Técnicas de Busca Heurística
Trabalho final

Alex Gliesch

7 de junho de 2016

1 Introdução

O artigo escolhido foi "Waste minimization in irregular stock cutting" [1]. O artigo estuda a classe de problemas de corte 2D, onde peças com formato poligonal, convexo ou não, são cortadas a partir de um pedaço de material, convexo ou não.

Especificamente, o problema estudado pelo artigo é: dada uma superfície poligonal simples qualquer e um conjunto de formas poligonais possíveis (peças), deseja-se cortar polígonos da superfície de forma a maximizar a soma das áreas dos polígonos cortados Em outras palavras, deseja-se maximizar o percentual de utilização da superfície de material. As formas poligonais podem ser rotacionadas arbitrariamente. Cada forma poligonal pode ser cortada múltiplas vezes. Como restrição, não pode haver interseção das áreas dos polígonos cortados. Este problema ocorre, por exemplo, na indústria têxtil, onde se quer cortar peças de roupa de um rolo de tecido minimizando a quantidade de tecido jogado fora.

O artigo endereça um problema diferente de outras abordagens da literatura: tipicamente, se tem uma folha de material com uma certa largura e comprimento infinito e se deseja encontrar as posições e orientações dos cortes de um conjunto finito de polígonos, de forma a minimizar comprimento da folha usado. Além disto, tipicamente os polígonos não podem ser rotacionados arbitrariamente: há um conjunto finito de ângulos de rotação permitidos.

Neste trabalho, apresenta-se em detalhes o algoritmo proposto em [1] e se faz uma análise crítica sobre variações de parâmetros e performance computacional de diferentes partes do algoritmo. Além disto, são propostas duas modificações simples, e os resultados são comparados.

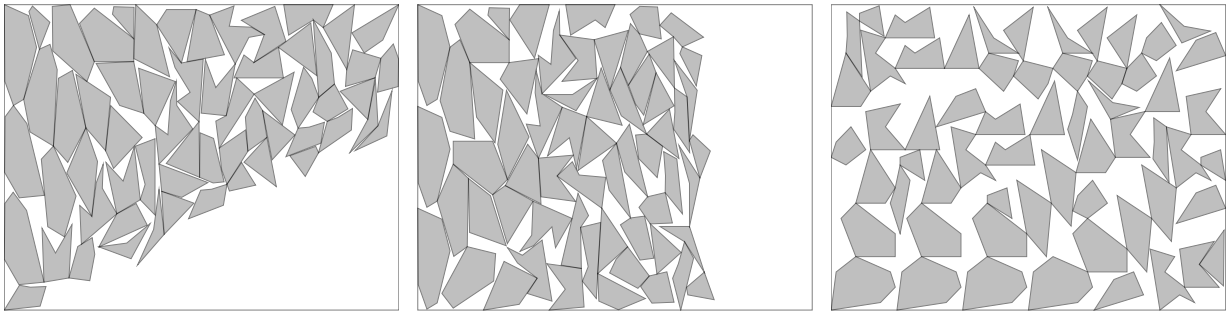
2 O algoritmo

O artigo usa uma estratégia construtiva gulosa para resolver o problema. Primeiramente, posiciona-se um polígono inicial qualquer em uma posição arbitrária na superfície de corte. Os polígonos restantes são então ordenados de forma descendente com respeito a suas áreas. Repetidamente, o algoritmo seleciona, entre os m polígonos com maior área, aquele com um posicionamento que menos aumenta a função objetivo. Este posicionamento é então realizado. Os posicionamentos de um polígono são as combinações de rotações dele por um ângulo $k\phi$, com $k \in \{0, \ldots, \lfloor 2\pi/\phi \rfloor\}$, com as translações de cada vértice do polígono para um vértice de um polígono já posicionado. m e ϕ são parâmetros. Cada posicionamento está sujeito a testes de factibilidade. Caso um polígono não tem posicionamento factível, ele é removido do conjunto de polígonos disponíveis. O algoritmo termina quando nenhum polígono possui posicionamento factível. O Apêndice A mostra o algoritmo em mais detalhes.

2.1 Factibilidade de um posicionamento

Um posicionamento é factível caso ele não faz interseção com nenhum outro polígono que já foi posicionado. São realizados 4 testes:

1. Teste dos ângulos: caso dois polígonos coincidem em um vértice, a soma dos ângulos internos não pode exceder 360°. Caso excede, há interseção.
2. Teste da caixa limitante (bounding box): este teste é realizado por questões de performance; caso as caixas limitantes de dois polígonos não têm interseção, então não há interseção entre os polígonos.
3. Teste de inclusão de pontos: caso qualquer vértice de um dos polígonos está dentro do outro, há interseção.
4. Teste de interseção de arestas: caso uma aresta do polígono intersecta com outra aresta de outro polígono, há interseção.



(a) (b) (c)

Figura 1: Em (a), a função objetivo utiliza ambos os termos x e y do centroide, priorizando posicionamentos na direção superior-esquerda. Em (b), a função objetivo utiliza apenas o termo x do centroide, priorizando posicionamentos à esquerda. Em (c), é usado apenas o termo de compactação, sem contribuição do centroide. Os resultados são da instância *poly4a*, com $m = 1$ e $\phi = 5°$, usando cada polígonos dado na instância apenas uma vez.

2.2 Função objetivo

A função objetivo correspondente à adição de um polígono z ao conjunto de polígonos fixos X é:

$$objective(z, X) = \frac{area(ch(X \cup \{z\})) - area(X \cup \{z\})}{area(ch(X \cup \{z\}))} \times \frac{centroid(z).x}{centroid(z).y}$$

onde $area(S)$ é a soma das áreas dos polígonos em um conjunto S, $ch(S)$ retorna um conjunto unitário contendo o polígono que representa a envoltura convexa da união dos pontos dos polígonos em S, e $centroid(p)$ é o centroide dos pontos do polígono p.

Esta função tem a intenção de minimizar dois componentes: a compactação, denotada pelo desvio relativo à área da envoltura convexa, e o empacotamento, denotado pela razão das posições x e y do centroide. A compactação faz com que os polígonos sejam cortados próximo um do outro, minimizando o desperdício de material entre os cortes; o empacotamento dá mais preferência a posicionamentos na direção superior-esquerda. Caso se queira minimizar o comprimento da folha de material, e não o desperdício, pode-se usar apenas o componente x do centroide. A Figura 1 ilustra o uso das componentes do centroide.

3 Modificação proposta 1: guloso-α

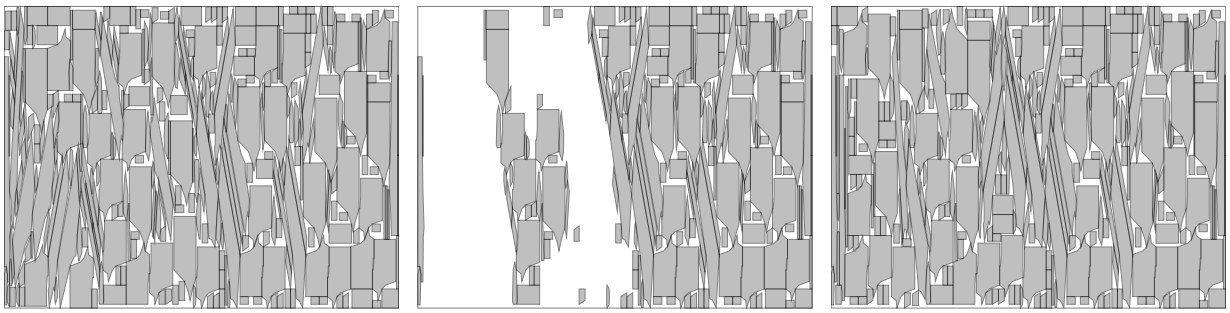
O algoritmo pode ser extendido para um guloso-α: em vez de selecionar o posicionamento que menos aumenta a função objetivo, seleciona com probabilidade uniforme um entre os $\alpha/100\%$ melhores posicionamentos. Note que, caso $m > 1$, os $\alpha/100\%$ melhores posicionamentos podem conter posicionamentos de polígonos diferentes; isto fornece uma maior variedade de soluções. O algoritmo construtivo é executado repetidamente até que o tempo limite seja atingido, e a melhor solução obtida é retornada.

4 Modificação proposta 2: algoritmo guloso iterado

O algoritmo guloso iterado começa com uma solução construída gulosamente e desconstroi parte dela usando um mecanismo aleatório. A solução então é reconstruída usando o procedimento guloso. Este processo é repetido até que o tempo limite seja atingido, e a melhor solução obtida é retornada.

A desconstrução é feita como segue. A solução construida pode ser vista como um grafo não-direcionado: cada polígono é um vértice, e dois vértices estão conectados caso os polígonos correspondentes se tocam. Note que, com a construção gulosa usada, o grau de cada vértice do grafo é pelo menos 1, pois os posicionamentos são obtidos conectando vértices de dois polígonos. Inicia-se uma busca em largura (BFS) neste grafo a partir de um polígono aleatório. A busca repetidamente expande polígonos de maneira FIFO e termina quando a soma das áreas dos polígonos expandidos é maior que d vezes a área total, onde d é um parâmetro. Todos os polígonos expandidos pela BFS são então removidos, o que caracteriza a desconstrução. Na prática, este procedimento prioriza remoções de polígonos que estão próximos um do outro. Caso a fila da BFS ficar vazia antes do limite de área ser atingido, um polígono aleatório é enfileirado.

A Figura 2 ilustra a construção e desconstrução do algoritmo.



(a) (b) (c)

Figura 2: Em (a), uma solução gulosa-α para a instância *trousers*, com $m = 4$, $\phi = 10°$ e $\alpha = 0.2$, tem uma utilização de 81.74% da área total. Em (b), a solução após desconstruir $d = 30\%$ da área total. Em (c), a solução após reconstrução gulosa, com utilização de 82.18%.

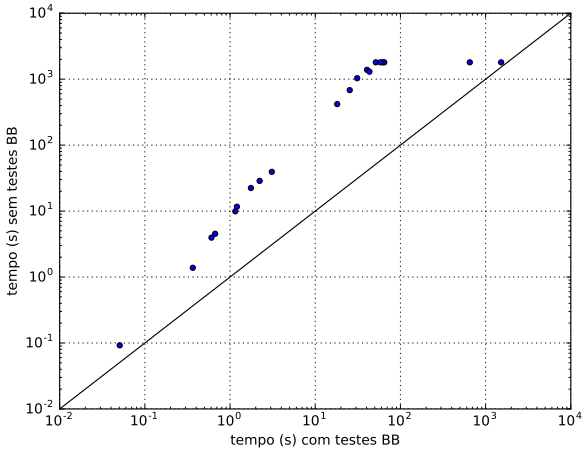


Figura 3: Comparação dos tempos de execução com e sem testes de caixa limitante.

5 Experimentos

Os experimentos foram executados em um processador Intel Core i7 930, rodando a 2.8GHz, e 12GB de memória. O compilador utilizado foi GCC 4.8 com flags `-O3 -std=c++11`. Cada teste foi executado por um limite de tempo de 30 minutos. Foram usadas 20 instâncias benchmark do Euro Special Interest Group in Cutting and Packing (ESICUP), disponíveis em `http://paginas.fe.up.pt/~esicup/`. Todas as instâncias são 2D com polígonos irregulares.

5.1 Análise de tempos de execução

Teste da caixa limitante

O artigo diz que o teste da caixa limitante (BB) economiza uma quantidade de tempo imensa durante o teste de factibilidade, por detectar a não-interseção de polígonos em estágios iniciais de teste. Para verificar se isto é verdade, foram realizados experimentos para todas as intâncias com e sem os testes BB. Foram utilizados $m = 1$ e $\phi = 90$.

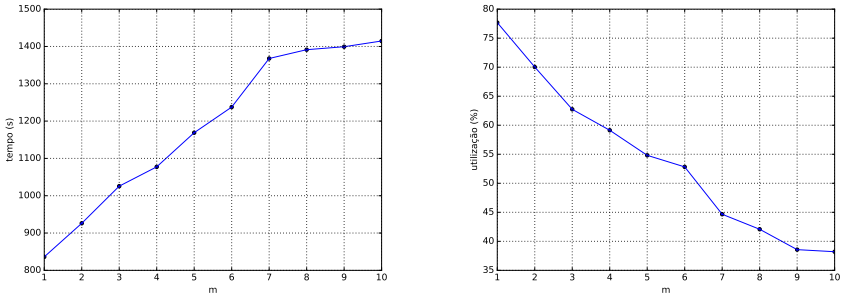
A Figura 3 compara os tempos de execução com e sem o teste BB. Pode-se observar que a versão sem testes BB precisa de cerca de uma ordem de magnitude mais tempo. Para as instâncias que não atingiram o tempo limite, a versão com testes BB obteve um tempo de execução médio de 8.47 s, enquanto a versão sem testes BB obteve tempo médio de 246.90 s. Aplicando o teste Wilcoxon de postos com sinais com a hipótese nula que a mediana das distribuições do tempos é igual, e hipótese alternativa de que é diferente, obteve-se um p-value de 0.0000019. Com um nível de significância 0.01, a hipótese nula é rejeitada.

Profiling do código

Foi realizado um profiling de uma execução do algoritmo, para ver quais partes têm maior impacto no tempo de execução. O profiling foi feito com o programa gprof, em uma execução da instância *trousers*, com $m = 1$ e

Tabela 1: Profiling da execução da instância *trousers* com $m = 2$ e $\phi = 10°$. A execução durou 197.67 s.

função	tempo (%)	tempo (% cumulativa)	número de chamadas
`Polygon::pointInside`	51.95	51.95	801749933
`Algorithm::isFeasible`	23.21	75.16	86290822
`lineSegmentsIntersect`	14.41	89.58	2168441174
`bondingBoxesIntersect`	6.09	95.67	1935166199
`Polygon::translated`	1.58	97.25	86290818
`Algorithm::constructiveGreedy`	1.51	98.76	1
`pointInRectangle`	0.90	99.66	398722265
outros	0.34	100.00	



(a) tempo (b) utilização

Figura 4: Resultados para a variação de m com $\phi = 5°$. Em (a), os tempos de execução médios; em (b), os valores resultado (percentual de utilização) médios.

$\phi = 5°$. A Tabela 1 mostra os resultados.

Embora o teste das caixas limitantes ajude a diminuir consideravelmente o tempo usado testando factibilidade, podemos ver que mais de 95% do tempo é gasto com eles: factibilidade, teste ponto dentro de polígono, interseção de segmentos de reta, caixas limitantes. O tempo usado com o cálculo da envoltura convexa é negligível.

5.2 Configuração de parâmetros m e ϕ com irace

O artigo não fornece os valores de m e ϕ foram usados nos testes. Acredito que foram usados diferentes valores para cada instância. Foi usado o irace [2] para configurar estes parâmetros: os intervalos usados foram $m \in [10]$ e $\phi \in \{5°, 10°, 30°, 45°, 60°, 90°, 180°\}$. Foram realizados no máximo 500 experimentos, cada um com tempo limite de 30 minutos. Caso a construção gulosa não termina até 30 minutos, a soma das áreas dos polígonos já posicionados é retornada.

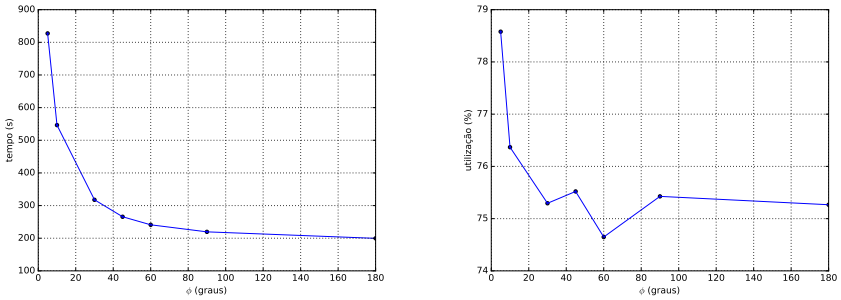
A configuração elite obtida pelo irace foi $m = 1$ e $\phi = 5°$. É importante notar que, como algumas execuções não terminam a construção em apenas 30 minutos, outras configurações elite podem ser obtidas com tempos limite diferentes. Como o parâmetro ϕ regula principalmente a precisão dos possíveis posicionamentos, espera-se que, conforme ϕ diminui para $< 5°$, os resultados médios sejam melhores, porém a um custo maior de tempo.

Variação do parâmetro m

A Figura 4 mostra os resultados da variação do parâmetro m com $\phi = 5$ fixo. Pode-se observar que o tempo de execução cresce aproximadamente linearmente conforme m aumenta, e os valores resultado decrescem. Como a configuração do irace sugere, $m = 1$ obteve os melhores resultados. Para $m > 3$ pode-se notar no gráfico valores muito baixos de utilização ($< 60\%$): isto ocorre pois, para m grande, na maioria das instâncias o tempo limite é alcançado antes da construção gulosa terminar, deixando uma grande quantidade de material não utilizado.

Variação do parâmetro ϕ

A Figura 5 mostra os resultados da variação do parâmetro ϕ com $m = 1$ fixo. Pode-se observar que o tempo decresce de uma forma logarítmica inversa conforme ϕ aumenta. Como esperado, valores menores de ϕ aparentam oferecer valores objetivo maiores, em média, apesar de algumas variações com $\phi \in \{30, 45, 60\}$. Como a configuração do irace sugere, $\phi = 5°$ obteve os melhores valores, porém, com um maior custo de tempo.



(a) tempo (b) utilização

Figura 5: Resultados para a variação de ϕ com $m = 1$. Em (a), os tempos de execução médios; em (b), os valores resultado (percentual de utilização) médios.

Tabela 2: Resultados do guloso-α.

α	0.01	0.05	0.1	0.25	0.5
iterações (média)	130.30	124.95	124.40	121.45	123.15
utilização (média)	77.53	75.54	74.72	73.83	73.17

5.3 Guloso-α

Foram executados testes com $m = 1$, $\phi = 5°$ e $\alpha \in \{0.01, 0.05, 0.1, 0.25, 0.5\}$. Os tempos de execução são omitidos pois todos são 30 minutos. Observa-se na tabela que $\alpha = 0.01$ obteve valores médios melhores. Foi realizado o teste de Friedman com a hipótese nula que as medianas dos resultados para cada α são iguais; obteve-se um p-value de 9.956×10^{-10}, e a hipótese nula foi rejeitada com nível de significância 0.01.

5.4 Guloso iterado

Foram executados testes com $m = 1$, $\phi = 5°$ e $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. A Tabela 3 mostra os resultados. A tabela completa pode ser vista no Apêndice **??**. Os tempos de execução são omitidos pois todos são 30 minutos. Observa-se que todos os d obtiveram utilização média bastante similar; $d = 0.4$ teve o maior valor, 79, que melhora a solução inicial por 1.32%. Apesar disto, as diferenças são estatisticamente significativas: o teste de Friedman obteve um p-value de 0.00026 e rejeitou a hipótese nula com nível de significância 0.01.

5.5 Comparação dos métodos

A Tabela 4 compara a configuração do irace, do gulosoα com $\alpha = 0.01$, do guloso iterado com $d = 0.4$, e de um algoritmo aleatório. Os posicionamentos no algoritmo aleatório são obtidos adicionando repetidamente um polígono aleatório na primeira posição factível encontrada, até que nenhum polígono possa ser mais adicionado.

Observa-se que os resultados do guloso iterado são melhores que os da configuração do irace em todas as instâncias. Isto já era esperado, pois o guloso iterado começa da solução original do irace e apenas a melhora. É interessante notar que o algoritmo aleatório obteve resultados melhores para as instâncias shirts e swim: por serem instâncias grandes, os demais algoritmos param por limite de tempo antes da solução inicial ser construída.

Foram realizados testes Wilcoxon pareado de postos com sinais para todos os pares de algoritmos. Como são 6 testes, foi feita a correção de Bonferroni, e foi usado o nível de significância $0.01/6 = 0.0016$. A Tabela 5 resume os resultados. Apenas obteve-se significância estatística na diferença entre os resultados do guloso-iterado e da configuração do irace.

Tabela 3: Resultados do guloso iterado.

d	0.1	0.2	0.3	0.4	0.5
iterações (média)	162.30	181.70	156.85	251.50	180.80
util. inicial (média)	77.68	77.68	77.68	77.68	77.68
util. final (média)	77.71	78.08	78.15	79.00	78.99

Tabela 4: Comparação dos métodos propostos.

instância	conf. irace	guloso-α	guloso iterado	aleatório
albano	77.81	79.05	78.63	59.06
blaz	94.51	92.74	95.25	56.51
dagli	82.05	82.36	82.05	62.05
dighe1	79.85	76.80	80.72	45.26
jakobs1	99.74	99.68	99.94	67.87
jakobs2	96.28	85.61	96.28	53.83
poly1a	77.03	77.35	77.03	57.77
poly2a	75.03	78.84	78.21	54.45
poly2b	77.91	78.33	77.91	59.29
poly3a	74.75	78.45	78.66	50.14
poly3b	86.62	85.09	88.17	66.24
poly4a	76.60	79.68	77.79	52.21
poly4b	87.17	84.80	88.64	67.23
poly5a	77.55	78.68	77.55	54.55
poly5b	86.95	85.94	89.48	66.40
shapes0	79.96	80.50	79.96	49.48
shapes1	79.96	80.34	79.96	48.97
shirts	44.50	44.93	49.66	73.17
swim	15.65	18.13	20.19	63.89
trousers	83.59	83.39	83.81	63.43
média	77.68	77.53	78.99	58.59

Tabela 5: Testes estatísticos sobre os dados da Tabela 4.

algoritmos	p-value	nív. sig.	conclusão
irace e guloso-α	0.5168	0.0016	inconclusivo
irace e guloso iterado	0.0011	0.0016	hip. nula rejeitada
irace e aleatório	0.0038	0.0016	inconclusivo
guloso-α e guloso iterado	0.1097	0.0016	inconclusivo
guloso-α e aleatório	0.0033	0.0016	inconclusivo
guloso iterado e aleatório	0.0025	0.0016	inconclusivo

6 Conclusão

Foi feita uma análise crítica do algoritmo guloso descrito em [1]. Analisando o tempo computacional do algoritmo, verificou-se que o maior gargalo está na computação da factibilidade de um posicionamento. Os parâmetros m e ϕ foram configurados com o irace e obteve-se uma configuração elite $m = 1$ e $\phi = 5°$. Foram propostas duas modificações simples ao algoritmo original: um guloso-α e um guloso iterado, e mostrou-se que o guloso iterado é pelo menos tão bom quanto o algoritmo original. Não foi possível obter significância estatística comparando o gulosoα e o algoritmo original.

Algumas ideias para trabalhos futuros:

- Combinar o guloso iterado com uma construção guloso-α. Configurar os parâmetros m, ϕ, α e d em conjunto, com o irace.
- Para obter uma maior precisão, pode-se fazer um super-sampling no conjunto de vértices, adicionando vértices uniformemente distribuídos sobre as arestas dos polígonos. Assim, há mais posicionamentos possíveis.
- Em vez de conectar vértices dos polígonos na construção gulosa, pode-se buscar aproximar duas arestas o máximo possível; arestas cujos coeficientes angulares são parecidos devem dar menos desperdício, quando aproximadas. O caso ideal seria quando duas arestas têm o mesmo coeficiente angular: os polígonos se tocam.

Referências

[1] Dalalah, D., Khrais, S., & Bataineh, K. (2014). *Waste minimization in irregular stock cutting.* Journal of Manufacturing Systems, 33(1), 27-40.

[2] López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). *The irace package, iterated race for automatic algorithm configuration.* Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

A Pseudocódigo

Algorithm 1 Algoritmo guloso. Denota-se um polígono por um conjunto de pontos 2D. A função *rotate* rotaciona todos os pontos de um polígono por um ângulo θ; a função *translate* adiciona um offset a todos os pontos de um polígono.

```
1: procedure FIND_BEST_POSITION(p, fixed, φ)
2:     best_value ← ∞
3:     best_polygon ← ∅
4:     for θ ∈ {kφ | k ∈ {0, ..., ⌊360°/φ⌋}} do
5:         p' ← rotate(p, θ)
6:         for q ∈ fixed do
7:             for v ∈ q do
8:                 for u ∈ p do
9:                     p'' ← translate(p', v − u)
10:                    if feasible(p'', fixed) and objective(p'', fixed) < best_value then
11:                        best_value ← objective(p'', fixed)
12:                        best_polygon ← p''
13:                    end if
14:                end for
15:            end for
16:        end for
17:    end for
18:    return (best_value, best_polygon)
19: end procedure
20:
21: procedure GREEDY_ALGORITHM(surface, polygons, m, φ)
22:    fixed ← ∅
23:    floating ← polygons
24:    while floating ≠ ∅ do
25:        best_value ← ∞
26:        best_polygon ← ∅
27:        lm ← m poligonos em fixed com maior area
28:        for p ∈ lm do
29:            value, polygon ← FIND_BEST_POSITION(p, fixed, φ)
30:            if value < best_value then
31:                best_value ← value
32:                best_polygon ← polygon
33:            end if
34:        end for
35:        if best_value = ∞ then
36:            floating = floating \ lm
37:        else
38:            fixed = fixed ∪ {best_polygon}
39:        end if
40:    end while
41:    return fixed
42: end procedure
```