# Problema do Corte Bidimensional Não-Guilhotinado solucionado pela Meta-Heurística *Simulated Annealing*

**Lívia de Azevedo da Silva**
Graduanda em Ciência da Computação - Universidade Federal Rural do Rio de Janeiro
R. Governador Roberto Silveira S/N – Nova Iguaçu – Rio de Janeiro – RJ – Brasil
livia_de_azevedo@ufrrj.br

**Hosana Gomes Pinto**
Graduanda em Ciência da Computação - Universidade Federal Rural do Rio de Janeiro
R. Governador Roberto Silveira S/N – Nova Iguaçu – Rio de Janeiro – RJ – Brasil
hosanagomes@ufrrj.br

**RESUMO**

Este trabalho tem por objetivo apresentar um estudo feito sobre corte bidimensional não guilhotinado, cujo objetivo é, dado um conjunto de peças com variados tamanhos, cortá-las sobre um retângulo de forma a maximizar o lucro a ser obtido na venda. Para isso, foi utilizada a meta-heurística *Simulated Annealing* na construção do algoritmo proposto. Essa técnica se baseia na geração de novas soluções através de um mecanismo de desmantelamento parcial de soluções antigas, o que varia o espaço de busca. Na fase de testes, foram usadas instâncias as quais eram conhecidos seus valores ótimos e os resultados obtidos neste estudo se mostraram bons na comparação com esses valores.

**PALAVRAS CHAVE. Corte não-guilhotinado, Meta heurística, *Simulated Annealing*.**

**ABSTRACT**

This paper aims to present a study of a non-guillotine two-dimensional cutting, whose objective is, given a set of pieces with different sizes, cut them on a rectangle in order to maximize the profit to be obtained in the sale. For this, the *Simulated Annealing* metaheuristic was used in the construction of the proposed algorithm. This technique is based on the generation of new solutions through a partial dismantling mechanism of old solutions, which varies the search space. In the test phase, instances were used whose optimal values were known and the results obtained in this study were good in comparison with these values.

**KEYWORDS. Non-guillotine cutting. Metaheuristic. *Simulated Annealing*.**

## 1. Introdução

O problema do corte bidimensional não guilhotinado tem por objetivo, dado um conjunto de peças com variados tamanhos, cortá-las sobre um retângulo de forma a maximizar o lucro a ser obtido na venda. O problema tem aplicação em vários processos de produção em indústrias têxtil, papel, metal, vidro e madeira [Alvarez-Valdes et al., 2007].

Esse problema pode ser definido da seguinte forma: um retângulo grande $R = (L, W)$ tem comprimento $L$ e largura $W$ . As peças são os retângulos menores, isto é, os moldes que demarcam como $R$ será recortado. Cada peça $i$ do conjunto de instâncias $S$ tem três atributos $i = (l_i, w_i, v_i)$, os quais $l_i$ representa o comprimento da peça, $w_i$ a largura e $v_i$ o valor da peça.

Há a possibilidade de atacar este problema levando em consideração o objetivo de diminuir o desperdício do material a ser cortado, considerando o valor de cada peça sendo equivalente a área que a mesma possui [Bortfeldt e Winter, 2009].

É estabelecido que as peças não são rotacionáveis e são posicionadas paralelamente às extremidades de $R$. Isso significa que posicionar as peças diagonalmente ou, simplesmente, girá-las a 90 graus são movimentos proibidos na construção desse problema. O retângulo $R$ deve ser cortado em $x_i$ peças menores. O objetivo, então, é maximizar $\sum_i v_i x_i$.

Além disso é postulado que, para cada peça $i$, existe um limite inferior $P_i$ e um limite superior $Q_i$ que representam, respectivamente, as quantidades mínima e máxima que devem existir dessa peça em $R$. Neste trabalho, $\forall i \in S, P_i = 0$, onde $S$ é o conjunto de peças disponíveis, e o valor de $Qi < \lfloor L * W \ / \ li * wi \rfloor$; atribuído conforme fornecido por cada peça i, o que configura um problema restrito [Alvarez-Valdes et al., 2005]. A Figura 1 mostra um exemplo de um retângulo $R$ cortado nessa definição. Vale ressaltar que a largura é definida verticalmente, enquanto o comprimento é definido horizontalmente. O valor de solução é obtido por $\sum_i v_i x_i, \forall i \in R$.

Tabela 1: Conjunto de peças disponíveis do exemplo.

| Estoque $R = (L, W)$ | | | | | |
|---|---|---|---|---|---|
| Peça | $l_i$ | $w_i$ | $P_i$ | $Q_i$ | $v_i$ |
| 1 | 3 | 2 | 0 | 2 | 7 |
| 2 | 7 | 2 | 0 | 3 | 20 |
| 3 | 4 | 2 | 0 | 2 | 11 |
| 4 | 6 | 2 | 0 | 3 | 13 |
| 5 | 9 | 1 | 0 | 2 | 21 |
| 6 | 8 | 4 | 0 | 1 | 79 |
| 7 | 4 | 1 | 0 | 2 | 9 |
| 8 | 1 | 10 | 0 | 1 | 14 |
| 9 | 3 | 7 | 0 | 3 | 52 |
| 10 | 4 | 5 | 0 | 2 | 60 |

No trabalho de [Amaral e Mariano, 2015], além do problema restrito, existe um detalhamento do que são os problemas irrestrito e o duplamente restrito. Como o escopo deste estudo se limita apenas ao problema restrito, as demais definições foram suprimidas aqui. Os problemas restritos são mais interessantes de serem estudados pois, em um problema real, é inevitável que, por questões comerciais, haja um limitante máximo para o número de peças a serem cortadas.

Na próxima seção será apresentada a heurística construtiva de solução para o problema. Na Seção 3, a descrição da implementação do *Simulated Annealing*. Na Seção 4, os testes computacionais e os resultados. Na última seção, a conclusão.

## 2. Ideia Básica de Construção de uma Solução

Na construção de uma solução são usadas duas listas ($P$ e $B$) e o retângulo $R$, no algoritmo aqui presente representada como uma matriz $L$ X $W$. Cada retângulo(ou solução) $R$ existem

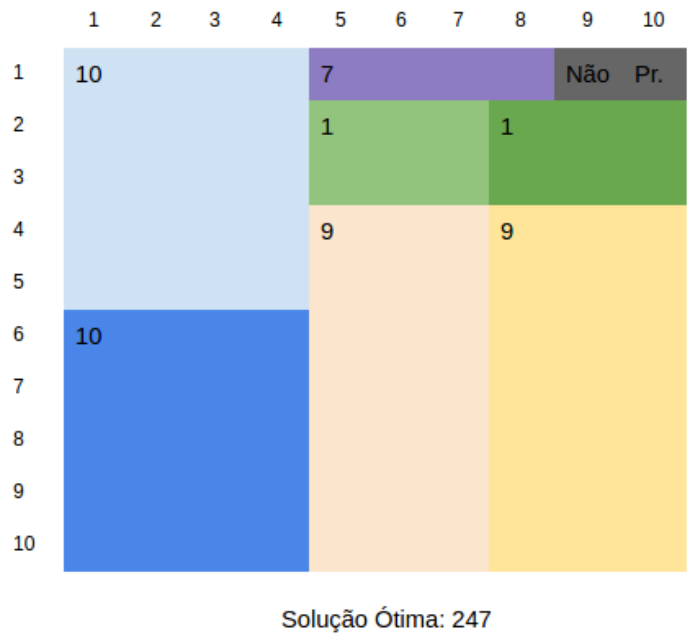


Figura 1: Exemplo de solução para o problema de corte (adaptado de [Amaral e Mariano, 2015]).

as listas $P$ e $B$ atreladas a ele. A lista $P$ contém todas as peças disponíveis que podem ser cortadas em $R$ e suas respectivas quantidades, enquanto $B$, inicialmente vazia, armazenará todas as peças já contidas em $R$. Para cada peça em $P$, o algoritmo testa se é possível adicioná-la na origem $O$ = (0,0) de $R$. Se não for possível, o algoritmo adiciona a peça no primeiro ponto factível, isto é, no ponto mais próximo da origem que não permita a sobreposição da nova peça a alguma já presente no retângulo $R$. Com o objetivo de realizar isso, existem dois fatos a serem considerados:

1. Cada peça $i$ possui dois pontos candidatos, sendo primeiro o mais à direita e mais acima $P1_i = (x_1, y_1)_i$ e o segundo ponto mais à esquerda e mais abaixo $P2_i = (x_2, y_2)_i$. A Figura 2 ilustra essa noção.

2. A determinação do espaço mais próximo à origem é feita a partir do cálculo da distância euclidiana de cada ponto candidato de cada peça já presente em $B$ com a origem.

O ponto candidato $P1_i$ recebe uma prioridade de escolha durante a seleção de primeiro ponto candidato factível. Isto se deve para tentar ocupar $R$ com as peças ao máximo.

Quando esse espaço é localizado, a peça é decrementada de $P$, inserida em $R$ e adicionada em $B$, tendo sido atualizado os seus pontos candidatos em $R$. Se a peça for a última do seu tipo em $P$, ela é eliminada da lista, ao invés de ser apenas decrementada. O algoritmo procura adicionar primeiramente as peças de mesmo tipo antes de dar prosseguimento para os outros tipos presentes em $P$. O procedimento de construção da solução é finalizado quando não é mais possível adicionar nenhuma peça contida em $P$ ou $P$ está vazia. O Algoritmo 1 fornece o pseudocódigo deste algoritmo.

## 3. *Simulated Annealing* - Algoritmo

O algoritmo do *Simulated Annealing* foi inicialmente proposto por [Kirkpatrick et al., 1983] tem por base o processo físico de recozimento de metais. O algoritmo inicia com uma temperatura inicial alta e, conforme as soluções intermediárias são obtidas, esta temperatura é resfriada a uma certa taxa até obter o seu estado de congelamento, que determina o fim do algoritmo. O *Simulated Annealing* tem um diferencial de poder aceitar com uma certa probabilidade soluções que são piores do que a atual para serem consideradas no cálculo do próximo vizinho. Para a utilização desta meta heurística, há a necessidade de determinar os seguintes fatores:

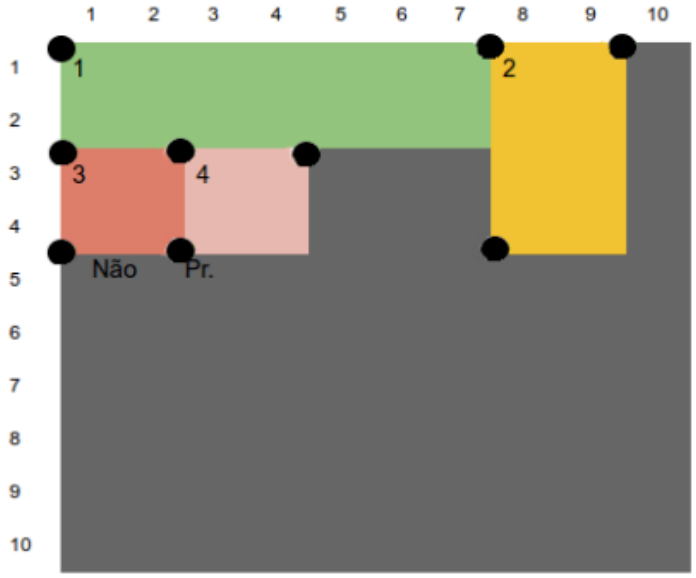


Figura 2: Exemplo de construção de uma solução com os pontos candidatos em $R$.

**Algoritmo 1** Criar Solução

```
1:  função CRIARSOLUÇÃO(Solução nova)
2:      listaAuxiliar ← (cópia da lista nova.P)
3:      pecaNova ← (primeiro nó de listaAuxiliar)
4:      enquanto ¬chegada(listaAuxiliar) faça
5:          contador ← 0
6:          qtdInicial ← pecaNova.quantidade
7:          enquanto contador < qtdInicial faça
8:              Tentar adicionar pecaNova ao ponto de origem de R ou, para cada peça em nova.B,
9:              tentar adicionar a pecaNova nos pontos candidatos.
10:             se se foi adicionada então
11:                 removerPecaDadoPeca(nova.P, novaPeca)
12:                 inserirPeca(nova.B, pecaNova)
13:             fim se
14:             contador ← contador + 1
15:         fim enquanto
16:     fim enquanto
17:     retorna nova
18: fim função
```

1. Uma temperatura inicial (valor suficientemente alto);

2. O estado de equilíbrio (quantas vezes o algoritmo irá iterar em uma determinada temperatura);

3. Seu estado de congelamento (critério de parada);

4. Taxa de resfriamento (a velocidade de diminuição da temperatura);

5. Uma heurística para a determinação do vizinho de uma solução (definição do espaço de busca).

A heurística para determinação do vizinho de uma solução específica, chamada de *trocarSolucao()*, é definida pelo Algoritmo 2. Esta função tem como base a ideia de destruição e reconstrução de uma solução tendo como referência a disposição das peças presentes atualmente em $R$. Assim, inicialmente, o algoritmo seleciona aleatoriamente peças contidas em $R$ até que se chegue a uma determinada porcentagem da área de $R$ a ser removida, processo que ocorre dentro da função *remocaoAleatoria()*.

No final deste passo existirá uma alta probabilidade da disposição das peças em $R$ gerar uma solução esparsa, isto é, com espaços pequenos disponíveis, o que poderia dificultar a inserção de novas peças futuramente. Para resolver esta situação, o algoritmo realiza uma compressão destas peças, concentrando-as de forma a eliminar ao máximo esta configuração esparsa. Para isso, as peças são ordenadas em ordem crescente de acordo com a distância de seus pontos de origem(localizado mais a esquerda e mais acima) ao ponto de origem $O = (0, 0)$ de $R$ e depois, seguindo esta ordem, a primeira peça é deslocada em direção ao ponto $O$. As demais são deslocadas em direção ao ponto candidato factível(que seja possível inserir a peça) mais próximo da origem de $R$ dentre as peças que já foram deslocadas anteriormente neste mesmo processo. A função que computa este método é denominada no algoritmo como *deslocarPecas()*.

Depois destas operações, é feita a chamada da função *criarSolucao()*, explicada anteriormente, para inserir novas peças em $R$ e assim finalizar a construção de uma nova solução.

**Algoritmo 2** Trocar Solução

```
1: função TROCARSOLUÇÃO(Solução atual, γ)
2:     remocaoAleatoria(atual,γ)
3:     deslocarPecas(atual)
4:     criarSolucao(atual)
5:     retorna atual
6: fim função
```

Por fim, o Algoritmo 3 representa o processo do *Simulated Annealing* em si. Uma informação importante é que a lista $P$, antes de ser criada a primeira solução para iniciar o *Simulated Annealing*, será ordenada em ordem decrescente de acordo com o custo benefício de cada tipo de peça, que é dada por $\frac{v_i}{l_i * w_i}$, caracterizando uma solução inicial construída baseada em uma estratégia gulosa.

## 4. Testes Computacionais

Para realizar os experimentos desta abordagem foram utilizados as mesmas 21 instâncias da literatura disponíveis no link que segue a referência [Beasley, 2016]. Estas instâncias também foram utilizadas por [Amaral e Mariano, 2015]. O objetivo deste trabalho é comparar os resultados que estes autores obtiveram com os resultados aqui obtidos, levando em consideração que o algoritmo aqui construído teve como base o algoritmo feito por eles.

Portanto, cada instância o algoritmo foi executado 10 vezes e o resultado final é a média aritmética entre eles, o mesmo método usado pelos autores os quais teve-se base. A Tabela 3 mostra estes resultados. A tabela fornece as características de cada caso de teste juntamente com o resultado final, o melhor valor encontrado e o desvio padrão dos 10 resultados encontrados para cada um dos algoritmos. A variável $SA_{AM}$ representa os resultados de [Amaral e Mariano, 2015] e $SA_{AB}$ a atual abordagem deste artigo. A variável $T_p$ é referente a quantidade de tipos de peças existentes em cada instância.

A Tabela 4 mostra o tempo médio de execução, em segundos, da abordagem $SA_{AB}$ para cada instância. Para obter esta média, o algoritmo foi executado 10 vezes e logo após calculada sua média. Não há uma comparação direta entre $SA_{AB}$ e $SA_{AM}$ pois os algoritmos foram efetivados em duas máquinas distintas.

**Algoritmo 3** *Simulated Annealing* para o problema de corte

**função** SA($T$, $T_c$, $It_{max}$, $\alpha$, $\gamma$)
    Inicializa *atual*.$P$ com as peças do conjunto considerado
    Ordenar *atual*.$P$ em ordem decrescente de $\frac{v_i}{l_i * w_i}$
    *atual* $\leftarrow$ criarSolução(*atual*)
    *melhor* $\leftarrow$ (cópia de *atual*)
    *iter* $\leftarrow 0$
    **enquanto** $T > T_c$ **faça**
        **enquanto** *iter* $< It_{max}$ **faça**
            *nova* $\leftarrow$ trocarSolução(*atual*, $\gamma$)
            $\Delta \leftarrow$ *atual*.valor - *nova*.valor
            **se** $\Delta < 0$ **então**
                *atual* $\leftarrow$ *nova*
                **se** *nova*.valor $>$ *melhor*.valor **então**
                    *melhor* $\leftarrow$ *nova*
                **senão**
                    Considerar aleatoriamente um valor $x \in [0, 1]$
                    **se** $x < e^{\frac{-\Delta}{T}}$ **então**
                        *atual* $\leftarrow$ *nova*
                    **fim se**
                **fim se**
            **fim se**
            *iter* $\leftarrow$ *iter* + 1
        **fim enquanto**
        $T \leftarrow \alpha * T$
        *iter* $\leftarrow 0$
    **fim enquanto**
    **retorna** *melhor*
**fim função**

Cada experimento foi feito em uma máquina com um processador Intel Core i5-3230M com 2.60 GHz, 8GB de memória RAM e um sistema operacional Ubuntu 14.04 LTS. O programa foi escrito em linguagem C e executado no compilador *GNU Compiler Collection* (GCC), versão 4.8.4.

Os quatro parâmetros do *Simulated Annealing* e a porcentagem de remoção de uma solução são descritos na Tabela 2. Todos foram obtidos empiricamente, selecionando a instância de número 21 e executando o algoritmo uma vez para os diferentes intervalos de valores definidos para cada parâmetro. Os parâmetros com melhores resultados foram considerados no algortimo. Os parâmetros aqui aplicados foram diferentes dos usados no trabalho de [Amaral e Mariano, 2015].

Tabela 2: Descrição e parâmetros adotados no *Simulated Annealing*

| **Parâmetro** | **Descrição** | **Valor** |
|---|---|---|
| $T$ | Temperatura inicial | 6000 |
| $T_c$ | Temperatura de congelamento | 0,4 |
| $It_{max}$ | Número máximo de iterações | 100 |
| $\alpha$ | Taxa de resfriamento | 0,98 |
| $\gamma$ | Percentual de remoção da solução | 0,35 |

Tabela 3: Resultados obtidos

| | | | | | $SA_{AM}$ | | | $SA_{AB}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | $L$ | $W$ | $T_p$ | Ótimo | Melhor | Média | D.P | Melhor | Média | D.P |
| 1 | 10 | 10 | 5 | 164 | **164** | 164 | 0 | **164** | 164 | 0 |
| 2 | 10 | 10 | 7 | 230 | **230** | 230 | 0 | **230** | 230 | 0 |
| 3 | 10 | 10 | 10 | 247 | **247** | 247 | 0 | **247** | 247 | 0 |
| 4 | 15 | 10 | 5 | 268 | **268** | 268 | 0 | **268** | 268 | 0 |
| 5 | 15 | 10 | 7 | 358 | **358** | 358 | 0 | **358** | 358 | 0 |
| 6 | 15 | 10 | 10 | 289 | **289** | 289 | 0 | **289** | 289 | 0 |
| 7 | 20 | 20 | 5 | 430 | **430** | 430 | 0 | **430** | 430 | 0 |
| 8 | 20 | 20 | 7 | 834 | **834** | 834 | 0 | **834** | 834 | 0 |
| 9 | 20 | 20 | 10 | 924 | **924** | 924 | 0 | **924** | 924 | 0 |
| 10 | 30 | 30 | 5 | 1452 | **1452** | 1452 | 0 | **1452** | 1452 | 0 |
| 11 | 30 | 30 | 7 | 1688 | **1688** | 1688 | 0 | **1688** | 1688 | 0 |
| 12 | 30 | 30 | 10 | 1865 | **1865** | 1865 | 0 | **1865** | 1865 | 0 |
| 13 | 30 | 30 | 7 | 1178 | **1178** | 1178 | 0 | **1178** | 1178 | 0 |
| 14 | 30 | 30 | 15 | 1270 | **1270** | 1270 | 0 | **1270** | 1270 | 0 |
| 15 | 70 | 40 | 19 | 2726 | 2716 | 2715 | 2,11 | 2721 | 2721 | 0 |
| 16 | 40 | 70 | 20 | 1860 | 1840 | 1826 | 9,66 | 1840 | 1822 | 6,324 |
| 17 | 100 | 100 | 15 | 27718 | **27718** | 27596,9 | 45,36 | 27589 | 27506,1 | 67,235 |
| 18 | 100 | 100 | 30 | 22502 | **22502** | 22028,6 | 166,33 | 21976 | 21624,8 | 402,587 |
| 19 | 100 | 100 | 30 | 24019 | 23789 | 23747 | 14,81 | 23655 | 23208,4 | 222,386 |
| 20 | 100 | 100 | 33 | 32893 | **32893** | 32893 | 0 | **32893** | 32525,6 | 426,161 |
| 21 | 100 | 100 | 29 | 27923 | **27923** | 27923 | 0 | 26380 | 26218,7 | 68,78 |
| Nº de Sol. Ótim. | | | | | 18 | | | 15 | | |

Observa-se que $SA_{AB}$ conseguiu alcançar o valor ótimo pelo menos uma vez em 15 instâncias, 3 a menos do que $SA_{AM}$. Em comparação ao melhor obtido em $SA_{AB}$ com $SA_{AM}$ percebe-se que a diferença entre os resultados não são muito distantes, com uma maior diferença para a instância 21, quantificado em 5,52%. Além disso, a variância de resultados para $SA_{AB}$ só existiu a partir da 16, quando o problema começou a aumentar a sua complexidade.

## 5. Conclusão

Neste artigo trabalhou-se com o problema de corte bidimensional não-guilhotinado. A proposta teve como base a meta heurística *Simulated Annealing* com uma estratégia de remoção, realocação e adição de peças em uma solução $R$ para definir o espaço de busca de uma solução. Aqui comparou-se dois algoritmos: $SA_{AM}$ e o próprio ($SA_{AB}$), concluindo que, dentre estes conjuntos de instâncias, o algoritmo de $SA_{AM}$ teve melhor desempenho que $SA_{AB}$. Mesmo assim, $SA_{AB}$ apresenta resultados competitivos e animadores, variando bastante seus valores (D.P) a medida que o tamanho das instâncias cresce e a disposição das peças torna o problema mais complexo, sendo possível afirmar que a estratégia alcançou as expectativas. Assim, é permitido considerar como um trabalho futuro a construção de novas estratégias de atacar o espaço de busca. Uma estratégia possível seria rotacionar uma peça a 90 graus caso aquele espaço não seja factível para a peça e inseri-la naquele espaço caso a rotação permita posicioná-la no local indicado. Como este exemplo, pode-se investigar outras tendências de soluções tendo como base as ideias implantadas aqui nesta abordagem.

Tabela 4: Tempo de execução

| | | | | Tempo |
|---|---|---|---|---|
| # | $L$ | $W$ | $T_p$ | $SA_{AB}$ |
| 1 | 10 | 10 | 5 | 0,704 |
| 2 | 10 | 10 | 7 | 1,244 |
| 3 | 10 | 10 | 10 | 1,543 |
| 4 | 15 | 10 | 5 | 0,605 |
| 5 | 15 | 10 | 7 | 0,992 |
| 6 | 15 | 10 | 10 | 1,180 |
| 7 | 20 | 20 | 5 | 0,955 |
| 8 | 20 | 20 | 7 | 1,171 |
| 9 | 20 | 20 | 10 | 1,534 |
| 10 | 30 | 30 | 5 | 1,195 |
| 11 | 30 | 30 | 7 | 1,474 |
| 12 | 30 | 30 | 10 | 2,241 |
| 13 | 30 | 30 | 7 | 0,674 |
| 14 | 30 | 30 | 15 | 1,283 |
| 15 | 70 | 40 | 19 | 4,150 |
| 16 | 40 | 70 | 20 | 4,679 |
| 17 | 100 | 100 | 15 | 14,067 |
| 18 | 100 | 100 | 30 | 4,883 |
| 19 | 100 | 100 | 30 | 4,947 |
| 20 | 100 | 100 | 33 | 11,475 |
| 21 | 100 | 100 | 29 | 28,678 |

**Referências**


Alvarez-Valdes, R., Parrenõ, F., e Tamarit, J. (2005). A grasp algorithm for constrained two-dimensional non-guillotine cutting problems. *Journal of Operational Research Society*, 56(4): 414–425.

Alvarez-Valdes, R., Parrenõ, F., e Tamarit, J. (2007). A tabu search algorithm for a two-dimensional non-guillotine cutting problem. *European Journal of Operational Research*, 183(3):1167–1182.

Amaral, A. R. S. e Mariano, G. P. (2015). Meta heurística simulated annealing aplicada ao problema de corte bidimensional naõ-guilhotinado. In *Anais do XLVII SBPO*, p. 2464–2472, Rio de Janeiro. SOBRAPO.

Beasley, J. E. (2016). Web page. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/ngcutinfo.html`. Acessado: 2016-01-29.

Bortfeldt, A. e Winter, T. (2009). A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces. *International Transactions in Operational Research*, 16:685–713.

Kirkpatrick, S., Gelatt, C. D., e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, New Series*, 220:671–680.