# APLICAÇÃO DE ALGORITMO GENÉTICO PARA MINIMIZAR O NÚMERO DE OBJETOS PROCESSADOS E *SETUP* NUM PROBLEMA DE CORTE UNIDIMENSIONAL

**Julliany Sales Brandão**
Instituto Politécnico do Rio de Janeiro – UERJ
Nova Friburgo, Brasil
jbrandao@iprj.uerj.br

**Alessandra Martins Coelho**
Instituto Federal de Educação, Ciência e Tecnologia
do Sudeste de Minas – Campus Rio Pomba
Rio Pomba, Brasil
ale_contatos@ig.com.br

**João Flávio V. Vasconcellos**
Instituto Politécnico do Rio de Janeiro – UERJ
Nova Friburgo, Brasil
jflavio@iprj.uerj.br

## RESUMO

Este artigo apresenta a aplicação de uma nova abordagem utilizando Algoritmo Genético na resolução do Problema de Corte Unidimensional na minimização de dois objetivos, geralmente conflitantes, o número de objetos processados e o *setup*, simultaneamente. O modelo do problema, a função objetivo e o método proposto denominado SingleGA10, bem como os passos utilizados para sua resolução, também são apresentados. Os resultados obtidos pelo SingleGA10 são comparados com os métodos SHP, Kombi234, ANLCP300 e Symbio10 encontrados na literatura, a fim de verificar a capacidade de encontrar soluções viáveis e competitivas. Os resultados computacionais mostram que o método proposto, o qual utiliza apenas um algoritmo genético para resolver esses dois objetivos inversamente relacionados, proporciona bons resultados.

**PALAVRAS CHAVE.** Problema de Corte Unidimensional. Algoritmos Genéticos. *Setup*. Otimização Combinatória.

## ABSTRACT

This paper presents the application of the one new approach using Genetic Algorithm in solving One-Dimensional Cutting Problems, in order to minimize two objectives, usually conflicting, the number of processed objects and setup, simultaneously. The model problem, the objective function, the method denominated SingleGA10 and the steps used to solve the problem are also presented. The obtained results of the SingleGA10 are compared to the following methods: SHP, Kombi234, ANLCP300 and Symbio10, found in literature, verifying its capacity to find feasible and competitive solutions. The computational results show that the proposed method, which only uses a genetic algorithm to solve these two objectives inversely related, provides good results.

**KEYWORDS. O**ne-dimensional Cutting Problem. Genetic Algorithm. Setup. Combinatorial Optimization

## 1. Introdução

O problema de corte consiste em encontrar a melhor maneira de se obter peças de tamanhos distintos (itens), a partir do corte de peças maiores (objetos), com o objetivo de minimizar alguma espécie de custo ou maximizar o lucro. Os itens são dispostos no objeto para a realização de cortes ao longo da produção. À disposição dos itens dá-se o nome de padrão de corte. É relevante destacar a importância da geometria em problemas de corte, uma vez que as formas e dimensões dos itens e objetos determinam os seus possíveis padrões.

Kantorovick (1960) foi um dos pioneiros na área de problemas de corte. No entanto, o grande avanço da área ocorreu com os trabalhos de Gilmore e Gomory (1961,1963) estudando o problema de corte através do processo de geração de colunas.

Para solucionar um problema de corte necessita-se dos padrões de corte e da frequencia de padrões, ou seja, da quantidade de vezes que os padrões serão executados.

Embora o objetivo geral seja a minimização de perdas, encontram-se várias modelagens do problema como a maximização de lucros, a redução de objetos utilizados, do tempo de produção e/ou uma combinação desses.

O custo de preparação da máquina é um fator relevante em alguns processos de corte. Assim, torna-se interessante avaliar os efeitos da minimização do número de objetos processados (minimização da entrada) e da minimização do número de padrões de corte (*setup*), objetivos esses parcialmente conflitantes, para uma avaliação mais geral do custo. Haessler (1975) foi o primeiro a tratar o problema de corte unidimensional não-linear desse modo.

Este trabalho baseia-se em uma abordagem recente apresentada por Golfeto et al. (2007), na qual dois algoritmos genéticos que se interagem em uma relação mutualística têm o objetivo de minimizar o número de objetos processados e o *setup*. A abordagem aqui apresentada difere da abordagem de Golfeto et al. (2007) pelo fato de utilizar apenas um algoritmo genético na minimização desses dois objetivos. A contribuição do método proposto reside no fato de não se ter conhecimento na literatura, até o momento, da implementação de um único algoritmo genético utilizado na minimização desses dois objetivos em simultâneo.

O trabalho está organizado como segue: a seção 2 aborda formalmente o problema de corte com o objetivo de reduzir o número de objetos processados e o *setup*. Os conceitos básicos de algoritmos genéticos são apresentados na seção 3. A seção 4 expõe a implementação computacional e as seções 5 e 6, apresentam, respectivamente, os resultados computacionais e conclusões.

## 2. Problema de Corte Unidimensional

O problema de corte consiste em decidir sobre a maneira de se cortar um conjunto de peças retangulares com demandas pré-estabelecidas, a partir de uma placa retangular padrão, com a finalidade de minimizar o custo, o tempo e a perda de material envolvidos no processo de fabricação, ou uma combinação desses (Carneiro, 1994). Mais especificamente tem-se um número $m$ de itens diferentes, com largura $w_i$, $i = 1,..., m$ que devem ser produzidos para atender cada demanda $d$, partindo de uma peça mestra de largura $W > w_i$ para todo $i$, através de cortes ao longo do seu comprimento. Este problema é classificado como 1/V/I/R, segundo a tipologia de Dyckoff (1990) e SSSCSP (*Single Stock Size Cutting Stock Problem*) na tipologia de Wascher et al. (2007).

À disposição dos itens na peça mestra dá-se o nome de padrão de corte, e o tempo de reconfiguração da máquina de corte, a cada troca de padrão, denomina-se *setup*.

Na indústria, minimizar o número de objetos processados e/ou desperdício, na maioria das vezes, pode não ser suficiente. Outros fatores como redução dos custos e eficiência na entrega dos pedidos vêm estimulando as pesquisas nessa área. Quando uma demanda precisa ser atendida num curto espaço de tempo, o custo para trocas de padrões torna-se relevante visto que essas trocas demandam um ajuste nas máquinas (facas para corte) e conseqüentemente, tempo e trabalho. Assim, a minimização do custo de *setup* passa a ser um objetivo a mais no processo de produção da indústria e, se tratando de objetivos inversamente relacionados, faz-se necessário encontrar o equilíbrio entre eles.

Um modelo matemático formal que representa estes objetivos é descrito a seguir:

$$\text{Minimizar } c_1 \sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \delta(x_j) \qquad \textbf{(1)}$$

$$\text{Sujeito a: } \sum_{j=1}^{n} a_{ij} x_j \geq d_i \qquad i = 1,...,m$$

$$x_j \in N \qquad j = 1,...,n$$

onde:

$$\delta(x_j) = \begin{cases} 1, \text{ se} & x_j > 0 \\ 0, \text{ se} & x_j = 0 \end{cases}$$

$n$ = número de possíveis padrões de corte;
$m$ = número de itens diferentes;
$c_1$ = custo de cada bobina;
$c_2$ = é o custo de troca do padrão de corte;
$x_j$ = número de bobinas processadas com o padrão de corte *j;*
$a_{ij}$ = número de itens do tipo *i* no padrão *j;*
$d_i$ = número de itens *i* demandados;

**3. Algoritmos Genéticos**

Algoritmos genéticos são algoritmos de busca e otimização de soluções, baseados nos mecanismos genéticos e evolucionários dos seres vivos, como a seleção natural e a sobrevivência do mais apto, introduzidos por Charles Darwin no clássico "*The Origin of Species*" (1859) cujos primeiros trabalhos publicados conhecidos datam do final da década de 50 e início da década de 60.

Os Algoritmos Genéticos, introduzidos rigorosamente por John Holland (Holland, 1975), trabalham com uma população de indivíduos, na qual cada indivíduo representa uma solução possível para um dado problema. A cada indivíduo é dado um *fitness*, isto é, um valor que quantifica a adaptabilidade do indivíduo ao meio (problema tratado). Os indivíduos com maiores *fitness* possuem maiores oportunidades de serem escolhidos para a reprodução, através do cruzamento, e, com isso, difundir suas características pelas gerações seguintes, propiciando que áreas mais promissoras de busca sejam exploradas, levando o algoritmo genético, na maioria dos casos, à convergência para a solução ótima do problema.

Os Algoritmos Genéticos utilizam um processo adaptativo e paralelo de busca de soluções em problemas complexos. É adaptativo, pois as soluções correntes influenciam na busca de soluções futuras. O paralelismo decorre do fato de que a cada momento um conjunto de soluções é considerado.

Alguns conceitos importantes e necessários ao entendimento dos Algoritmos Genéticos são apresentados a seguir:

- gene: representação de cada parâmetro de acordo com o alfabeto utilizado (binário, inteiro ou real);
- genoma: conjuntos de genes para formar um indivíduo;
- fenótipo: codificação do genoma;
- população: conjunto de indivíduos no espaço de busca;
- seleção: escolha dos indivíduos que privilegia os mais aptos a permanecerem e multiplicarem a população. Os métodos de seleção mais utilizados são: elitismo, diversidade e torneio;

- recombinação (c*rossover)*: responsável por recombinar as características dos pais, durante a reprodução, permitindo que os seus descendentes herdem essas características. Os mais conhecidos são: ponto(s) de corte e uniforme;
- mutação: possibilidade de um gene qualquer do indivíduo apresentar características diferentes das existentes nos pais, após uma operação de recombinação.

## 4. Construção Computacional

O gene da população de solução foi representado por dois elementos. O primeiro elemento refere-se à quantidade de vezes (frequência) com que o padrão indicado pelo segundo elemento foi processado. Já o segundo elemento da população de solução serviu apenas como um ponteiro para a população de padrões.

O gene da população de padrões foi representado por um número real que indica o comprimento do item da lista de pedidos realizado pelo cliente. A Figura 1 apresenta uma representação do gene da população de padrões e da população de soluções.

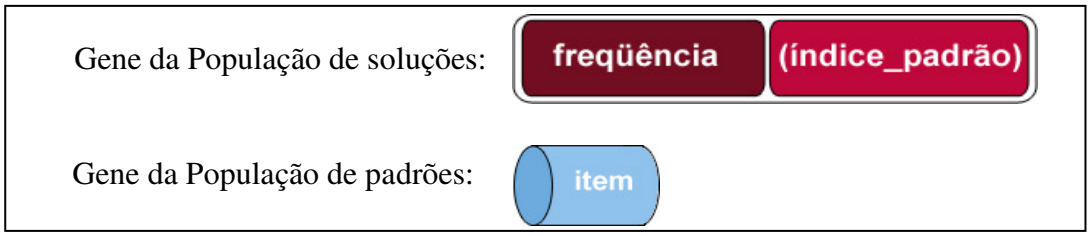



Figura 1 - Representação dos genes

A quantidade máxima de genes do indivíduo solução (tamanho máximo do genoma) adotada foi igual ao número de *setups* (quantidade de itens diferentes). No entanto, se o maior item a ser cortado apresentasse um comprimento inferior ou igual a 50% do comprimento da bobina, adotava-se o seguinte procedimento apresentado na Figura 2, que é uma adaptação de Golfeto et al. (2007):

```
se (Quantidade de itens > 30) entao
        GenesSolucoes = 16
senao
        se (Quantidade de itens > 15) entao
                GenesSolucoes = 12
                        senao
                GenesSolucoes = 8
        fim-se;
fim-se
```

Figura 2 - Número de Genes do Indivíduo Solução

O tamanho do genoma do padrão adotado foi igual ao maior inteiro, menor ou igual ao resultado da divisão entre o tamanho da bobina padrão pelo item de menor tamanho do padrão.

Os itens foram adicionados ao padrão da esquerda para a direita, se somente se, o padrão possuísse espaço livre o suficiente para acomodá-lo (abordagem inspirada nos trabalhos de Golfeto et al. (2007)).

A população de soluções (indivíduos formados por genes da população de soluções) foi formada por 600 indivíduos e a população de padrões (indivíduos formados por genes da população de padrões) por 400, em ambos os casos, todos os indivíduos foram gerados aleatoriamente, sem nenhum direcionamento. Os tamanhos das populações foram definidos experimentalmente, após a realização de simulações com diversos valores.

A estrutura de seleção adotada foi o elitismo com *steady state*. Escolheu-se o elitismo pelo fato desse recurso aumentar o desempenho do algoritmo genético, uma vez que ele garante que a melhor solução encontrada até o momento seja mantida nas próximas gerações. O *steady state* utilizado está disponível no pacote do GALib (Wall, 1996) da classe de algoritmos genéticos GASteadyStateGA. Segundo o tutorial do GALib (Wall, 1996), o método gera, a cada geração, uma população temporária de indivíduos por clonagem e os inserem na população da geração atual, removendo os piores indivíduos. A taxa de substituição, encontrada experimentalmente, foi de 25%, isso equivale a dizer que 75% dos melhores indivíduos da população de soluções serão selecionados e permanecerão na população.

O cálculo do *fitness* (*Fs)* do indivíduo solução para atender o objetivo de minimizar ao mesmo tempo, o número de objetos processados e o *setup*, baseado e descrito em Golfeto et al. (2007), foi realizado do seguinte modo:

$$F_s = c_1 \sum_{j=1}^{n} x_j + \alpha \sum_{j=1}^{n} \delta(x_j) + \sum_{j=1}^{n} \tau(x_j) + \rho \qquad (2)$$

onde:

$\tau$: é a perda relativa e pode ser calculada por $\tau(x_j) = \dfrac{t_j}{w\sum_{j=1}^{n} x_j}$ , sendo $t_j$ o desperdício do padrão *j*.

$\rho$: são penalidades, caso a solução seja inviável. É proporcional à soma das infactibilidades, ou seja, o valor que falta para atingir a demanda do item, multiplicado por 1000 (valor escolhido experimentalmente).

Os valores $c_1$ e $\alpha$, respectivamente, são os custos de objetos processados e *setup*, tratados de maneira explícita na função objetivo, não necessitando quaisquer outras modificações para alcançar objetivos distintos, ou seja, o custo já participa diretamente do processo de minimização da função objetivo.

Para o cálculo da função objetivo uma interpretação dos padrões era realizada, ou seja, verificava-se quais os genes ativos do padrão. Os genes ativos são aqueles que cabem na peça-mestra sem ultrapassar o seu tamanho.

O operador de recombinação (*crossover*) utilizado foi o uniforme, o qual consiste em sortear aleatoriamente um valor entre 0 e 1. Caso o número sorteado fosse menor ou igual a 0,7 a recombinação era realizada com o indivíduo que possuísse o maior *fitness*, caso contrário, era realizada com o pior indivíduo. A taxa de recombinação, encontrada de maneira experimental, foi de 30%.

O operador de mutação adotado consistiu em sortear aleatoriamente a posição no genoma do gene a ser mutado e, em seguida, determinar aleatoriamente qual dos seus elementos (freqüência ou indice_padrao) sofreria mutação. Caso o elemento escolhido do gene fosse a freqüência, um valor entre os limites mínimo e máximo era sorteado, caso contrário, sorteava-se um indivíduo da população de padrão e um ponteiro seria criado. Para o índice do padrão, elemento que compõe o gene da população de solução, os limites foram entre zero e a quantidade máxima de *setups* e para a freqüência, os limites estavam entre zero e o valor da maior demanda (Golfeto et al., 2007). A taxa de mutação foi dada pelo quociente entre 1 e número diferentes de itens (1/*m).*

Com relação ao critério de parada foram utilizados três:

- número máximo de gerações: 1000;
- tempo máximo de execução: 500s;
- convergência: 500 gerações, ou seja, se o algoritmo não melhorar a solução por 500 gerações ele pára.

Apresenta-se, a seguir, o pseudocódigo do método proposto denominado SingleGA10:

**Procedimento SingleGA10**

**1** Gere os indivíduos da população de padrões aleatoriamente;
**2** Gere os indivíduos da população de soluções aleatoriamente;
**3** Calcule a função objetivo dos indivíduos soluções;
**4** Selecione os indivíduos soluções genitores**;**
**5** Utilize operadores de recombinação e mutação para gerar novas soluções;
**6** Evolua a população;
**7 Se** algum critério de parada for satisfeito PARE a execução do algoritmo;
**8 Senão** retorne ao passo 3.

**Fim-Procedimento SingleGA10**

## 5. Testes Computacionais

O método proposto SingleGA10 foi implementado na linguagem C++ no compilador Bloodshed DevC++ num microcomputador Pentium Dual Core 1.73 533MHz, 2G de memória RAM.

Para efeito de avaliação da qualidade do método proposto, na resolução do problema de corte mencionado na seção 2, o mesmo foi comparado com quatro métodos diferentes encontrados na literatura, e descritos a seguir:

I. SHP: programado na linguagem FORTRAN, utilizando o compilador g77 para Linux, num microcomputador AMD AthlonXP 1800 MHz com 512 de memória RAM. Método baseado em uma técnica exaustiva, na qual, a cada iteração, calculam-se alguns parâmetros de aspiração, fazendo uma busca por padrões de corte que satisfaçam tais parâmetros até que se atenda toda a demanda (Haessler, (1975));
II. Kombi234: código em MODULA-2 em MS-DOS 6.0 utilizando um IBM 486/66. Método baseado na combinação de padrões visando diminuir o *setup* de uma solução inicial. Ele tem por base o fato das somas das freqüências dos padrões resultantes serem as mesmas das freqüências dos padrões originais, mantendo assim, o número de objetos processados constante, com uma possível redução de *setup,* ou seja, reduz-se o número de padrões para realização da mesma demanda original (Foerster e Wascher (2000)).
III. ANLCP300: programado na linguagem FORTRAN, utilizando o compilador g77 para Linux, num microcomputador AMD AthlonXP 1800 MHz com 512 de memória RAM. Método proposto por Salles Neto e Moretti (2005), o qual busca minimizar o número de objetos processados e o *setup* através da aplicação, do método lagrangiano aumentado num problema com função objetivo, proposta por Haessler (1975), suavizada. O ANLCP300 adapta o método de geração de colunas de Gilmore e Gomory (1961; 1963) e para a obtenção de uma solução viável utiliza uma simples heurística de arredondamento.
IV. Symbio10**:** implementado na linguagem FORTRAN, utilizando compilador Microsoft FORTRAN Power Station num micro-computador AMD SEMPRON 2300+ 1533MHz com 640 MB de memória RAM. É um método formado por dois algoritmos genéticos que, baseado num processo simbiótico entre duas populações de espécies diferentes, gera seus próprios padrões em conjunto com soluções para o problema (Golfeto et al. (2007)).

Os problemas utilizados para a realização dos testes computacionais com o SingleGA10 foram gerados aleatoriamente pelo CUTGEN1, gerador de problemas de corte unidimensional desenvolvido por Gau e Wascher (1995). Foram geradas 18 classes caracterizadas pelos diferentes valores dos parâmetros de entrada, apresentados na Tabela 1 e Tabela 2, a seguir, contendo cada uma 100 problemas, totalizando 1800 testes para aferir a qualidade do método proposto.

Os parâmetros e a semente usados para gerar os 1800 problemas no CUTGEN1 foram os mesmos utilizados por Foerster e Wascher (2000), Salles Neto e Moretti (2005) e Golfeto et al. (2007).

Nas Tabelas 1 e 2, $v_1$ e $v_2$ representam os limites superior e inferior dos comprimentos dos itens, $m$ representa o número de itens demandados e $d$ indica a demanda média (Gau e Wascher

(1995)).

Tabela 1: Parâmetros de entrada para gerar as classes de 1 a 9

| **Classe** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|
| $v_1$ | 0,01 | 0,01 | 0,01 | 0,01 | 0,01 | 0,01 | 0,01 | 0,01 | 0,01 |
| $v_2$ | 0,2 | 0,2 | 0,2 | 0,2 | 0,2 | 0,2 | 0,8 | 0,8 | 0,8 |
| *M* | 10 | 10 | 20 | 20 | 40 | 40 | 10 | 10 | 20 |
| *D* | 10 | 100 | 10 | 100 | 10 | 100 | 10 | 100 | 10 |

Tabela 2: Parâmetros de entrada para gerar as classes de 10 a 18

| **Classe** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** |
|---|---|---|---|---|---|---|---|---|---|
| $v_1$ | 0,01 | 0,01 | 0,01 | 0,2 | 0,2 | 0,2 | 0,2 | 0,2 | 0,2 |
| $v_2$ | 0,8 | 0,8 | 0,8 | 0,8 | 0,8 | 0,8 | 0,8 | 0,8 | 0,8 |
| *M* | 20 | 40 | 40 | 10 | 10 | 20 | 20 | 40 | 40 |
| *D* | 100 | 10 | 100 | 10 | 100 | 10 | 100 | 10 | 100 |

## 6. Resultados Computacionais

Para diferenciar de outros testes que possam ser realizados em trabalhos futuros com o método proposto, empregou-se a denominação SingleGA10 para os resultados aqui apresentados, o qual utilizou os parâmetros $c_1 = 1$ e custo de *setup* $\alpha = 10$.

Tabela 3: Média dos números de objetos processados

| **Classe** | **SHP** | **Kombi234** | **ANLCP300** | **Symbio10** | **SingleGA10** |
|---|---|---|---|---|---|
| 1 | 14,17 | 11,49 | 14,30 | 14,84 | 15,56 |
| 2 | 116,47 | 110,25 | 121,48 | 117,80 | 127,36 |
| 3 | 25,29 | 22,13 | 25,14 | 28,23 | 32,50 |
| 4 | 255,33 | 215,93 | 224,86 | 239,72 | 269,73 |
| 5 | 46,89 | 42,96 | 45,66 | 61,42 | 71,14 |
| 6 | 433,59 | 424,71 | 432,72 | 518,85 | 623,58 |
| 7 | 55,84 | 50,21 | 51,54 | 53,75 | 54,23 |
| 8 | 515,76 | 499,52 | 495,94 | 514,65 | 518,21 |
| 9 | 108,54 | 93,67 | 114,52 | 101,62 | 104,63 |
| 10 | 1001,59 | 932,32 | 969,20 | 975,72 | 1031,98 |
| 11 | 202,80 | 176,97 | 231,84 | 201,44 | 249,01 |
| 12 | 1873,05 | 1766,20 | 1861,20 | 1917,40 | 2462,56 |
| 13 | 69,97 | 63,27 | 70,54 | 67,32 | 67,63 |
| 14 | 643,55 | 632,12 | 634,02 | 650,52 | 652,43 |
| 15 | 136,05 | 119,43 | 127,38 | 128,87 | 130,16 |
| 16 | 1253,55 | 1191,80 | 1194,74 | 1251,90 | 1301,94 |
| 17 | 256,01 | 224,68 | 297,58 | 247,62 | 270,00 |
| 18 | 2381,54 | 2342,4 | 2430,04 | 2422,41 | 2751,49 |

As Tabelas 3 e 4, respectivamente, apresentam, para efeito de comparação, as médias dos resultados de objetos processados e *setup* para as 18 classes testadas com o SingleGA10 e os métodos SHP, ANLCP300, Kombi234 e Symbio10.

Visando uma análise mais detalhada, calculou-se a variação do *setup* e do número de objetos processados do SingleGA10 em relação aos demais métodos através da expressão representada pela equação (3).

$$100 \times \left( \frac{setup_{SingleGA} - setup_{metodocomparado}}{setup_{SingleGA}} \right) \qquad (3)$$

Analogamente, a expressão para determinar a variação percentual do número de objetos processados é igual a:

$$100 \times \left(\frac{Obj\,\mathrm{Pr}\,oc_{SingleGA} - Obj\,\mathrm{Pr}\,oc_{metodocomparado}}{Obj\,\mathrm{Pr}\,oc_{SingleGA}}\right) \qquad (4)$$

Tabela 4: Média dos números de *setup*

| Classe | SHP | Kombi234 | ANLCP300 | Symbio10 | SingleGA10 |
|---|---|---|---|---|---|
| 1 | 3,95 | 3,40 | 3,02 | 1,85 | 1,97 |
| 2 | 5,94 | 7,81 | 4,50 | 4,68 | 3,78 |
| 3 | 5,00 | 4,84 | 4,84 | 4,47 | 4,18 |
| 4 | 7,31 | 7,28 | 7,28 | 9,36 | 7,10 |
| 5 | 6,87 | 10,75 | 7,02 | 8,23 | 7,79 |
| 6 | 10,81 | 25,44 | 10,92 | 14,08 | 12,36 |
| 7 | 8,84 | 7,90 | 5,54 | 5,21 | 5,35 |
| 8 | 9,76 | 9,96 | 8,00 | 7,76 | 6,07 |
| 9 | 17,19 | 15,03 | 10,58 | 10,01 | 9,39 |
| 10 | 19,37 | 19,28 | 13,96 | 16,04 | 13,34 |
| 11 | 32,20 | 28,74 | 20,00 | 21,56 | 24,02 |
| 12 | 37,25 | 37,31 | 23,68 | 31,52 | 34,58 |
| 13 | 9,38 | 8,97 | 5,64 | 6,68 | 6,56 |
| 14 | 9,85 | 10,32 | 7,92 | 8,21 | 7,22 |
| 15 | 18,03 | 16,88 | 9,92 | 12,99 | 12,90 |
| 16 | 19,63 | 19,91 | 13,88 | 16,34 | 15,33 |
| 17 | 34,39 | 31,46 | 22,60 | 25,62 | 25,86 |
| 18 | 38,23 | 38,28 | 27,44 | 32,18 | 33,76 |

Tabela 5: Variação percentual do *setup* em relação ao SingleGA10 com os demais métodos

| Classe | SHP | Kombi234 | ANLCP300 | Symbio10 |
|---|---|---|---|---|
| 1 | -100,508 | -72,5888 | -53,2995 | 6,091370558 |
| 2 | -57,1429 | -106,614 | -19,0476 | -23,80952381 |
| 3 | -19,6172 | -15,7895 | -15,7895 | -6,937799043 |
| 4 | -2,95775 | -2,53521 | -2,53521 | -31,83098592 |
| 5 | 11,81001 | -37,9974 | 9,884467 | -5,648267009 |
| 6 | 12,54045 | -105,825 | 11,65049 | -13,91585761 |
| 7 | -65,2336 | -47,6636 | -3,5514 | 2,61682243 |
| 8 | -60,7908 | -64,0857 | -31,7957 | -27,84184514 |
| 9 | -83,0671 | -60,0639 | -12,6731 | -6,602768903 |
| 10 | -45,2024 | -44,5277 | -4,64768 | -20,23988006 |
| 11 | -34,055 | -19,6503 | 16,73605 | 10,24146545 |
| 12 | -7,72123 | -7,89474 | 31,52111 | 8,849045691 |
| 13 | -42,9878 | -36,7378 | 14,02439 | -1,829268293 |
| 14 | -36,4266 | -42,9363 | -9,69529 | -13,71191136 |
| 15 | -39,7674 | -30,8527 | 23,10078 | -0,697674419 |
| 16 | -28,0496 | -29,8761 | 9,458578 | -6,58838878 |
| 17 | -32,9853 | -21,6551 | 12,60634 | 0,928074246 |
| 18 | -13,2405 | -13,3886 | 18,72038 | 4,680094787 |

Tabela 6: Variação percentual do número de objetos processados em relação ao SingleGA10 com os demais métodos

| Classe | SHP | Kombi234 | ANLCP300 | Symbio10 |
|---|---|---|---|---|
| 1 | 8,933162 | 26,15681 | 8,097686 | 4,627249357 |
| 2 | 8,550565 | 13,43436 | 4,616834 | 7,506281407 |
| 3 | 22,18462 | 31,90769 | 22,64615 | 13,13846154 |
| 4 | 5,338672 | 19,94587 | 16,63515 | 11,12594076 |
| 5 | 34,08771 | 39,61203 | 35,8167 | 13,66319933 |
| 6 | 30,46762 | 31,89166 | 30,60714 | 16,79495814 |
| 7 | -2,96884 | 7,412871 | 4,960354 | 0,885118938 |
| 8 | 0,472781 | 3,606646 | 4,297486 | 0,686980182 |
| 9 | -3,73698 | 10,47501 | -9,45236 | 2,876803976 |
| 10 | 2,944825 | 9,657164 | 6,083451 | 5,45165604 |
| 11 | 18,55749 | 28,93057 | 6,895305 | 19,10365046 |
| 12 | 23,93891 | 28,27789 | 24,42012 | 22,13793776 |
| 13 | -3,46 | 6,446843 | -4,30282 | 0,45837646 |
| 14 | 1,361066 | 3,112978 | 2,821759 | 0,292751713 |
| 15 | -4,5252 | 8,2437 | 2,135833 | 0,991087892 |
| 16 | 3,716761 | 8,459683 | 8,233866 | 3,843495092 |
| 17 | 5,181481 | 16,78519 | -10,2148 | 8,288888889 |
| 18 | 13,44544 | 14,86794 | 11,68276 | 11,96006527 |

As Tabelas 5 e 6 mostram, respectivamente, a variação percentual do *setup* e números de objetos processados. Fazendo uma análise dessas tabelas, quanto ao *setup,* o SingleGA10 foi melhor que todos os outros métodos na maioria das classes, exceto em relação ao ANLCP300 com o qual empatou em número de classes; superou o SHP em 16 classes; obteve melhores resultados em todas as classes quando comparado com o Kombi234; foi superior ao ANLCP300 em 9 classes; e foi melhor em 12 classes comparando-o com o Symbio10.

Os resultados negativos, obtidos pelas equações (3) e (4), indicam que, naquela classe, o SingleGA10 apresentou resultados melhores do que o método comparado.

As variações apresentadas nas Tabelas 5 e 6 são consideradas válidas, pois apesar de analisar cada objetivo em separado, eles foram tratados simultaneamente (Tabelas 3 e 4) permitindo uma análise da performance de cada parte do método bem como dele como um todo.

Em relação ao número de objetos processados, o desempenho do SingleGA10 foi reduzido, quando comparado com a média dos *setups*. Isso se justifica por se tratar de objetivos inversamente relacionados. No entanto, o SingleGA10 foi superior a 4 classes quando comparado com o SHP e a 3 classes em relação ao ANLCP300. Nas demais classes, mesmo sem apresentar resultados melhores, as diferenças obtidas foram muito pequenas. O SingleGA10, ao ser comparado com o Symbio10 apresentou uma diferença percentual por volta de 1% em 5 classes. A variação percentual do número de objetos processados mostrou que, apesar do desempenho do SingleGA10 ter sido inferior aos demais, essas diferenças percentuais, na maioria das médias das classes do SingleGA10, quando comparados com os outros métodos, não foi muito grande.

O custo total é o custo da função objetivo e seu cálculo foi realizado pela seguinte expressão representada abaixo:

$$custo_{total} = c_1 \times obj_{media} + \alpha \times setup_{media} \qquad (5)$$

onde:

$obj_{media}$ é a média do número de objetos processados entre os 100 problemas de cada classe;

$setup_{media}$ é a média do número de *setup* entre os 100 problemas de cada classe.

Tabela 7: Médias do custo total

| Classe | SHP | Kombi234 | ANLCP300 | Symbio10 | SingleGA10 |
|---|---|---|---|---|---|
| 1 | 53,67 | 45,49 | 44,5 | 33,34 | 35,26 |
| 2 | 175,87 | 188,35 | 166,48 | 164,6 | 165,16 |
| 3 | 75,29 | 70,53 | 73,54 | 72,93 | 74,3 |
| 4 | 328,43 | 288,73 | 297,66 | 333,32 | 340,73 |
| 5 | 115,59 | 150,46 | 115,86 | 143,72 | 149,04 |
| 6 | 541,69 | 679,11 | 541,92 | 659,65 | 747,18 |
| 7 | 144,24 | 129,21 | 106,94 | 105,85 | 107,73 |
| 8 | 613,36 | 599,12 | 575,94 | 592,25 | 578,91 |
| 9 | 280,44 | 243,97 | 220,32 | 201,72 | 198,53 |
| 10 | 1195,29 | 1125,12 | 1108,8 | 1136,12 | 1165,38 |
| 11 | 524,8 | 464,37 | 431,84 | 417,04 | 489,21 |
| 12 | 2245,55 | 2139,3 | 2098 | 2232,6 | 2808,36 |
| 13 | 163,77 | 152,97 | 126,94 | 134,12 | 133,23 |
| 14 | 742,05 | 735,32 | 713,22 | 732,62 | 724,63 |
| 15 | 316,35 | 288,23 | 226,58 | 258,77 | 259,16 |
| 16 | 1449,85 | 1390,9 | 1333,54 | 1415,3 | 1455,24 |
| 17 | 599,91 | 539,28 | 523,58 | 503,82 | 528,6 |
| 18 | 2763,84 | 2725,2 | 2704,44 | 2744,21 | 3089,09 |

Tabela 8: Variação percentual do custo total do SingleGA10, em relação aos demais métodos

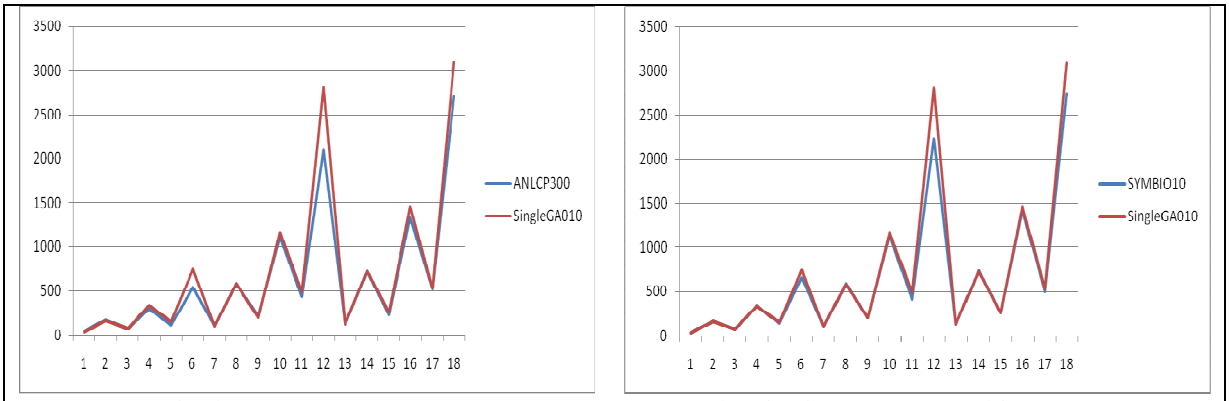
| Classe | SHP | Kombi234 | ANLCP300 | Symbio10 |
|---|---|---|---|---|
| 1 | -52,2121 | -29,01304594 | -26,20533182 | 5,445263755 |
| 2 | -6,48462 | -14,04093001 | -0,799224994 | 0,339065149 |
| 3 | -1,33244 | 5,074024226 | 1,022880215 | 1,843876178 |
| 4 | 3,609896 | 15,26135063 | 12,64050715 | 2,174742465 |
| 5 | 22,44364 | -0,952764359 | 22,26247987 | 3,569511541 |
| 6 | 27,50207 | 9,110254557 | 27,47129206 | 11,71471399 |
| 7 | -33,8903 | -19,93873573 | 0,733314768 | 1,745103499 |
| 8 | -5,95084 | -3,491043513 | 0,513033114 | -2,304330552 |
| 9 | -41,2582 | -22,88822848 | -10,97567118 | -1,606810054 |
| 10 | -2,56654 | 3,454667147 | 4,855068733 | 2,51076902 |
| 11 | -7,27499 | 5,077574048 | 11,72707017 | 14,75235584 |
| 12 | 20,04052 | 23,82386873 | 25,29447792 | 20,50164509 |
| 13 | -22,9228 | -14,81648277 | 4,721158898 | -0,668017714 |
| 14 | -2,40399 | -1,475235637 | 1,574596691 | -1,102631688 |
| 15 | -22,0674 | -11,2170088 | 12,57138447 | 0,150486186 |
| 16 | 0,370386 | 4,421263847 | 8,362881724 | 2,74456447 |
| 17 | -13,4904 | -2,020431328 | 0,949678396 | 4,687854711 |
| 18 | 10,52899 | 11,77984455 | 12,45188713 | 11,16445296 |

O custo total médio foi calculado para aferir a qualidade e desempenho do método proposto tratando, ao mesmo tempo, os dois objetivos para efeito de comparação do valor da função objetivo com os demais métodos. A Tabela 7 traz a média do custo total e para uma melhor compreensão dos seus, apresenta-se, na Tabela 8, a variação percentual do custo total do método

proposto SingleGA10, em relação aos demais métodos comparados, o que torna mais fácil a análise de desempenho do método.

Os resultados obtidos pelo SingleGA10 foram melhores que o SHP em 12 classes e em mais uma, na classe 16, apresentou uma diferença de apenas 0,4%. Superou o Kombi234 em 10 classes, o ANLCP300 em 3 classes, e nesse último, em mais 5 classes apresentou uma diferença inferior a 2%. Teve um desempenho melhor em 4 classes quando comparados com o Symbio10 e em 7 classes a diferença apresentada foi inferior a 3%.

O SingleGA10 foi melhor que o Kombi234 e o SHP na maioria das classes, fato que em relação ao ANLCP300 e Symbio10 não ocorreu. Porém, devido às pequenas diferenças das variações percentuais, analisou-se graficamente o comportamento do SingleGA10 com o ANLCP300 e o Symbio10 (Figura 3), e notou-se que o desempenho praticamente se igualou em 15 classes com o ANLCP300 e a 16 classes quando comparado com o Symbio10. As maiores diferenças apresentadas pelo SingleGA10, em relação a todos os métodos comparados, se encontram nas classes 12 e 18.




a: SingleGA10 X ANLCP300 b: SingleGA10 X Symbio10

Figura 3: Representação gráfica das médias do custo total do SingleGA10 com o ANLCP300 (a) e com o Symbio10 (b)

A escolha do valor do *setup*, também abordado e exemplificado por Salles Neto e Moretti (2005), bem como o do custo do número de objetos processados levam em consideração, diversas variáveis como tempo de entrega, demanda exigida, custo de mão de obra, entre outros fatores.

O tempo computacional não foi considerado neste trabalho, uma vez que cada método foi implementado em linguagem e máquina diferente, tornando inviável a comparação dos mesmos.

## 6. Conclusões

Este trabalho realizou um estudo sobre o Problema de Corte unidimensional com o objetivo de minimizar o número de objetos processados e o *setup,* e o implementou através da aplicação de um novo algoritmo genético que, sozinho, conseguiu resolver esses objetivos, geralmente, conflitantes.

A contribuição deste trabalho foi o desenvolvimento de um novo método capaz de solucionar o problema de minimização proposto, denominado SingleGA10, bem como, prover um estudo comparativo do seu desempenho com os métodos SHP, Kombi234, ANLCP300 e o Symbio10, além da possibilidade de trabalhar com os dois custos, ($c_1$ e $\alpha$), diretamente na função objetivo, que é uma vantagem do método sobre o Kombi234 e o SHP.

Para validar o método proposto foram utilizados 1800 problemas-teste, gerados aleatoriamente pelo gerador de problemas de corte unidimensional, CUTGEN1. Verificou-se que o SingleGA10 foi superior ao Kombi234 e SHP na maioria das classes testadas, e obteve bons resultados em algumas classes quando comparados com o ANLCP300 e o Symbio10.

Quando o *setup* foi o principal objetivo, o SingleGA10 apresentou seu melhor desempenho, em relação à sua minimização, em praticamente todas as classes de todos os métodos, exceto quando comparado ao ANLCP300 com o qual empatou em número de classes.

Observou-se que o excelente desempenho em minimizar o número de *setups*, prejudica a eficiência em minimizar o número de objetos processados na maioria das classes, fato justificado por se tratarem de objetivos inversamente relacionados.

As análises gráficas realizadas mostraram o bom desempenho do método e deixaram mais evidente que são pequenas as variações percentuais para a maioria das classes, quando comparado o SingleGA10 com os outros métodos. No entanto, a grande desvantagem do SingleGA10 em relação aos demais métodos comparados se refere ao tempo computacional. Acredita-se que as escolhas dos parâmetros e os operadores genéticos adotados possam ser os responsáveis pelo alto custo computacional, principalmente o operador de seleção que seleciona apenas os mais aptos, prejudicando assim, a diversidade da população.

E, baseando-se nos resultados computacionais apresentados, pode-se afirmar que o novo método é competitivo e promissor no ambiente de Problemas de Corte unidimensional.

**Referências**


**Carneiro, S. A.**, 1994. Problema de Corte via Algoritmo Genético. Dissertação (Mestrado) Universidade Federal do Espírito Santo, UFES, Brasil.

**Dyckhoff, H., 1990**. A typology of cutting and packing problems. European Journal of Operation Research, vol. 444, pp. 145–159.

**Foerster, H. & Wascher, G.**, 2000. Pattern reduction in one-dimensional cutting-stock problems.International Journal of Prod. Res., vol. 38, pp. 1657–1676.

**Gau, T. e Wascher, G**., 1995. CUTGEN1: A Problem Generator for the Standard One-dimensional Cutting Stock Problem. European Journal of Operational Research, vol. 84, pp: 572-579.

**Gilmore, P. C. & Gomory, R. E.**, 1961. A linear programming approach to the cutting stock problem. Operations Research, vol. 9, pp. 849–859.

**Gilmore, P. C. & Gomory, R. E.,** 1963. A linear programming approach to the cutting stock problem. Operations Research, vol. 11, pp. 863–888.

**Golfeto, R. R., Moretti, A. C., & SallesNeto, L. L**., 2007. Algoritmo genético simbiótico aplicado ao problema de corte unidimensional. In Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional.

**Haessler, R.,** 1975. Controlling cutting pattern changes in one-dimensional trim problems. Operations Research, vol. 23, pp. 483–493.

**Holland**, **J.H**. Outline for a logical theory of adaptive systems. J. Assoc. Comput. Mach., vol. 3, pp. 297-314, 1962.

**Kantorovich, L. V.**, 1960. Mathematical Methods of Organizing and Planning Production. Management Science, vol. 6, pp 366-422.

**Salles Neto, L. L. & Moretti, A. C**., 2005. Modelo não-linear para minimizar o número de objetos processados e o setup num problema de corte unidimensional. In Anais do XXXVII Simpósio Brasileiro de Pesquisa Operacional, pp. 1679–1688.

**Wall, M.,** 1996. A C++ Library of Genetic Algorithm Components, Mechanical Engineering Department Massachusetts Institute of Technology.

**Wäscher, G., Haubner, H., Schumann, H**., 2007 An improved typology of cutting and packing problems. European Journal of Operational Research 183, pp 1109-1130.