packing problem. *European Journal of Operational Research*, 238(1):95–103.

Hopper, E. and Turton, B. (2002). Problem generators for rectangular packing problems. *Stud. Inform. Univ.*, 2(1):123–136.

Liu, Y., Chu, C., and Wang, K. (2011). A dynamic programming-based heuristic for the variable sized two-dimensional bin packing problem. *International Journal of Production Research*, 49(13):3815–3831.

Lodi, A., Martello, S., and Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS journal on computing*, 11(4):345–357.

Mezghani, S., Haddar, B., and Chabchoub, H. (2023). The evolution of rectangular bin packing problem — a review of research topics, applications, and cited papers.

Ortmann, F. G., Ntene, N., and Van Vuuren, J. H. (2010). New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. *European Journal of Operational Research*, 203(2):306–315.

Pisinger, D. and Sigurd, M. (2005). The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154–167.

Polyakovskiy, S. and M'Hallah, R. (2022). A lookahead matheuristic for the unweighed variable-sized two-dimensional bin packing problem. *European Journal of Operational Research*, 299(1):104–117.

Wei, L., Oon, W.-C., Zhu, W., and Lim, A. (2013). A goal-driven approach to the 2d bin packing and variable-sized bin packing problems. *European Journal of Operational Research*, 224(1):110–121.