



Otimização do corte de peças em chapas de metal

Marcos P. Vigliassi - Computação BioInspirada – 2004

E-mail: mvigliassi@inteligenciaartificial.com.br

Abstract

O Algoritmo Evolutivo (AE) mostra-se altamente eficiente na resolução de problemas complexos, com um amplo universo de soluções. Demonstramos a sua utilização na otimização do corte de peças em chapas de metal. Essa organização visa diminuir a área ocupada pelas peças, evitando espaços vazios. Utilizamos o AE para encontrar a sequência de peças e mais um algoritmo para tratar as bordas das peças, diminuindo o universo de pesquisa das possíveis soluções.

Palavras-chave

Algoritmo Evolutivo, Algoritmo Genético, Computação BioInspirada, otimização, corte de peças, bordas, crossover, mutação, predação.

1. Introdução

Atualmente as empresas buscam otimizar seus processos a fim de diminuir custos e tempo de fabricação, tornando seus produtos mais competitivos. Nessa busca de melhorias são substituídos muitos métodos empíricos, que normalmente geram acréscimos nos custos. Sugerimos nesse artigo a organização automática de peças para o corte em chapas de metal, maximizando o uso de matéria prima e diminuindo o tempo de fabricação. Essa solução poderá ser adotada em indústrias automotivas, aeronáuticas, metalúrgicas e adaptada para muitas outras, como por exemplo, têxteis e móveis.

Para a solução desse problema abordagens clássicas como buscas randômicas, métodos heurísticos (LFLA), redes neurais artificiais mostram-se muito lentas para convergir até soluções aceitáveis [1], requerendo muito tempo de processamento. Propomos uma solução baseada em novas técnicas de Computação BioInspirada utilizando Algoritmos Evolutivos e uma nova abordagem combinando um algoritmo que trata as bordas das peças [2] com rotação. O AE será responsável pela otimização da sequência de peças a serem assentadas e o algoritmo das bordas fará o correto assentamento dessas peças, sem sobreposição. A melhor solução será a que apresentar a maior área livre (descrita mais adiante). Calculada a área livre temos a definição do aproveitamento como sendo:

$$A_p = \frac{A_L * 100}{\left(H_c * W_c \right) - \left(\sum_{i=1}^n W_i * H_i \right)}$$

onde:

A_p : aproveitamento obtido;

A_L : área livre;

H_c : altura da chapa;

W_c : largura da chapa;

N : total de peças;

i : peça.

2. Algoritmo Evolutivo

Os Algoritmos Genéticos (AGs) surgiram na década de 60, criados por John Holland na Universidade de Michigan e popularizados por David Goldberg. O autor pretendia desenvolver um algoritmo para a resolução de problemas não específicos, usando uma analogia aos sistemas adaptativos da natureza. Esse algoritmo consiste em representar uma possível solução na forma de um cromossomo (*array*). Gerando-se várias possíveis soluções temos um conjunto de cromossomos, chamado de população. A otimização ocorre nessas populações em etapas que chamamos de gerações. Uma geração é um conjunto de possíveis soluções (cromossomos) que será avaliada segundo um critério de seleção. No processo de avaliação, cada indivíduo recebe um *score* (aptidão), que será utilizado na seleção dos indivíduos melhores adaptados (maior *score*). Existem muitos métodos para a seleção, dentre os quais os mais utilizados são o método da Roleta e o Elitismo. No método da Roleta os indivíduos com maior aptidão tem maior probabilidade de serem escolhidos, enquanto no Elitismo, os indivíduos com maior aptidão serão sempre escolhidos. Após a seleção dos mais adaptados é produzida uma nova geração, excluindo os indivíduos menos adaptados e criando novos (filhos). Na criação são usados basicamente dois operadores:

- Crossover: Recombinação de características dos pais. Na Fig. 1 temos um exemplo simples de crossover. Um pai dando origem a um filho através de uma divisão ao meio e uma inversão de posições. Podemos ter muitas maneiras diferentes para gerar filhos como crossover de um ponto, dois pontos, multipontos e uniforme.

Pai	1	0	1	1	0	0	1	0	1	0
Filho	0	1	0	1	0	1	0	1	1	0

Fig. 1. Exemplo de crossover

- Mutação: Introduz a diversidade genética a população. Ela altera aleatoriamente um ou mais genes de um cromossomo. Essa diversidade faz com que o AG tenha um espaço de busca maior, normalmente não explorado somente através de crossover. Se não a utilizarmos ficamos sujeitos a uma solução que pode não ser a ideal, é o que chamamos de Ótimo Local, ou seja, uma população que evoluiu em um pequeno campo de busca, não atingindo o Ótimo Global. A figura 2 mostra uma evolução que atinge o Ótimo Global. Note que a mutação deve ser suficiente para alternar entre os espaços de busca locais, mas não deve ter uma taxa muito alta, já

que ela elimina uma informação evoluída anteriormente. Normalmente utilizamos uma taxa de 0,1 a 5%.

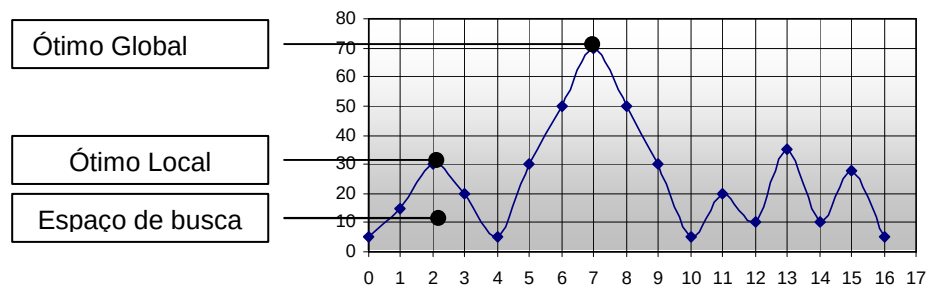


Fig. 2. Ótimo Local, Global e Espaço de busca

Descrevemos novas técnicas que surgiram a partir dos estudos de Holland. A sua concepção encontrava soluções para problemas determinados, impossibilitando o seu uso em problemas complexos, cujas variáveis são mutantes. Para suprir essa deficiência foram criadas técnicas capazes de evoluir uma solução mesmo com mudança das variáveis, esse novo modelo de algoritmo chamamos de Algoritmo Evolutivo (figura 3).

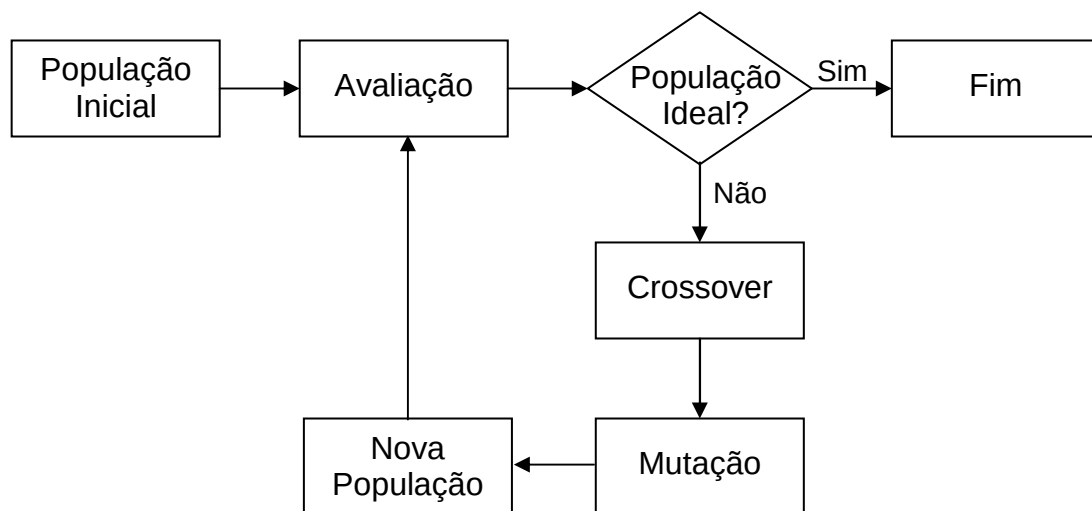


Fig. 3. Algoritmo Evolutivo

3. Implementação

Nesse artigo abordamos a resolução do problema da otimização do corte de peças em chapas de metal. Tratamos somente *layout's* 2D com peças retangulares. Para a resolução desse problema temos algumas premissas:

- Não deve haver sobreposição de peças

$$-\left[\sum_{i=1}^n W_i * H_i\right] < \left[H_c * W_c\right]$$

- W_i deve ser $\leq H_c$ ou W_c
- H_i deve ser $\leq H_c$ ou W_c

Atendendo as premissas acima, o algoritmo deverá solucionar os problemas básicos: alto tempo de processamento e a falta de flexibilidade diante de novas situações.

4. Representação do Algoritmo Evolutivo

4.1. Cromossomo

Representado por $[P1, P2, \dots, Pn, score]$

O cromossomo representa a sequência em que as peças serão assentadas na chapa de metal. Nessa sequência cada peça recebe um número, onde P1 é peça de número um, P2 é a peça de número dois e n é o total de peças. O *score* é a área livre na chapa após o assentamento dessa sequência de peças. Na Fig. 4 a área pintada representa área ocupada (A_o) em uma determinada sequência.

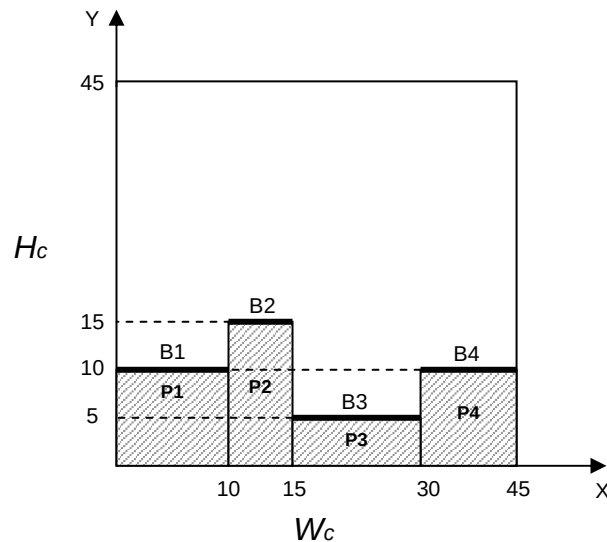


Fig. 4. Representação da área ocupada

Considerando a Fig. 4 temos:

$P_i (X_{pi}, Y_{pi}, W_{pi})$, onde, X_{pi} é a posição da borda esquerda da peça no eixo X, Y_{pi} é a posição da borda superior no eixo Y e W_{pi} é a largura da peça. Ex.: $P1(0,10,10)$.

$B_m (X_{bm}, Y_{bm}, W_{bm})$, onde B_m é a representação de uma borda m , X_{bm} é a posição inicial da borda no eixo X, Y_{bm} é a posição da borda no eixo Y e W_{bm} é a largura da borda. Borda é a linha formada após o assentamento de uma peça. Ela aparece destacada na Fig. 4. O cálculo da área ocupada é definido por:

$$A_o = \sum_{i=1}^m (W_{bm} * Y_{bm})$$

$$\text{logo } A_L = ((H_c * W_c) - A_o)$$

Na Fig. 4 as peças encaixam-se perfeitamente. Já na Fig. 5, a seqüência criou uma lacuna entre as peças, com isso a área ocupada é maior que a soma das áreas das peças e a área livre diminui.

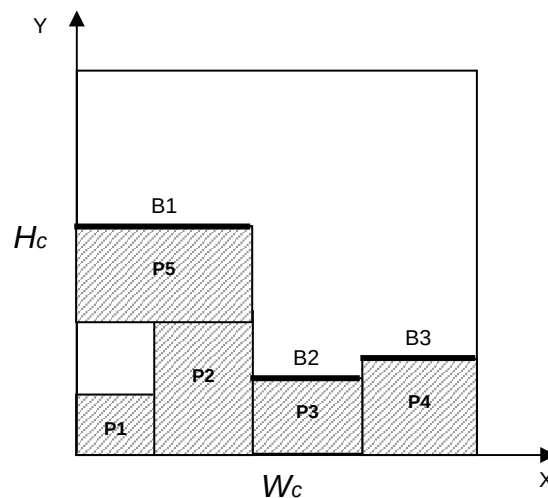


Fig. 5. Representação da área ocupada – com uma lacuna

4.2. População

É o número de cromossomos que será avaliado a cada geração. Quanto maior é a quantidade de cromossomos, maior a probabilidade de encontrar a melhor solução, por outro lado o tempo de processamento aumenta. Essas variações são demonstradas no item Resultados. A população inicial é gerada randomicamente.

5. Assentamento das Peças

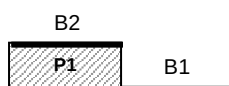
Para assentar as peças sobre a chapa usamos uma combinação do algoritmo de Leo Yeung [2] com um teste para rotacionar as peças. Dada uma seqüência de peças $[P1, P2, \dots, Pn]$ iremos assentar as peças nessa ordem, buscando sempre a borda com o valor de Y mais baixo. Podem ocorrer as seguintes situações ao assentar uma peça:

- Se $W_{p1} < W_{b1}$, a peça tem uma largura menor que a largura da borda. Se houver essa condição, é verificado ainda se:

$$H_{p1} > W_{p1} \text{ e } H_{p1} < W_{b1}.$$

Caso isso aconteça, a peça é rotacionada 90°, pois isso minimizará a utilização da chapa no eixo Y, visto que a altura da peça é maior que a sua largura.

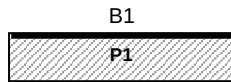
Será gerada uma nova borda filha B2, como mostrado no exemplo abaixo:



B1 ($X_{b1} + W_{p1}, Y_{b1}, W_{b1} - W_{p1}$)

B2 ($X_{b1}, Y_{b1} + H_{p1}, W_{p1}$)

- Se $W_{p1} = W_{b1}$, a peça tem uma largura igual à largura da borda. Isso faz com que somente seja alterada a altura da borda atual.



B1 ($X_{b1}, Y_{b1} + H_{p1}, W_{b1}$)

- Se $W_{p1} > W_{b1}$, a largura da peça é superior a largura da borda. Dessa forma a peça não pode ser assentada diretamente. Verificamos se a peça pode ser rotacionada:

Se $H_{p1} < W_{b1}$, então rotacionaremos a peça 90°.

Caso isso não seja possível teremos que unir as bordas até que a largura da borda seja suficiente para assentar a peça. Primeiramente tentamos unir as bordas à esquerda da borda atual. Soma-se a largura da borda atual com a próxima à esquerda. O Y da nova borda gerada será o de valor mais alto, seja da atual ou da próxima. Fazemos isso até que $W_{p1} \leq W_{b1}$ ou $X_{b1} = 0$. Se ainda não conseguirmos uma largura suficiente começamos a fazer o mesmo processo com as bordas à direita da atual. Quando alcançarmos a largura desejada assentaremos a peça como na Fig. 5, onde houve a união de duas bordas.

O tratamento das bordas através desse algoritmo otimiza o número de soluções avaliadas pelo Algoritmo Evolutivo, diminuindo assim o tempo de processamento.

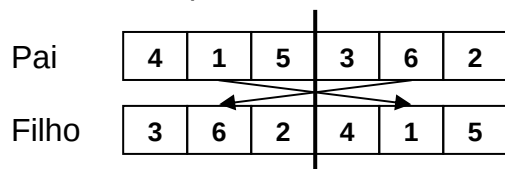
6. Geração das Novas Populações

Depois de assentadas todas as seqüências de uma população e calculadas as respectivas áreas livres, ordenamos essas seqüências em ordem crescente levando em consideração a área livre. Dividimos a população ao meio, eliminamos os piores indivíduos e aplicamos os operadores abaixo, para a geração da nova população:

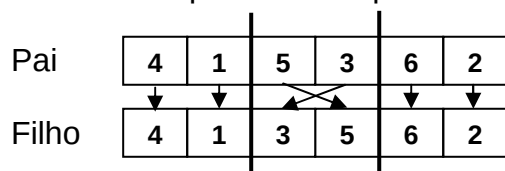
6.1. Crossover

Para gerar a nova população aplicamos o crossover nos pais gerando, cada um, um filho, segundo as operações abaixo:

- Divisão do pai ao meio e inversão



- Divisão do pai em duas partes aleatórias e inversão dos genes do centro.



Os resultados de cada tipo de crossover estão descritos no item Resultados.

6.2. Mutação

Para manter a diversidade da população precisamos inserir indivíduos diferentes dos padrões de crossover acima. Para isso usamos o seguinte método:

1. Sorteamos um número de 1 a 10
2. Se
 - número sorteado ≤ 7 , então muta um gene
 - número sorteado > 7 e ≤ 9 , então mutam dois genes
 - número sorteado = 10, então mutam 4 genes
3. Sorteamos uma linha entre a metade da população gerada através de crossover, onde será aplicada a mutação.
4. Na linha escolhida, sorteamos um número entre um e o total de peças, para encontrarmos a posição que será mutada.
5. Sorteamos um número entre um e o total de peças, que será o novo valor da posição sorteada no item 4.
6. Substituímos o antigo valor da posição sorteada pelo novo valor, sorteado no item 5.
7. Localizamos na sequência onde estava o novo valor sorteado no item 5 e substituímos pelo valor antigo da posição sorteada no item 5.

6.3. Predação

Essa é uma nova técnica utilizada no Algoritmo Evolutivo para gerar diversidade na população. Ela consiste em “eliminar” um indivíduo pouco adaptado a cada número específico de gerações. Neste artigo estamos eliminando um indivíduo pouco adaptado a cada dez gerações, e incluindo em seu lugar um totalmente randômico. Dessa forma cobrimos um espaço de busca maior do que somente com a mutação e não prejudicamos a evolução dos indivíduos, já que a Predação acontece a cada dez gerações.

7. Resumo do Algoritmo

Segue uma representação simples do Algoritmo Evolutivo e do tratamento das bordas.

Algoritmo Evolutivo

```
GerarPopulacaoInicial
while (condicao <> true)
  i := 0
  // varre toda a geração
  while (i < TotalIndividuos) do
    j := 0
    // o array de bordas fica somente com a borda inferior da chapa
    ZerarBordas()
    while (j < TotalPecas) do
      // posiciona a peça
      AssentaPeca(NumeroPeca)
      inc(j)
    end
    // calcula área livre
    z := 0
    AreaLivre := 0
```

```

        while (z < TotalBordas) do
            AreaLivre := (AreaLivre + ((AlturaChapa - AlturaBorda) * LarguraBorda)
            inc(z)
        end
        inc(i)
    end
    // se a condição fim ainda não for atingida, gera uma nova população
    GeraNovaPopulacao()
end

```

Descrição de algumas funções.

AssentaPeca(NumeroPeca)

```

// ordena o array de bordas pelo y
ordenaBordas()
i := 0
while (i < TotalBordas) do
    // largura da borda suficiente
    if (LarguraPeca < LarguraBorda) then
        // testa a possibilidade da rotação da peça
        if (AlturaPeca > LarguraPeca) and (AlturaPeca <= LarguraBorda) then
            temp := AlturaPeca
            AlturaPeca := LarguraPeca
            LarguraPeca := temp
            continue
        end
        // divide a borda atual e cria uma nova borda
        // lembrando que bordas(x,y,w)
        BordaAtual := (BordaAtual + LarguraPeca, BordaAtual, LarguraBordaAtual - _
            LarguraPeca)
        BordaNova := (BordaAtual, BordaAtual + AlturaPeca, LarguraPeca)
    end
    // largura da borda igual a largura da peça
    if (LarguraPeca = LarguraBorda) then
        // somente o y da borda atual é alterado
        BordaAtual := (BordaAtual, BordaAtual + AlturaPeca, BordaAtual)
    end
    // largura da borda insuficiente
    if (LarguraPeca > LarguraBorda) then
        // testa a possibilidade de rotação
        if (AlturaPeca <= LarguraBorda) then
            temp := AlturaPeca
            AlturaPeca := LarguraPeca
            LarguraPeca := temp
            continue
        end
        // ordena a borda por x
        ordenaBordasPorX()
        while (LarguraPeca > LarguraBorda) do
            // testa se é a última borda a esquerda x = 0
            // Se sim, une as bordas a direita
            if (x da BordaAtual = 0) then
                BordaAtual =
                    (BordaAtual, BordaAtual + ProximaBorda, BordaAtual + ProximaBorda)
                ExcluiProximaBorda()
            else
                BordaAtual =
                    (BordaAtual, BordaAtual + BordaAnterior, BordaAtual + _
                        BordaAnterior)
                ExcluiProximaBorda()
            end
        end
        ordenaBordas()
    end
    end
    inc(i)
end

```

8. Resultados

Implementamos o algoritmo proposto na linguagem Delphi V, sendo executado em um PC Athlon XP 2.0 GHz, 512 MB de memória RAM e Sistema Operacional Microsoft Windows XP. O valor dos resultados abaixo é uma média extraída de 20 execuções.

Na Tabela 1, montamos um layout com 10 peças para analisar o tempo de processamento e as diferenças entre os tipos de crossover, para variações de tamanho da população.

População	Divisão ao meio e inversão		Divisão em três partes aleatórias e inversão do centro	
	Tempo (seg:ms)	Aproveitamento (%)	Tempo (seg:ms)	Aproveitamento (%)
10	00:867	99.211	00:757	99.421
20	01:585	99.263	01:394	99.632
50	04:147	99.421	03:058	99.632
100	08:197	99.842	06:343	99.737

Tabela 1 – Variações do tipo de crossover utilizado em 5.000 gerações

Na Tabela 2, temos o tempo de processamento e o aproveitamento com a variação do número de peças. A população é de 20 indivíduos e o número de gerações é 5000.

Peças	Tempo (min:seg:ms)	Aproveitamento (%)
5	00:00:516	100,00
10	00:01:992	99,05
50	00:25:668	83,65
100	01:24:276	69,55

Tabela 2 – Tempo de processamento

A Tabela 3 expressa as diferenças de tempo e aproveitamento quando utilizada a rotação das peças. Nesse teste a população é de 30 indivíduos e o número de gerações é 5000. Note a grande diferença de aproveitamento utilizando a rotação das peças.

Peças	Sem rotação		Com rotação	
	Tempo (min:seg:ms)	Aproveitamento (%)	Tempo (min:seg:ms)	Aproveitamento (%)
5	00:00:624	100.00	00:00:632	100.00
10	00:02:676	98.35	00:02:657	98.80
50	00:33:070	82.45	00:39:308	89.50
100	01:25:558	23.80	01:59:040	52.25

Tabela 3 – Rotação das peças

Na Tabela 4 montamos um layout com vinte peças onde o aproveitamento de 99% é possível e executamos o algoritmo até que ele encontre esse aproveitamento. Mostramos a otimização da nova técnica de Predação.

Sem predação		Com predação	
Tempo (min:seg:ms)	Gerações	Tempo (min:seg:ms)	Gerações
00:02:303	935	00:01:350	541

Tabela 4 – Predação – Aproveitamento fixo de 99%

A Fig. 7 mostra um exemplo do gráfico da evolução do aproveitamento. O aproveitamento inicial já tem um valor alto devido ao algoritmo de tratamento das bordas. Ele maximiza as soluções encontradas pelo AE.

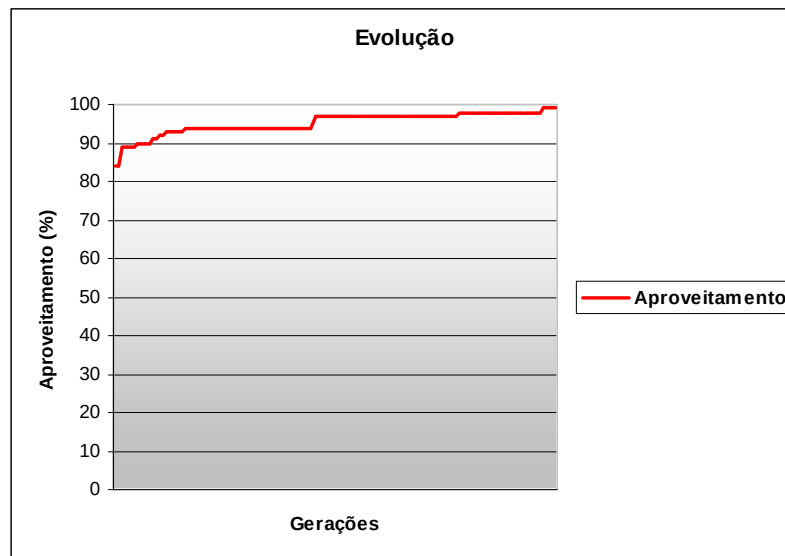


Fig. 7. Evolução do aproveitamento

9. Conclusões

Nesse artigo demonstramos uma nova abordagem para a solução do problema da otimização do corte de peças em chapas de metal. Usando para isso o tratamento das bordas das peças assentadas e novas técnicas na construção do Algoritmo Evolutivo. Verificamos que o maior ganho foi em relação ao tempo de processamento, que diminuiu muito em comparação a outras abordagens estudadas. Esse artigo pode ser utilizado como referência para abordagens mais complexas, como o assentamento de polígonos complexos e de forma irregulares.

Referências

1. Cus F, Balic J. Optimization of cutting process by GA approach. University Of Maribor, Slovenia, 2003.
2. Yeung LHW, Tang WKS. A Hybrid Genetic Approach for Garment Cutting in the Clothing Industry. IEEE Transactions on Industrial Eletronics, vol. 50, no. 3, 2003.
3. Elmaghraby AS, Abdelhafiz E, Hassan MF. An Intelligent Approach to Stock Cutting Optimization. University of Louisville, Louisville, USA, 2000.