# META-HEURÍSTICA GRASP APLICADA AO PROBLEMA DO CORTE BIDIMENSIONAL NÃO-GUILHOTINADO

**Thiago Nascimento Rodrigues**
Universidade Federal do Espírito Santo
Av. Fernando Ferrari, 541, 29075-910, Vitória, ES, Brasil
nascimenthiago@gmail.com

**André Renato Sales Amaral**
Universidade Federal do Espírito Santo
Av. Fernando Ferrari, 541, 29075-910, Vitória, ES, Brasil
amaral@inf.ufes.br

**RESUMO**

Este trabalho trata do problema do corte bidimensional não-guilhotinado que visa a maximização do valor total de peças cortadas a partir de uma placa retangular. É proposto um algoritmo GRASP baseado na junção de diferentes estratégias para as fases construtiva e de busca local. Instâncias disponíveis na literatura são utilizadas para validação do algoritmo. Uma comparação com resultados obtidos por outros métodos da literatura é apresentada afim de atestar a qualidade da estratégia utilizada.

**PALAVRAS CHAVE. Meta-heurística, GRASP, Corte bidimensional não-guilhotinado.**

**Tópicos (MH - Metaheurísticas)**

**ABSTRACT**

This work deals with the two-dimensional non-guillotine cutting problem, which aims at maximizing the total value of pieces cut from a rectangular plate. A GRASP algorithm is proposed based on the union of different strategies for the constructive and local search phases. Instances available in the literature are used to validate the algorithm. A comparison with results obtained by other classic methods of the literature is presented in order to attest to the quality of the used strategy.

**KEYWORDS. Metaheuristic, GRASP, Two-dimensional non-guillotine cutting.**

**Paper topics (MH - Metaheuristics)**

## 1. Introdução

Em diversas aplicações relacionadas a processos industriais surgem problemas de relevante interessante para a Pesquisa Operacional. Uma classe conhecida destes problemas são os Problemas de Corte e Empacotamento (PCE). De maneira geral, em processos industriais que envolvam um PCE, é necessária a produção de determinadas quantidades de placas de um tipo de material, as quais devem, subsequentemente ser cortadas em peças de tamanho menor [Sepúlveda 2013]. Naturalmente, o corte de placas para a obtenção de produtos intermediários ou finais geralmente implica em perdas de matéria-prima. Logo, a diminuição dos gastos causados por estas perdas pode ser um fator decisivo para a garantia da competitividade de determinada indústria [Temponi 2007].

Nesse cenário, o *problema do corte bidimensional não-guilhotinado* consiste em obter um conjunto finito de pequenas peças retangulares a partir de cortes efetuados sobre um dado retângulo de estoque de dimensões fixas, visando maximizar o aproveitamento do retângulo a ser cortado. Este problema aparece em muitos processos de produção nas indústrias têxtil, de papel, aço, vidro e madeira. Mais formalmente, seja $R = (L, W)$ o retângulo de estoque de comprimento $L$ e de largura $W$. Cada peça $i$ tem dimensões $(l_i, w_i)$, e valor $v_i, i = 1, \ldots, m$. As peças têm orientações fixas e precisam ser cortadas de tal modo que suas faces fiquem paralelas às faces do retângulo de estoque (cortes ortogonais). O problema reside em cortar o retângulo $R$ em $x_i$ cópias de cada peça $i$, levando-se em conta que $0 \leq P_i \leq x_i \leq Q_i$, e de tal maneira que o valor total obtido a partir de cada peça cortada, ou seja $\sum_i v_i x_i$, seja maximizado. Nesse sentido, $M = \sum_i Q_i$ corresponde ao número máximo de peças que podem ser cortadas [Alvarez-Valdés et al. 2007].

*Retângulo de estoque* : $R = (10, 10)$

| *Peça* | $l_i$ | $w_i$ | $P_i$ | $Q_i$ | $v_i$ |
|---|---|---|---|---|---|
| 1 | 3 | 2 | 1 | 2 | 7 |
| 2 | 7 | 2 | 1 | 3 | 20 |
| 3 | 4 | 2 | 1 | 2 | 11 |
| 4 | 6 | 2 | 0 | 3 | 13 |
| 5 | 9 | 1 | 0 | 2 | 21 |
| 6 | 8 | 4 | 0 | 1 | 79 |
| 7 | 4 | 1 | 1 | 2 | 9 |
| 8 | 1 | 10 | 0 | 1 | 14 |
| 9 | 3 | 7 | 0 | 3 | 52 |
| 10 | 4 | 5 | 0 | 2 | 60 |

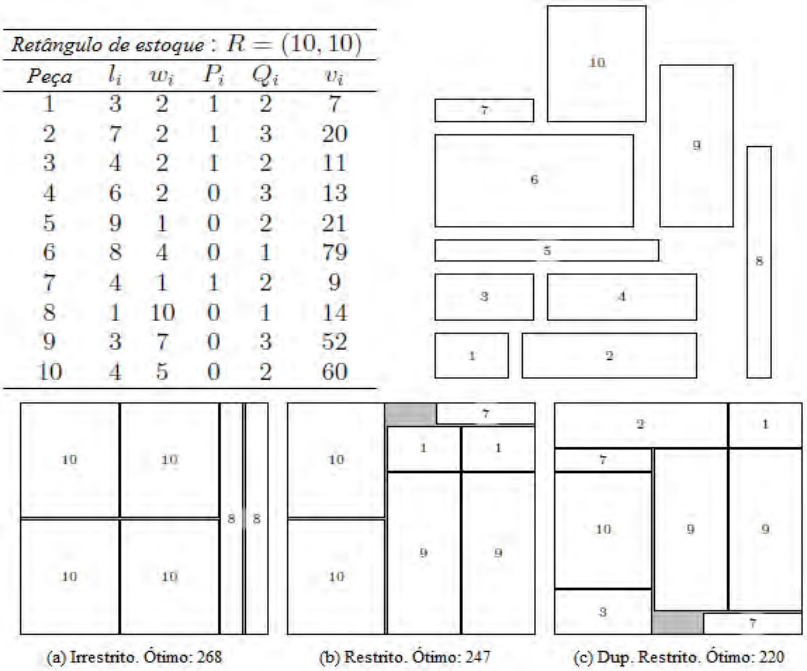


Figura 1: Exemplo de um Problema de Corte Bidimensional Não-guilhotinado adaptado de [Alvarez-Valdés et al. 2007].

Ainda confome [Alvarez-Valdés et al. 2007], de acordo com os valores de $P_i$ e $Q_i$ é possível distinguir-se três tipos de problemas:

1. *Irrestrito:* $\forall i, P_i = 0, Q_i = \lfloor L * W / l_i * w_i \rfloor$.
2. *Restrito:* $\forall i, P_i = 0; \exists i, Q_i < \lfloor L * W / l_i * w_i \rfloor$.
3. *Duplamente Restrito:* $\exists i, P_i > 0; \exists j, Q_j < \lfloor L * W / l_i * w_i \rfloor$.

A Figura 1 apresenta um exemplo de um Problema de Corte Bidimensional Não-guilhotinado tendo um retângulo de estoque $R = (10, 10)$, e $m = 10$ peças a serem cortadas. As três soluções (a), (b) e (c) representam, cada qual, a configuração ótima para os problemas irrestrito, restrito e duplamente restrito, respectivamente.

Por pertencerem à classe de problemas NP-Difícil, a maioria dos estudos sobre PCE se concentram em desenvolver métodos heurísticos para encontrar uma boa solução a um custo computacional razoável, ainda que sem a garantia de obtenção da solução ótima. Neste trabalho, propomos uma implementação da meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*) para o problema do corte bidimensional não-guilhotinado. As fases de construção de uma solução e de busca local do nosso algoritmo GRASP foram implementadas com inspiração no trabalho de [Coelho 2011] o qual apresentou um algoritmo GRASP para a resolução de outro problema (*Open Dimensional Problem*). Os resultados computacionais sobre um conjunto de 21 problemas da literatura, disponibilizados através da biblioteca *OR-Library* [Beasley 1990], e outro conjunto de 30 instâncias apresentadas por [Amaral 1999], mostraram que o algoritmo GRASP proposto é competitivo em relação a outros métodos.

Este artigo está organizado como segue: a seção 2 apresenta uma heurística para a fase de construção de soluções locais do algoritmo GRASP, assim como uma descrição da estratégia adotada para a exploração das vizinhanças das soluções obtidas na fase de construção. Os resultados obtidos pelos experimentos computacionais são analisados na seção 3. Finalizando na seção 4, as conclusões são apresentadas.

## 2. Um algoritmo GRASP

GRASP (*Greedy Randomized Adaptive Search*) é uma metaheurística iterativa e randômica na qual cada iteração provê uma solução para o problema em questão. Existem duas fases dentro de cada iteração GRASP: uma primeira fase construtiva, na qual uma solução inicial é construída de forma inteligente via alguma função randômica, adaptativa e gulosa; e uma segunda fase de melhoramento, onde um procedimento de busca local é aplicado sobre a solução construída com o intento de se agregar alguma melhora à solução [Feo e Resende 1995]. Neste trabalho, a fase de construção do algoritmo GRASP ficou a cargo do procedimento BuildLocalSolution descrito na seção 2.1.

### 2.1. Heurística Construtiva

No procedimento GRASP, a cada iteração, a construção de uma solução é realizada elemento a elemento e, posteriormente, uma busca local é realizada na vizinhança da solução construída, o que viabiliza o identificação de um ótimo local. O procedimento BuildLocalSolution, baseado no algoritmo apresentado por Alvarez-Valdes [Alvarez-Valdés et al. 2007] e explorado por Coelho [Coelho 2011], efetua a construção de uma solução local a partir de uma lista $\mathcal{P}$ de peças candidatas. A primeira ação acontece na linha 3 do procedimento, onde a lista $\mathcal{P}$ de peças é ordenada de forma descrescente em conformidade com um valor de peso determinado para cada elemento. Neste trabalho, uma função $g(\cdot)$ atribui a cada $p_i \in \mathcal{P}$ um valor dado por $g(p_i) = (l_i * w_i)/v_i$. Esta escolha constitui uma diferença em relação ao trabalho de [Coelho 2011], dado que, naquele trabalho, a função $g(\cdot)$ adotada avalia cada item da lista $\mathcal{P}$ de acordo com sua altura.

A cada iteração da fase de construção, os $k$ melhores itens da lista $\mathcal{P}$, de acordo com a função $g(\cdot)$, ou seja, as $k$ primeiras peças desta lista, são colocadas em uma Lista Restrita de Candidatos ($LRC$). O componente aleatório consiste em selecionar, para cada inserção na solução, um item aleatório dentre os $k$ itens candidatos da $LRC$ (linha 7). Em seguida, caso o elemento selecionado caiba no retângulo de estoque $R$, o seu encaixe é efetuado de forma a ocupar a menor área possível (menor retângulo $R' \subset R$) de $R$. Após a atualização das listas $\mathcal{P}$ e $LRC$, o processo se repete até que todos os itens tenham sido encaixados em $R$ [Coelho 2011].

A escolha da menor área retangular do retângulo $R$ a ser cortado (linha 10) é de suma importância para o procedimento a fim de que áreas retangulares maiores sejam reservadas para peças de maiores dimensões. Nesse sentido, a abordagem apresentada por Ferraz [Ferraz e Senne 2009]

```
1: procedure BUILDLOCALSOLUTION(R, P, k)
2:     Solução ← ∅;
3:     Ordene o conjunto P de peças de acordo com g(·);
4:     LRC ← conjunto das k melhores peças de P;
5:     while LRC ≠ ∅ do
6:         P ← P − LRC;
7:         Selecione, aleatoriamente, uma peça p_i ∈ LRC;
8:         L ← o conjunto de retângulos vazios obtidos a partir do retângulo R a ser cortado;
9:         if p_i cabe em algum retângulo R' ∈ L then
10:            Selecione o menor retângulo R* ∈ L que comporte p_i;
11:            Aloque p_i na área correspondente a R* em R;
12:            Solução ← Solução ∪ {p_i};
13:            if P ≠ ∅ then
14:                LRC ← LRC ∪ {melhor peça p_j ∈ P};
15:            end if
16:        end if
17:        LRC ← LRC − {p_i};
18:    end while
19:    Retorne Solução;
20: end procedure
```

foi a utilizada neste trabalho para a seleção da menor área retangular de $R$ em cada passo. Nela, há uma contagem em $R$ até que seja encontrada a menor área retangular que comporte a peça a ser cortada.

A Figura 2 ilustra esta etapa do procedimento. Supondo que a sequência de peças a serem cortadas de $R$ corresponda à apresentada na Figura 2(a), após a alocação da peça 1 são listadas as áreas retangulares vazias de $R$ (linha 8 do procedimento), conforme indicado pelos retângulos em linhas pontilhadas (Figura 2(b)). Assumindo que a verificação identifique aquela de contorno mais claro como a de menor dimensão e que comporta a próxima peça (linha 10), então a peça 2 é alocada nesta área de $R$ e o processo se repete até que todas as peças tenham sido alocadas. A Figura 2(c) explicita o estado de $R$ após a alocação das peça 1 e 2 e a subsequente escolha da menor área retangular vazia para a alocação da peça 3. Na Figura 2(d) é apresentado o estado de $R$ ao final do procedimento.

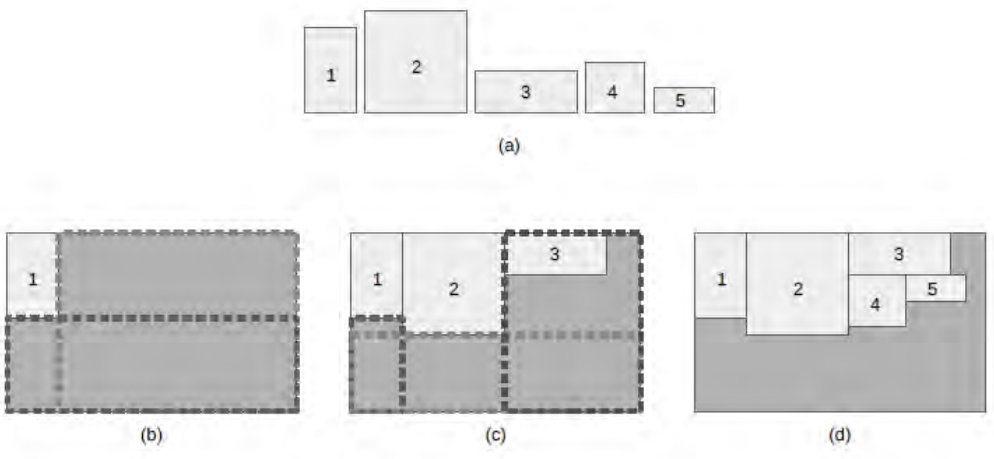


Figura 2: Exemplo de uma solução obtida pela fase de construção do GRASP.

### 2.2. Busca Local

A Busca Local é baseada em uma exploração iterativa da vizinhança de soluções com o objetivo de viabilizar um melhoramento da solução corrente através de perturbações locais. Os tipos de permutações passíveis de serem aplicados a uma solução são definidos com base na estrutura de sua própria vizinhança [Stützle 1998].

Neste trabalho, para a geração da vizinhança $\mathcal{N}(s)$ de uma dada solução $s$ (linha 2 do procedimento LocalSearch) foi adotado o movimento de troca de dois itens usado por Coelho [Coelho 2011]. De acordo com a estratégia, os vizinhos de $s$ são dados pela escolha aleatória de um item para ser permutado com os demais. Logo, para uma solução $s$ = (1 2 4 3 5), na qual o item 2 é o escolhido para atuar como pivô nas trocas, a vizinhança da solução é dada por:

$$\mathcal{N}(s) = \{s_1 = (2\ 1\ 4\ 3\ 5),\ s_2 = (1\ 4\ 2\ 3\ 5),\ s_3 = (1\ 3\ 4\ 2\ 5),\ s_4 = (1\ 5\ 4\ 3\ 2)\}$$

```
 1: procedure LocalSearch(s, f(s), P, R)
 2:     Gerar vizinhança N(s);
 3:     for all s' ∈ N(s) do
 4:         Alocar s' no retângulo de estoque R;
 5:         for all p_i ∈ P - s do
 6:             if p_i cabe em R then
 7:                 Alocar p_i em R;
 8:                 s' ← s' ∪ {p_i};
 9:             end if
10:         end for
11:         if f(s') > f(s) then
12:             s ← s';
13:         end if
14:     end for
15:     Retorne s;
16: end procedure
```

Subsequentemente à avaliação de cada vizinho de uma solução $s$, um novo passo foi adicionado a fim de se complementar o melhoramento da solução local. Nesta etapa adicional (linhas 5 a 10) - não descrita nos trabalhos referenciados - todas as demais peças de $\mathcal{P}$ que não compõem a solução $s$, e consequentemente não compõem nenhum vizinho $s'$ de $s$, são avaliadas a fim de se encontrar alguma que possa ser alocada no retângulo de $R$. Caso seja encontrada alguma peça $p_i \in \mathcal{P} - s$ que satisfaça a condição, esta é alocada seguindo a estratégia da menor área retangular de $R$ usada no procedimento BuildLocalSolution. Cada nova peça alocada passa a integrar a solução final.

### 2.3. Definição de Parâmetros

Dois parâmetros principais foram extensivamente testados para a aferição do algoritmo GRASP implementado: o tamanho da Lista Restrita de Candidados - utilizada pela heurística construtiva descrita na seção 2.1, e o número de iterações do GRASP.

Para a definição do tamanho *k* (linha 4 do procedimento BuildLocalSolution) da Lista Restrita Candidatos (*LRC*) foram avaliados uma gama de valores de *k*, sendo que os que levaram a melhores resultados foram *k* igual a 25%, 50% ou 75% do tamanho da lista de peças a serem cortadas (conjunto $\mathcal{P}$). Em seguida, foram feitos experimentos computacionais adicionais com esses três valores, os quais demonstraram que uma lista *LRC* com tamanho correspondente à metade (50%) do tamanho de $\mathcal{P}$, viabilizavam as melhores soluções. Logo, este valor foi fixado como tamanho máximo da lista $LRC$. Esta definição caracteriza uma diferença em relação ao trabalho de [Coelho 2011], o qual obteve os melhores resultados utilizando o parâmetro $k$ com valor igual a 2.

Os testes computacionais com diferentes valores para o número de iterações GRASP mostraram que a meta-heurística implementada produzia resultados menos oscilantes e de melhor qualidade quando o laço principal executava 1000 iterações. Este valor foi definido para a execução dos experimentos finais. Novamente, esta definição também significou outra diferença sobre o trabalho de [Coelho 2011], o qual fez uso de um número máximo de iterações do GRASP igual a 30.

## 3. Experimentos Computacionais

O algoritmo GRASP proposto foi implementado na linguagem C, compilado com o *GNU Compiler Collection* (GCC) - versão 4.8.2 - e executado em uma CPU de 2.871 MHz. O primeiro conjunto de testes é composto de 21 casos extraídos da própria literatura: 12 instâncias de Beasley [Beasley 1985], 2 instâncias de Hadjiconstantinou e Christofides [Hadjiconstantinou e Christofides 1995], 1 instância de Wang [Wang 1983], 1 instância de Christofides e Whitlock [Christofides e Whitlock 1977] e 5 instâncias de Fekete e Scheppers [Fekete e Schepers 1997].

Tabela 1: Resultados Computacionais - Problemas da literatura.

| | Problemas | | | Soluções | | | GRASP[1] | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | (L, W) | m | M | Ótimo | Beasley | GRASP[2] | Melhor | Média | D. P. | % Dif. |
| 1 | (10, 10) | 5 | 10 | 164 | 164 | 164 | **164** | 164 | 0 | 0 |
| 2 | (10, 10) | 7 | 17 | 230 | 230 | 230 | **230** | 230 | 0 | 0 |
| 3 | (10, 10) | 10 | 21 | 247 | 247 | 247 | **247** | 247 | 0 | 0 |
| 4 | (15, 10) | 5 | 7 | 268 | 268 | 268 | **268** | 268 | 0 | 0 |
| 5 | (15, 10) | 7 | 14 | 358 | 358 | 358 | **358** | 358 | 0 | 0 |
| 6 | (15, 10) | 10 | 15 | 289 | 289 | 289 | **289** | 289 | 0 | 0 |
| 7 | (20, 20) | 5 | 8 | 430 | 430 | 430 | **430** | 430 | 0 | 0 |
| 8 | (20, 20) | 7 | 13 | 834 | 834 | 834 | **834** | 834 | 0 | 0 |
| 9 | (20, 20) | 10 | 18 | 924 | 924 | 924 | **924** | 924 | 0 | 0 |
| 10 | (30, 30) | 5 | 13 | 1452 | 1452 | 1452 | **1452** | 1452 | 0 | 0 |
| 11 | (30, 30) | 7 | 15 | 1688 | 1688 | 1688 | **1688** | 1688 | 0 | 0 |
| 12 | (30, 30) | 10 | 22 | 1865 | 1801 | 1865 | **1865** | 1865 | 0 | 0 |
| 13 | (30, 30) | 7 | 7 | 1178 | 1178 | 1178 | **1178** | 1178 | 0 | 0 |
| 14 | (30, 30) | 15 | 15 | 1270 | 1270 | 1270 | **1270** | 1270 | 0 | 0 |
| 15 | (70, 40) | 19 | 42 | 2726 | 2721 | 2726 | 2716 | 2713 | 6,26 | 0,37 |
| 16 | (40, 70) | 20 | 62 | 1860 | 1720 | 1860 | **1860** | 1840 | 20 | 0 |
| 17 | (100, 100) | 15 | 50 | 27718 | 27486 | 27589 | 27661 | 27597 | 72,22 | 0.21 |
| 18 | (100, 100) | 30 | 30 | 22502 | 21976 | 21976 | **22502** | 22223 | 167,57 | 0 |
| 19 | (100, 100) | 30 | 30 | 24019 | 23743 | 23743 | **24019** | 23917 | 140,46 | 0 |
| 20 | (100, 100) | 33 | 61 | 32893 | 31269 | 32893 | **32893** | 32636 | 255,67 | 0 |
| 21 | (100, 100) | 29 | 97 | 27923 | 26332 | 27923 | 26249 | 25906 | 266,97 | 6.0 |

[1]Proposto neste trabalho. [2]Proposto por [Alvarez-Valdés et al. 2005].

A Tabela 1 apresenta um comparativo dos resultados obtidos. Nas colunas referentes a Problemas, são apresentados as dimensões do retângulo em estoque ($L$: comprimento, $W$: largura), número de tipos de peças ($m$) e o número máximo ($M$) de peças que podem ser cortadas. As colunas Melhor, Média, D. P. e % Dif., de GRASP (implementado neste trabalho), tabulam as informações da melhor solução encontrada, da média aritimética dos resultados obtidos, do desvio-padrão das soluções geradas em 5 execuções, assim como do percentual de diferença encontrada entre a melhor solução GRASP e a solução ótima, respectivamente. As melhores soluções obtidas pelo algoritmo GRASP implementado são contrapostas com os resultados encontrados por Beasley [Beasley 2004] e por outra implementação GRASP proposta na literatura [Alvarez-Valdés et al. 2005].

Dentre os 21 problemas testados, foi possível alcançar a solução ótima conhecida para 18 deles. Apenas três casos não se obteve sucesso, ou seja, problemas #15, #17 e #21. Em comparação com as soluções obtidas por Beasley [Beasley 2004], a implementação GRASP proposta neste trabalho alcançou resultados de melhor qualidade, tendo em vista que a primeira falhou na obtenção da solução ótima para 8 instâncias. Ressaltando ainda a qualidade do algoritmo construído, em um dos casos onde ambas implementações não obtém sucesso, o GRASP proposto consegue uma melhor aproximação do ótimo quando comparado com o resultado obtido por Beasley - problema #15.

Na Tabela 2 são apresentados os tempos de execução do algoritmo GRASP proposto (média e desvio padrão em 5 execuções), considerando-se o conjunto de instâncias mostrado na Tabela 1. Como ambientes computacionais distintos foram utilizados neste trabalho e nos demais de referência, os tempos demandados para o cômputo das soluções não podem ser diretamente comparados. Contudo, os dados apresentados na Tabela 2 podem agregar confiabilidade para fins de validação da abordagem proposta.

Tabela 2: Tempos de Execução - Problemas da literatura.

| | Problemas | | | GRASP - Tempo (s) | |
|---|---|---|---|---|---|
| # | $(L, W)$ | $m$ | $M$ | Média | Desvio Padrão |
| 1 | (10, 10) | 5 | 10 | 0.45 | 0.053 |
| 2 | (10, 10) | 7 | 17 | 0.96 | 0.08 |
| 3 | (10, 10) | 10 | 21 | 1.12 | 0.01 |
| 4 | (15, 10) | 5 | 7 | 0.79 | 0.01 |
| 5 | (15, 10) | 7 | 14 | 1.44 | 0.15 |
| 6 | (15, 10) | 10 | 15 | 1.74 | 0.02 |
| 7 | (20, 20) | 5 | 8 | 6.30 | 0.10 |
| 8 | (20, 20) | 7 | 13 | 9.29 | 0.21 |
| 9 | (20, 20) | 10 | 18 | 10.14 | 0.57 |
| 10 | (30, 30) | 5 | 13 | 29.49 | 0.27 |
| 11 | (30, 30) | 7 | 15 | 42.37 | 1.90 |
| 12 | (30, 30) | 10 | 22 | 52.19 | 0.94 |
| 13 | (30, 30) | 7 | 7 | 14.46 | 0.11 |
| 14 | (30, 30) | 15 | 15 | 25.29 | 0.48 |
| 15 | (70, 40) | 19 | 42 | 340.48 | 3.46 |
| 16 | (40, 70) | 20 | 62 | 333.01 | 2.93 |
| 17 | (100, 100) | 15 | 50 | 7549.14 | 97.09 |
| 18 | (100, 100) | 30 | 30 | 4221.56 | 42.57 |
| 19 | (100, 100) | 30 | 30 | 3724.84 | 55.65 |
| 20 | (100, 100) | 33 | 61 | 6354.19 | 38.02 |
| 21 | (100, 100) | 29 | 97 | 9569.83 | 136.52 |

Em um comparativo do algoritmo GRASP proposto com uma outra implementação do GRASP encontrada na literatura [Alvarez-Valdés et al. 2005], ambos apresentam desempenho similar na obtenção de soluções ótimas. Embora não alcancem o melhor resultado para a mesma quantidade de problemas, apenas para a instância #17 ambos falham simultaneamente. Contudo, para esse problema, o GRASP implementado neste trabalho consegue uma melhor aproximação da solução ótima. Além disso, apesar da estratégia de busca local utilizada tenha sido a apresentada por [Coelho 2011], nenhum comparativo foi feito contrapondo os resultados obtidos pelas duas implementações. Isso porque [Coelho 2011] considera a alocação de todas as peças retangulares de um dado conjunto em uma área retangular de largura fixa e altura variável. Como o problema difere

Tabela 3: Resultados Computacionais - Instâncias de [Amaral 1999].

| | Problemas | | | | GRASP | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | $(L, W)$ | $m$ | $M$ | Ótimo | Melhor | Média | D. P. | % Dif. | Tempo M. [1] | Tempo D. P.[2] |
| 1 | (30, 40) | 10 | 18 | 2763 | **2763** | 2763 | 0 | 0 | 85.16 | 1.45 |
| 2 | (30, 40) | 10 | 21 | 2766 | **2766** | 2766 | 0 | 0 | 58.91 | 0.65 |
| 3 | (30, 40) | 10 | 17 | 2774 | **2774** | 2774 | 0 | 0 | 65.04 | 0.33 |
| 4 | (30, 40) | 10 | 24 | 2591 | **2591** | 2591 | 0 | 0 | 56.63 | 0.32 |
| 5 | (30, 40) | 10 | 18 | 3116 | **3116** | 3116 | 0 | 0 | 58.57 | 0.90 |
| 6 | (40, 40) | 15 | 32 | 4014 | **4014** | 4014 | 0 | 0 | 128.98 | 1.49 |
| 7 | (40, 40) | 15 | 31 | 3966 | **3966** | 3938 | 33,84 | 0 | 242.48 | 1.70 |
| 8 | (40, 40) | 15 | 32 | 4092 | **4092** | 4092 | 0 | 0 | 148.71 | 2.03 |
| 9 | (40, 40) | 15 | 32 | 4178 | **4178** | 4173 | 6,57 | 0 | 188.88 | 1.79 |
| 10 | (40, 40) | 15 | 32 | 4308 | **4308** | 4308 | 0 | 0 | 220.09 | 2.34 |
| 11 | (50, 30) | 15 | 28 | 3538 | **3538** | 3538 | 0 | 0 | 138.64 | 1.13 |
| 12 | (50, 30) | 15 | 24 | 3449 | **3449** | 3432 | 9,86 | 0 | 125.37 | 0.48 |
| 13 | (50, 30) | 15 | 34 | 3405 | **3405** | 3405 | 0 | 0 | 147.89 | 1.60 |
| 14 | (50, 30) | 15 | 25 | 2908 | **2908** | 2908 | 0 | 0 | 80.71 | 0.44 |
| 15 | (50, 30) | 15 | 31 | 2854 | **2854** | 2777 | 165,10 | 0 | 164.14 | 1.71 |
| 16 | (50, 30) | 20 | 41 | 3935 | **3935** | 3935 | 0 | 0 | 150.45 | 1.11 |
| 17 | (50, 30) | 20 | 43 | 3608 | **3608** | 3572 | 32,03 | 0 | 154.93 | 1.09 |
| 18 | (50, 30) | 20 | 38 | 3695 | **3695** | 3643 | 33,09 | 0 | 123.45 | 1.41 |
| 19 | (50, 30) | 20 | 38 | 3681 | **3681** | 3621 | 45,35 | 0 | 282.51 | 2.64 |
| 20 | (50, 30) | 20 | 38 | 3539 | **3539** | 3490 | 42,51 | 0 | 166.97 | 2.27 |
| 21 | (50, 40) | 15 | 32 | 4714 | **4714** | 4681 | 42,41 | 0 | 283.62 | 2.40 |
| 22 | (50, 40) | 15 | 31 | 4735 | 4660 | 4637 | 25,04 | 1,6 | 309.51 | 3.19 |
| 23 | (50, 40) | 15 | 35 | 4948 | 4765 | 4770 | 42,73 | 3,7 | 273.47 | 1.59 |
| 24 | (50, 40) | 15 | 31 | 4585 | **4585** | 4585 | 0 | 0 | 290.69 | 2.84 |
| 25 | (50, 40) | 15 | 31 | 4798 | **4798** | 4798 | 0 | 0 | 158.21 | 1.06 |
| 26 | (50, 40) | 20 | 42 | 4527 | **4527** | 4527 | 0 | 0 | 221.24 | 0.99 |
| 27 | (50, 40) | 20 | 43 | 5294 | **5294** | 5294 | 0 | 0 | 272.83 | 1.26 |
| 28 | (50, 40) | 20 | 40 | 5203 | 5165 | 5140 | 34,04 | 0,7 | 375.58 | 4.14 |
| 29 | (50, 40) | 20 | 40 | 5167 | 5090 | 5061 | 22,87 | 1,5 | 260.18 | 1.63 |
| 30 | (50, 40) | 20 | 44 | 4993 | **4993** | 4844 | 87,12 | 0 | 346.48 | 3.48 |

[1]Tempo: Média (segundos). [2]Tempo: Desvio Padrão (segundos).

do estudado neste trabalho, um comparativo dos resultados obtidos pelas implementações não se mostrou pertinente.

O segundo conjunto de testes utilizado é composto de 30 instâncias geradas para o problema do corte bidimensional não-guilhotinado [Amaral 1999]. A Tabela 3 apresenta os resultados obtidos para cada instância. As colunas estão dispostas de forma análoga à Tabela 1. Uma coluna adicional foi incluída a fim de apresentar os tempos computacionais (média e desvio padrão em 5 execuções) obtidos para cada instância. A eficiência do algoritmo implementado é novamente atestada pelos resultados obtidos. Para os 30 casos testados, a solução ótima conhecida é alcançada para 26 deles e com boa aproximação para os outros 4 casos.

## 4. Conclusão

A implementação GRASP apresentada para o problema do corte bidimensional não-guilhotinado é resultado da junção de diferentes estratégias encontradas na literatura, associada

a uma etapa complementar que agrega robustez à busca por soluções eficientes. Naturalmente, o trabalho de se tentar efetuar o corte de todas as demais peças e de incluir tantas quantas seja possível em uma solução local, exige um esforço computacional que demanda uma análise mais apurada. A qualidade das soluções obtidas atesta a eficácia da estratégia. Contudo, em processamentos que envolvam instâncias de maior porte, o cômputo relacionado a esta etapa pode comprometer significativamente o desempenho do algoritmo.

Assim, direcionamentos podem ser dados no sentido de aprimorar a etapa de avaliação das demais peças que ainda não fazem parte de uma solução local. Ademais, novas estratégias de geração da vizinhança podem ser testadas a fim de se obter um processamento mais eficiente quando comparado à abordagem baseada na obtenção de todas as combinações possíveis de uma solução obtida na fase de construção.

**Referências**

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

Alvarez-Valdés, R., Parreño, F. e Tamarit, J. M. (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. (1999). A new mixed-integer programming model and solution approach for the two-dimensional nonguillotine cutting problem. Technical report, Universidade Federal do Espírito Santo.

Beasley, J. E. (1985). An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64.

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Beasley, J. E. (2004). A population heuristic for constrained two-dimensional nonguillotine cutting. *European Journal of Operational Research*, 156(3):601–627.

Christofides, N. e Whitlock, C. (1977). An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30–44.

Coelho, D. G. (2011). Um algoritmo evolutivo multiobjetivo aplicado ao problema de corte bidimensional guilhotinado. Master's thesis, Centro Federal de Educação Tecnológica de Minas Gerais.

Fekete, S. P. e Schepers, J. (1997). On more-dimensional packing iii: Exact algorithms. Technical Report 97290, Technical University of Berlin.

Feo, T. A. e Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

Ferraz, L. F. N. S. e Senne, E. L. F. (2009). Um algoritmo grasp para problemas de corte bidimensional não-guilhotinado. Relatório final pibic/cnpq, Universidade Estadual Paulista, Guaratinguetá.

Hadjiconstantinou, E. e Christofides, N. (1995). An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, 83(1):39–56.

Sepúlveda, G. P. L. (2013). Solução do problema de corte bidimensional de peças retângulares tipo não-guilhotinado usando simulated annealing. Master's thesis, Universidade Estadual Paulista, Ilha Solteira.

Stützle, T. G. (1998). *Local Search for Combinatorial Problems - Analysis, Improvements, and New Applications*. PhD thesis, Technische Universität Darmstadt, Alemanha.

Temponi, E. C. C. (2007). Uma metaheurística híbrida grasp-ils aplicada à solução do problema de corte bi-dimensional guilhotinado. In *Anais do XXXIX SBPO*, p. 1708–1717, Fortaleza. SOBRAPO.

Wang, P. Y. (1983). Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31(3):573–586.