Relatório: Bin Packing Problem

Bianca Beppler Dullius, Caroline Souza Camargo, Yasmin Souza Camargo
Universidade Federal de Pelotas (UFPel)
Pelotas, Brasil
{bianca.bd, caroline.sc, yasmin.sc}@inf.ufpel.edu.br

I. INTRODUÇÃO

O problema de empacotamento de caixas (Bin Packing Problem) é um desafio de otimização combinatorial que envolve a alocação eficiente de itens de diferentes tamanhos em caixas de capacidade limitada. O objetivo é minimizar o número de caixas utilizadas para acomodar todos os itens, garantindo que nenhuma caixa exceda sua capacidade máxima. O Problema de Empacotamento de Caixas pertence à classe NP-difícil o que significa que não há um algoritmo eficiente (em tempo polinomial) para encontrar a solução ótima para todas as instâncias do problema. Logo, encontrar uma solução ideal se torna um desafio cada vez mais demorado à medida que os tamanhos dos itens e a quantidade de caixas aumentam. Nesse contexto, este relatório irá explorar a implementação prática deste problema e realizar a análise dos resultados encontrados.

II. ALGORITMO EXATO

Esta etapa consistiu em desenvolver um algoritmo que tenha a solução ótima para o problema bin packing. O algoritmo dessa solução é baseado em força bruta, ou seja, todas as permutações possíveis são geradas para os itens de entrada. Isso significa que a complexidade é O(n!), onde "n" é o número de itens a serem empacotados. Na solução, são geradas todas as permutações possíveis dos itens usando recursão. Quando uma permutação completa é gerada, a solução aplica uma instância do algoritmo Next Fit, onde os itens da permutação são dispostos em caixas na ordem em que estão. Se a solução encontrada foi melhor do que a melhor solução atual, ela é armazenada como a melhor solução atual.

Para essa solução, podemos observar o seguinte gráfico da relação entre quantidade de itens a serem empacotados e tempo de execução:

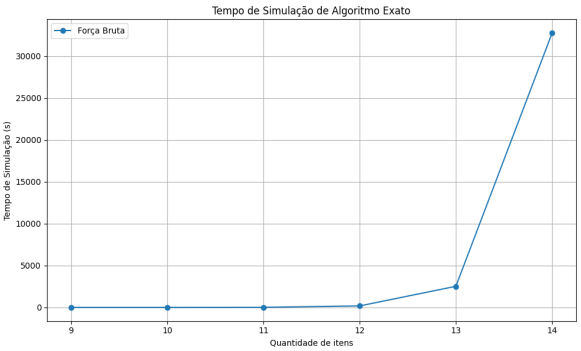

Fig. 1. Gráfico do tempo de simulação do algoritmo exato

O gráfico de tempo de execução do algoritmo exato mostra um aumento exponencial no tempo à medida que o número de elementos aumenta. Analisando o mesmo, podemos observar que: até 11 elementos o algoritmos executa em questão de segundos; acima disso, com 12 elementos foram gastos 3 minutos; para 13 elementos foram 42 minutos; para 14 elementos foram 9 horas; e para 15 elementos estimamos, com os valores obtidos anteriormente, que o algoritmo levaria aproximadamente 5 dias para executar todas as permutações e encontrar a melhor solução.

Com esses resultados, podemos concluir que seria inviável executar o algoritmo exato para alguns dos dados de experimentos que temos disponível, sendo o maior deles com 277 elementos. Até mesmo para um experimento de 33 elementos, realizando uma estimativa com os valores obtidos com os experimentos realizados, chegamos ao tempo de execução de $1.0339922047303978 \times 10^{23}$ anos.

Embora o algoritmo exato possa encontrar a solução ótima, sua complexidade torna-o impraticável para conjuntos de itens grandes. Para esses casos, é necessário explorar alternativas mais eficientes, como algoritmos de aproximação, heurísticas ou métodos de busca local, que podem oferecer soluções aceitáveis em tempo hábil. Portanto, a escolha do algoritmo deve ser baseada nas restrições de tempo e recursos disponíveis, bem como no tamanho do conjunto de itens a serem empacotados. A seguir, vamos explorar algumas soluções de algoritmos aproximativos.

III. ALGORITMOS APROXIMATIVOS

Os algoritmos de aproximação, embora não garantam a obtenção da solução ótima, desempenham um papel crucial na resolução de problemas complexos, como o problema de empacotamento bin. Eles fazem uso de heurísticas para encontrar soluções que se aproximem do ótimo em tempo polinomial, fornecendo uma garantia de resultado através do fator de aproximação. No contexto do empacotamento, essas heurísticas podem ser categorizadas em duas abordagens: heurísticas online e heurísticas offline.

Na categoria das heurísticas online, o algoritmo toma decisões imediatas, muitas vezes em tempo real, à medida que recebe cada item sem conhecimento prévio sobre o tamanho ou a ordem dos itens subsequentes. Por outro lado, as heurísticas offline têm acesso completo a informações sobre todos os itens antes de tomar qualquer decisão. Elas conhecem antecipadamente o tamanho e a ordem de todos os itens. No contexto deste relatório, foram desenvolvidos diversos algoritmos aproximativos (utilizando tanto heurísticas online como offline) para abordar o problema..

Além disso, levando em consideração o volume de dados obtidos e visando automatizar a geração dos resultados, o grupo desenvolveu também um script em Python utilizando a biblioteca de geração de gráficos matplotlib (pode ser encontrado na pasta

‘src/scripts-analysis/’ do github) que cria gráficos de barras empilhadas para representar visualmente a distribuição de itens em caixas para o problema de bin packing.

A seguir, forneceremos breves descrições de cada um desses algoritmos, descreveremos como foi realizada a implementação e por fim um exemplo gráfico da aplicação dos algoritmos para caixas de no máximo tamanho 80.

Entrada: [26, 57, 18, 8, 45, 16, 22, 29, 5, 11, 8, 27, 54, 13, 17, 21, 63, 14, 16, 45, 6, 32, 57, 24, 18, 27, 54, 35, 12, 43, 36, 72, 14, 28, 3, 11, 46, 27, 42, 59, 26, 41, 15, 41, 68]

Next Fit (NF): Itera sobre os itens na ordem em que são dados. Cada item é colocado na caixa atual se couber, caso contrário, uma nova caixa é usada.

- Complexidade, onde n é o número de itens: O(n)
- Fator de Aproximação: 2 aproximado (prova do fator de aproximação no link)

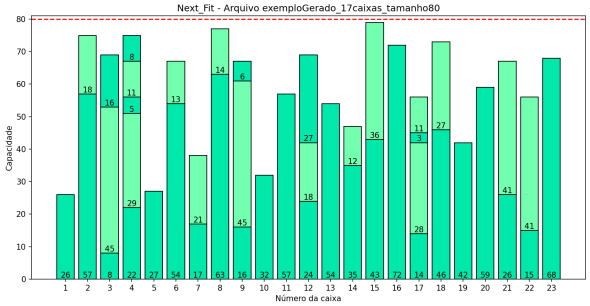

Fig. 2. Gráfico - Next Fit

Next Fit Decreasing (NFD): Ordenar os itens do maior para o menor (decrescente). Depois aplicar o algoritmo Next-Fit

- Complexidade: O(n log n)
- Fator de Aproximação: 1.691 aproximado

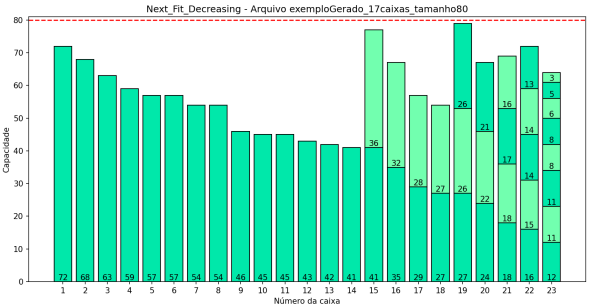

Fig. 3. Gráfico - Next Fit Decreasing

Best Fit (BF): Se o item couber em alguma caixa aberta, colocar na que sobra menos espaço. Caso contrário, abre uma nova caixa

- Complexidade: $O(n^2)$*.
- Fator de Aproximação: 1.7 aproximado

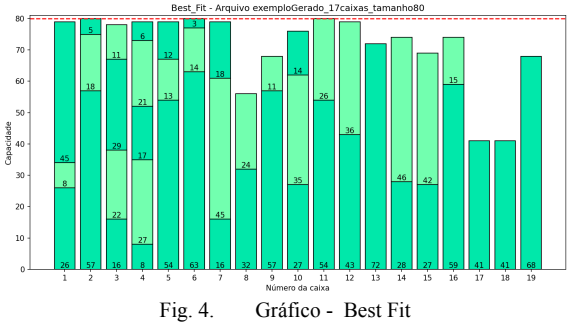

Fig. 4. Gráfico - Best Fit

[1] *pode ser implementado em tempo O(n Log n) usando Árvores de Pesquisa Binária com Auto-Balanceamento

Best Fit Decreasing (BFD): Ordenar os itens do maior para o menor (decrescente). Depois aplicar o algoritmo Best-Fit

- Complexidade: $O(n^2)$*
- Fator de Aproximação: 1.222 aproximado

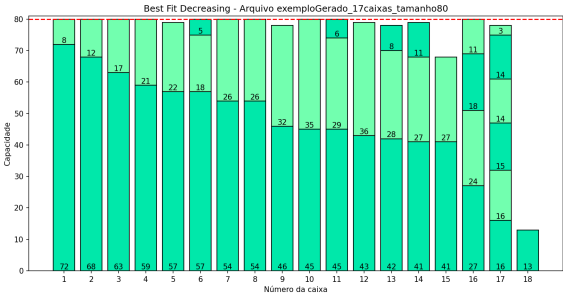

Fig. 5. Gráfico - Best Fit Decreasing

First Fit (FF): Colocar o item na primeira caixa na qual o item cabe. Se não cabe em nenhuma, abre uma nova caixa

- Complexidade: $O(n^2)$*
- Fator de Aproximação: 1.7 aproximado

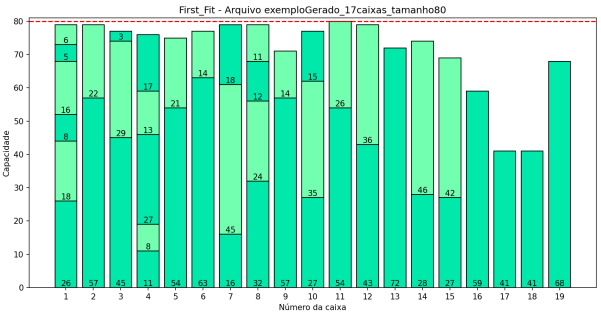

Fig. 6. Gráfico - First Fit

First Fit Decreasing (FFD): Ordenar os itens do maior para o menor (decrescente). Depois aplicar o algoritmo First-Fit

- Complexidade: $O(n^2)$*
- Fator de Aproximação: 1.222 aproximado

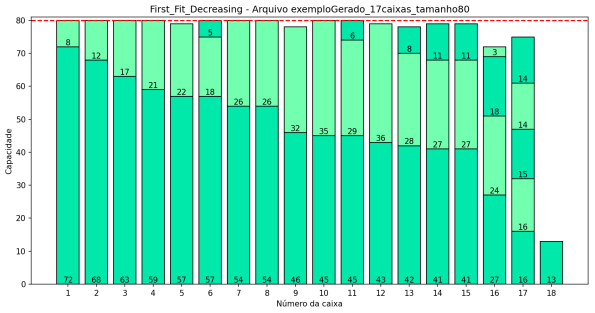

Fig. 7. Gráfico - First Fit Decreasing

Modified First Fit Decreasing (MFFD):

1. Organize os itens em caixas de diferentes tamanhos, da maior para a menor: Grande (1/2, 1], Médio (1/3, 1/2], Pequeno (1/6, 1/3], Minúsculo (0,1/6].
2. Comece pela maior caixa e insira o maior item médio que caiba. Pule a caixa se não couber.
3. Volte pelas caixas sem itens médios e insira o menor e o maior item que caibam, pulando a caixa se ambos não couberem.
4. Continue pelas caixas, inserindo o item maior que couber para cada classe de tamanho, pulando se não couber.
5. Use o método FFD para embalar os itens restantes em novas caixas.

- Complexidade: $O(n^2)$*
- Fator de Aproximação: 1.183 aproximado

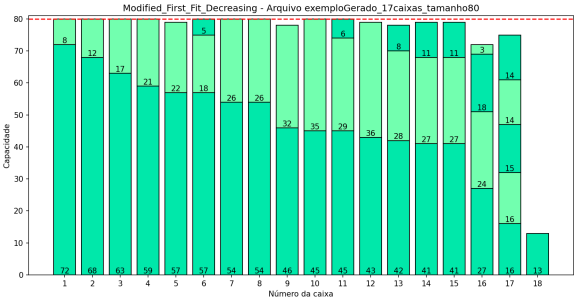

Fig. 8. Gráfico - Modified First Fit Decreasing

A linguagem escolhida para desenvolver os algoritmos foi Java, sendo que os códigos estão disponíveis no repositório do github. O programa pode receber argumentos da linha de comando, incluindo o tamanho da caixa, o algoritmo a ser usado e o nome do arquivo de entrada. Ele permite que o usuário escolha entre várias opções, como exibir a lista de itens, alterar o tamanho da caixa e executar diferentes algoritmos de empacotamento. Os algoritmos de empacotamento são executados com base nas escolhas do usuário e os resultados são exibidos. O programa também inclui funcionalidades adicionais, como a capacidade de gerar a média do tempo de execução dos algoritmos, para isso um grande número de repetições é realizado quando escolhida essa opção. Além disso, há um exemplo prático de aplicação real do problema de empacotamento, que será discutido posteriormente.

Para isso, foi desenvolvido uma classe para cada um dos algoritmos, com suas implementações específicas de manipulação do vetor de itens. Uma classe mais geral chamada 'Packing', é usada para representar o resultado do empacotamento de itens em caixas e as principais manipulações que podem se realizar. Essa classe possui métodos para adicionar itens às caixas, calcular várias estatísticas relacionadas ao empacotamento e exportar os resultados para um arquivo CSV.

IV. ANÁLISES

Geração dos Datasets:

O grupo primeiramente utilizou alguns exemplos pequenos encontrados na internet para validar se o resultado obtido pelo algoritmo exato estava correto, também sendo utilizados como teste durante a construção dos algoritmos aproximativos. Depois de desenvolvidos os algoritmos foram criados datasets maiores para aumentar a complexidade dos testes, fizemos da seguinte maneira: estabelecemos um limite máximo para as caixas, colocamos dados aleatórios dentro delas de forma que a soma não ultrapassasse o limite, para fazer isso utilizamos uma planilha para organizar os dados. Nesse ponto sabemos qual o resultado ótimo (número de caixas mínimo) e depois embaralhamos estes dados. No total criamos oito datasets, no qual os dados gerados com seus respectivos resultados podem ser encontrados na pasta '/src/dataset' no github.

Análise de resultado e tempo de execução:

Vale destacar, que o grupo se preocupou em realizar várias repetições para cada um dos testes nos algoritmos aproximativos. No total foram executadas 1000 repetições de forma a obter resultados de tempo de simulação mais exatos. O grupo também criou uma planilha com todas informações das simulações de cada algoritmo, com informações de cada experimento como o nome do arquivo em que se encontra nos testes, tamanho da caixa, número de itens, peso total de todos itens, algoritmo aplicado, tempo de execução, a solução obtida, porcentagem de uso geral das caixas e a solução ótima. Entretanto, devido a extensão dessa planilha, não foi possível mostrar ela neste relatório, mas pode ser encontrada aqui.

A Partir dos dados da planilha, foi gerado o gráfico abaixo usando um script desenvolvido em python, sendo possível visualizar os resultados de tempo de simulação para os sete algoritmos desenvolvidos e os 8 diferentes datasets.

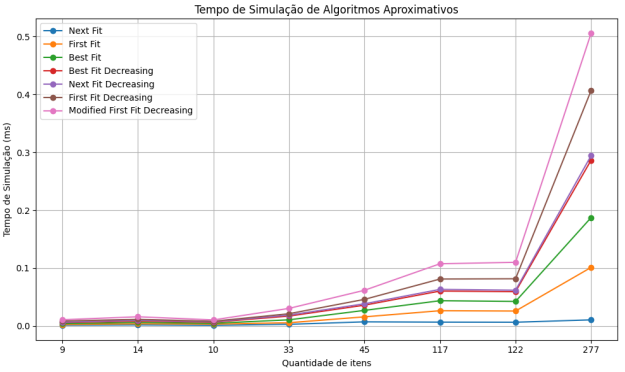

Fig. 9. Gráfico do tempo de simulação dos algoritmos aproximativos

Uma observação inicial revela que os algoritmos de empacotamento no geral alcançaram soluções obtidas idênticas às soluções ótimas conhecidas, sugerindo que esses algoritmos são bastantes eficazes. No entanto, também foram observados casos em que houve uma pequena diferença entre a solução obtida e a solução ótima. Essa diferença é mais evidente nos algoritmos que possuem um maior fator de aproximação, como o NF e NFD.

Além disso, foi evidente que os algoritmos tiveram melhor desempenho (tempo de execução) em cenários com quantidade de itens mais baixos, o que é uma observação esperada, considerando que resolver problemas mais desafiadores com um maior número de verificações e testes exigirá um maior esforço computacional. Outro ponto a se destacar é que o algoritmo NF foi o único que apresentou aparentemente um crescimento linear de tempo (não apresentando uma curva acentuada no gráfico), o que era esperado visto que sua complexidade é O(n).

Percebe-se também que a etapa adicional de ordenação aplicada nesses algoritmos desempenha um papel significativo na melhoria do desempenho dos mesmos em termos de encontrar soluções próximas à ótima. Ou seja, ao priorizar os itens maiores ou mais pesados no início do processo de empacotamento, esses algoritmos têm maior probabilidade de encontrar soluções próximas à ótima, uma vez que reduzem o espaço desperdiçado nas caixas.

A quantidade de repetições realizadas para cada algoritmo forneceram informações mais precisas sobre a eficiência computacional dos mesmos. Assim, percebe-se que os algoritmos offline se mostraram mais eficientes em termos de tempo de execução do que os algoritmos online, devido a etapa de ordenação adicional. Essa observação é crucial ao escolher um algoritmo para lidar com problemas que envolvam um grande número de itens.

Com base nos experimentos realizados com os conjuntos de dados "dataSet1_FSU_tamanho100", "dataSet3_FSU_tamanho100" e um conjunto de exemplo com caixas de tamanho 5 e os itens 1, 2, 3 e 4, foram obtidos resultados interessantes em relação ao desempenho dos algoritmos de empacotamento. Conseguimos chegar em

dois gráficos demonstrando a relação de caixas utilizadas e a porcentagem de uso médio das caixas na melhor solução encontrada por cada algoritmo.

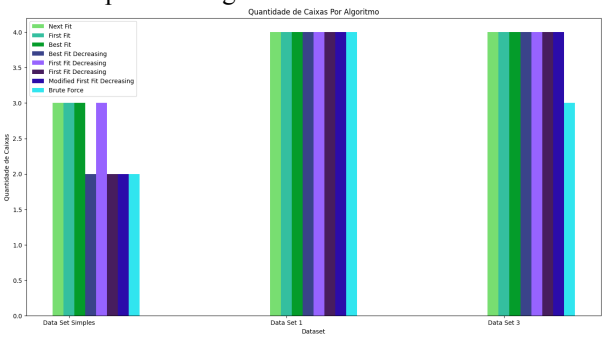

Fig. 10. Gráfico do quantidade de caixas por algoritmo na melhor solução

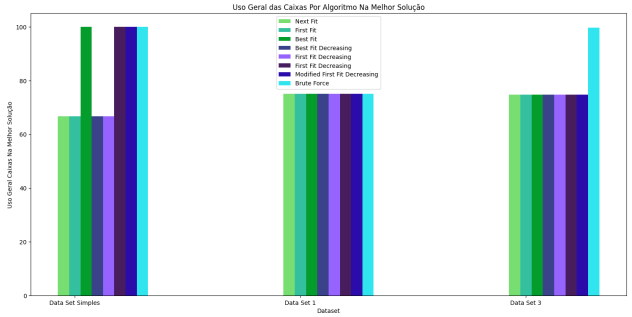

Fig. 11. Gráfico do uso médio das caixas na melhor solução encontrada por algoritmo

Para o gráfico de uso geral das caixas: no conjunto de dados "Data Set Simples", os algoritmos BF, FFD, MFFD e Brute Force obtiveram uma porcentagem de uso geral das caixas maiores. Isso indica que o espaço nas caixas foi mais bem utilizado por esses algoritmos, resultando em um menor desperdício de espaço. No conjunto de dados "Data Set 1", todos os algoritmos alcançaram uma utilização média das caixas igual, assim como no gráfico de quantidade de caixas para a melhor solução encontrada, que para todos os algoritmos foi 4 caixas. No conjunto de dados "Data Set 3", o único algoritmo que conseguiu usar 99.67% das caixas foi o algoritmo de força bruta. Todos os outros algoritmos resultaram em uma porcentagem de uso geral das caixas menor, indicando que o algoritmo de força bruta encontrou uma solução que empacotou os itens disponíveis de forma eficiente.

Para o gráfico de quantidade de caixas utilizadas por algoritmo na melhor solução encontrada: no conjunto de dados "Data Set Simples", os melhores algoritmos em termos de quantidade de caixas na melhor solução foram BFD, FFD, MFFD e o algoritmo de força bruta. Eles conseguiram encontrar a solução que usa o menor número de caixas possível, indicando uma eficiência no empacotamento. No conjunto de dados "Data Set 1", todos os algoritmos alcançaram a melhor solução possível, que consistia em utilizar 4 caixas. No conjunto de dados "Data Set 3", apenas o algoritmo de força bruta conseguiu encontrar a melhor solução, que consistia em utilizar 3 caixas. Todos os outros algoritmos resultaram em soluções que utilizaram 4 caixas.

Os experimentos demonstraram que a escolha do algoritmo de empacotamento adequado depende das características específicas do problema. Algoritmos como NF e FF mostraram ser eficazes em muitos cenários, enquanto BF e BFD também tiveram bom desempenho. MFFD também se destacou em vários experimentos. A análise cuidadosa das características do problema, como o tamanho da caixa, o número de itens, a disposição e os pesos desses itens, é essencial para selecionar o algoritmo mais apropriado.

V. APLICAÇÃO REAL

A aplicação escolhida para o problema do empacotamento de caixas consiste na visualização de filmes, onde o objetivo é determinar a melhor maneira de assistir a todos os filmes de uma lista em um número mínimo de dias, considerando a restrição de que não é legal parar um filme pela metade e tem disponíveis 6 horas por dia para assistir. Os filmes escolhidos para simulação foram da produtora Marvel, totalizando 44 no total e podem ser consultados na pasta *src/dataSet/filmesMarvel.csv* no repositório do github.

Nesse arquivo são colocados primeiramente o nome do filme e logo em seguida a duração do mesmo em minutos. A ‘caixa’ é representada pelo tempo disponível por dia para assistir esses filmes, na simulação foi determinado o limite máximo de 6 horas por dia. Para esses filmes o número mínimo de dias para concluir o objetivo seria 23. Em resumo, esta classe lida com a leitura de dados de filmes a partir de um arquivo CSV, utiliza um algoritmo de bin packing (podendo ser escolhido pelo usuário qual o algoritmo a ser selecionado) para determinar a melhor ordem de assistir aos filmes em dias com duração limitada, e, finalmente, exporta os resultados para um arquivo.

Neste relatório, exploramos o desafio do Problema de Empacotamento de Caixas, revelando a impraticabilidade do algoritmo exato para conjuntos grandes e destacando a eficácia de algoritmos aproximativos, além de aplicá-los à otimização de visualização de filmes no mundo real.

REFERÊNCIAS

JOHNSON, D. S. Fast algorithms for bin packing. Journal of Computer and System Sciences, v. 8, n. 3, p. 272 – 314, 1974

David S. Johnson, Michael R. Garey, "A 7160 theorem for bin packing," Journal of Complexity, Volume 1, Issue 1, 1985, Pages 65-106, ISSN 0885-064X, https://doi.org/10.1016/0885-064X(85)90022-6

JOHNSON, D. S.; ULLMAN, J. D.; GAREYI, M. R.; GRAHAMII, R. L. Worst-case performance bounds for simple one-dimensional packing algorithms.

JOHNSON, D. S. Near-optimal bin packing algorithms. 1973. Tese (Doutorado em Fíısica) - Massachusetts Institute of Technology, 1973.

W. T. Rhee and M. Talagrand, "The Complete Convergence of Best Fit Decreasing," in SIAM Journal on Computing, vol. 18, no. 5, pp. 909-918, 1989. doi: 10.1137/0218063.

Dósa, G., Sgall, J. (2014). Optimal Analysis of Best Fit Bin Packing. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds) Automata, Languages, and Programming. ICALP 2014. Lecture Notes in Computer Science, vol 8572. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-43948-7_36.

Baker, B. S.; Coffman, Jr., E. G. (1981-06-01). "A Tight Asymptotic Bound for Next-Fit-Decreasing Bin-Packing". SIAM Journal on Algebraic and Discrete Methods. 2 (2): 147–152. doi:10.1137/0602019. ISSN 0196-5212 .

Saraiva, R. V., & Schouery, R. C. S. (2020). Algoritmos de Aproximação para o Problema do Empacotamento (Technical Report No. IC-20-04 - Relatório Técnico). Universidade Estadual de Campinas, Instituto de Computação. URL: https://www.ic.unicamp.br/~reltech/2020/20-04.pdf

Mitch. Bin Packing. 31 Mar 2006 URL: https://www.developerfusion.com/article/5540/bin-packing/