# Algoritmo genético para o Problema de Dimensionamento de Lotes Multi-item Capacitado

André Homem Dornas*,
Raíssa Cristina Corrêa Dutra,
Vitor Lemos Silveira,
Rogério Martins Gomes, and
Joao Fernando Machry Sarubbi

Centro Federal de Educação Tecnológica de Minas Gerais
andrehdornas@gmail.com,raissadutra@hotmail.com,
vitorlsilveira10@gmail.com,rogerio@lsi.cefetmg.br,
joaosarubbi@gmail.com

**Resumo** O presente artigo trata sobre o Problema de Dimensionamento de Lotes Monoestágio Multi-item Capacitado, que consiste em determinar a quantidade de itens a serem produzidos em diferentes períodos de tempo a fim de minimizar o custo total de produção, considerando a demanda relativa ao item e a capacidade produtiva do período. O problema foi resolvido através de duas abordagens:um algorítmo genético e um resolvedor CPLEX. Foram realizados experimentos computacionais comparando os resultados obtidos pelo software de otimização e pelo algoritmo genético desenvolvido a partir de instâncias geradas aleatoriamente. Os resultados obtidos mostraram que o algoritmo genético, apesar de não encontrar a melhor solução para o problema na maior parte dos casos, apresentou uma resposta em um tempo substancialmente menor que a do CPLEX. Outra contribuição importante se refere a representação da solução utilizada, que define uma ordem de produção para cada período.

**Keywords:** Pesquisa Operacional. Dimensionamento de Lotes. algoritmo genético.

## 1 Introdução

Ao longo dos anos a indústria manufatureira tenta ampliar seus lucros cada vez mais, e uma forma de alcançar esse objetivo é melhorar a eficiência dos seus processos de produção, diminuindo os custos gerados devido a ineficiência operacional. Deste modo, o Planejamento e Controle de Produção [8] (PCP) se tornou um campo de estudo de bastante interesse para as organizações produtivas, uma

* Os autores gostariam de agradecer ao CEFET-MG e as agências de fomento CAPES, CNPq e FAPEMIG.

vez que o PCP busca determinar o melhor período de produção, bem como a quantidade ótima de itens a ser produzida que minimize o custo total de produção. A partir desse contexto o Problema de Dimensionamento de Lotes (PDL) surgiu para auxiliar o PCP.

O Problema de Dimensionamento de Lotes pode ser visto como o planejamento da produção de uma certa quantidade de itens em um determinado intervalo de tempo com o objetivo de atender a uma certa demanda. Além disso, o PDL pode estar sujeito a algumas restrições, como por exemplo, limitações de capacidade de produção, dependência de processamento prévio (*setup*) ou demandas de outros produtos, bem como geração de estoque desnecessário [4].

O problema em questão pode ser classificado de acordo com os itens produzidos e os estágios, capacidade, horizonte e demanda de produção. Um PDL que trata da produção de apenas um tipo de item é chamado de Único-item, enquanto que o Multi-item trata da produção de diversos tipos de produtos. O PDL multiestágio trata de demandas que dependem de demandas anteriores, ou seja, produtos que necessitam de produtos ou processamentos prévios, e no PDL monoestágio os produtos são independentes uns dos outros [4].

A limitação de mão de obra, matéria-prima e capacidade produtiva de maquinário determina se o PDL será classificado como capacitado ou não. Caso haja alguma limitação, o PDL é capacitado, e caso contrário, é não-capacitado [4].

O horizonte do PDL significa o intervalo de tempo utilizado na produção, que pode ser Finito ou Infinito. Com relação a demanda de produção, o PDL pode ser considerado Estático, isto é, possui demanda definida; Dinâmico, ou seja, a demanda varia com relação ao período; Estocástico, quando o valor da demanda não é necessariamente conhecido, e, caso contrário, Determinístico, isto é, sabe-se ao certo qual o valor da demanda [4].

Este artigo trata da análise do Problema de Dimensionamento de Lotes Capacitado Multi-produto com horizonte finito e demanda dinâmica e determinística, ou seja, será implementada uma solução para a otimização da produção de lotes de diversos tipos de itens, considerando que este processo possui limitações, intervalo de tempo de produção finito e que os valores das demandas de cada produto em cada período são conhecidas e podem variar.

Para solucionar o problema de dimensionamento de lotes foi utilizada uma heurística baseada em algoritmo genético [5], por tratar-se de uma técnica muito utilizada para encontrar soluções satisfatórias em problemas de busca e otimização. Os resultados obtidos utilizando algoritmo genético serão comparados com os valores obtidos utilizando o CPLEX [6].

O presente artigo está dividido da seguinte forma: a Seção 2 apresenta os trabalhos relacionados sobre o assunto abordado; a Seção 3 apresenta a definição e a modelagem matemática do Problema de Dimensionamento de Lotes tratado neste artigo; a Seção 4 detalha a implementação do algoritmo genético desenvolvido; a Seção 5 apresenta os resultados obtidos neste trabalho e a Seção 6 conclui o trabalho.

## 2 Trabalhos Relacionados

Existem diversos modelos matemáticos que tratam o problema de dimensionamento de lotes. Um dos modelos mais conceituados foi proposto por Trigeiro et al. [10], no qual os autores levaram em consideração o tempo de preparação, produção e estocagem, balanceamento de estoque e minimização do custo. Além disso, este modelo leva também em consideração uma restrição de capacidade, ou seja, a quantidade de itens que podem ser produzidos num dado período é limitada. Trigeiro et al. [10] também foram pioneiros ao apresentarem uma heurística capaz de resolver o tempo de *setup*, bem como de mostrar que a inserção dessa limitação pode gerar soluções infactíveis.

Rocha Junior [9] propôs uma aplicação do modelo matemático para o problema de dimensionamento de lotes multi-item multiestágio e com demanda dinâmica determinística para uma indústria de móveis. O modelo foi testado utilizando dados reais de uma empresa do setor moveleiro de Arapongas, Paraná. Os resultados mostraram que o modelo matemático proposto obteve um desempenho superior à metodologia de produção usada na empresa, demonstrando sua aplicabilidade.

Haase e Kohlmorgen [3] propuseram um algoritmo genético para resolver um problema de dimensionamento de lotes com restrição de capacidade. Os autores introduziram a ideia de começar a definir a quantidade de cada item produzido partindo do último período, de forma determinística. Esse algoritmo utiliza conceitos de **demanda acumulada** e **capacidade disponível** que são responsáveis, respectivamente, por transferir a demanda não produzida para períodos anteriores (gerando estoque) e pelo gerenciamento da capacidade de cada período.

Karimi et al. [7] estudaram a fundo o problema de dimensionamento de lotes, além de abordarem as principais características e variações do problema, propondo estratégias de soluções. Goren et al. [2], além de mostrarem características e algumas variações do PDL, fazem uma breve revisão de outros trabalhos sobre esse problema que foram resolvidos utilizando Algoritmos Genéticos.

## 3 Definição do problema

Dado um conjunto N de itens e um conjunto T de períodos, o Problema de Dimensionamento de Lotes (PDL) consiste em definir a quantidade de cada item $i \in N$ a ser produzido em cada período $t \in T$. A variação do PDL abordada no presente artigo consiste no Problema de Dimensionamento de Lotes Capacitado (PDLC), no qual é levado em consideração a capacidade de produção $CAP_t$ em cada período $t$. A produção de um item $i$ requer $b_i > 0$ unidades de capacidade e custa $c_{it}$ por unidade. A demanda no item $i$ no período $t$ é dada por $d_{it} > 0$, que deve ser satisfeita sem atraso. Além disso, a produção do item $i$ no período $t$ requer a ativação (*setup*) da máquina, que possui um custo $S_{it}$ e um $s_i > 0$ unidades de capacidade. O custo de estocagem do item $i$ em $t$ é dado por $H_{it}$. Dessa forma, o objetivo do PDLC é reduzir o custo total de produção [9].

$$\min \quad \sum_{i \in N} \sum_{t \in T} (H_{it} I_{it} + c_{it} X_{it} + S_{it} Y_{it}) \tag{1a}$$

s.t.

$$I_{i,t-1} + X_{it} - I_{it} = d_{it}, \qquad i \in N; t \in T, \tag{1b}$$

$$\sum_{i \in N} (b_i X_{it} + s_i Y_{it}) \leq CAP_t, \quad t \in T, \tag{1c}$$

$$X_{it} - M Y_{it} \leq 0, \qquad i \in N; t \in T, \tag{1d}$$

$$I_{i0} = 0, \qquad i \in N, \tag{1e}$$

$$X_{it}, I_{it} \in R_+, \qquad i \in N; t \in T, \tag{1f}$$

$$Y_{it} \in \{0, 1\}, \qquad i \in N; t \in T \tag{1g}$$

**Figura 1.** Modelo do PDLC [10]

O Modelo mostrado na Figura 1, baseado no modelo de otimização do PDLC proposto por Trigeiro et al. [10], propõe a minimização dos custos de produção, a partir da função objetivo 1a. As variáveis de decisão utilizadas são: $X_{it}$, que representa a quantidade do item $i$ produzida no período $t$; $Y_{it}$, que indica se há produção do item $i$ no período $t$; e $I_{it}$, que indica a restrição de estoque do item $i$ no período $t$. As restrições 1b estão relacionadas ao balanço de estoque, enquanto que a restrição 1e define que o estoque inicial é igual a zero. As restrições 1c definem o limite de capacidade de produção em cada período de tempo. A restrição 1d, na qual M representa um número muito grande, limita que a preparação (tempo e custo) só seja considerada caso haja produção. As restrições 1f e 1g definem a não negatividade e integralidade das variáveis.

*Resolução do PDLC a partir do modelo.* O PDLC será resolvido, inicialmente, utilizando o software de otimização CPLEX. O CPLEX [6] é um software de otimização desenvolvido pela IBM que pode ser utilizado para resolução de problemas de programação linear complexos, tendo o objetivo de encontrar a solução ótima para o problema.

Para a utilização do CPLEX, cada instância gerada foi transformada em um arquivo LP. Este formato de arquivo é legível ao CPLEX e representa um programa linear, contendo a definição do problema (restrições e função objetivo) obtidas a partir do modelo 1. Da mesma forma, o PDLC será resolvido utilizando uma heurística computacional, o algoritmo genético, que será descrita a seguir.

## 4 Algoritmo genético

A heurística escolhida para resolver o problema proposto foi o Algoritmo Genético (AG), ilustrado pelo Algoritmo 1. Para o presente artigo, o algoritmo tem sua execução interrompida quando a melhor solução encontrada se mantém

inalterada por 100 iterações. De forma a ajudar na convergência do AG, foi utilizado o elitismo, ou seja, a melhor solução de uma geração é mantida na próxima geração sem sofrer alteração.

**Algoritmo 1:** Algoritmo_Genético

```
Data: txCruz, txMut, qntMut, T
populacao ←GeraPopulacaoInicial();
Avaliar(populacao);
t ← 0 ;
while critério não atingido do
    elite ← Retira_Melhor(populacao)/* armazena o melhor indivíduo    */
    pai1, pai2 ← Selecao(populacao);
    populacao ← populacao + Cruzamento(pai1, pai2, txCruz);
    Mutacao(populacao, T, txMut, qntMut);
    populacao ← populacao + elite/* retorna o melhor indivíduo        */
    Avaliar(populacao);
    t ← t + 1;
end
```

### 4.1 Codificação

Um indivíduo, ou cromossomo, equivale a uma solução do PDLC. Dessa forma, cada indivíduo é representado por uma matriz N x T, onde cada coluna contém uma ordem de produção para o período $t$. Essa escolha foi definida uma vez que a capacidade de produção de cada período e a necessidade da preparação para produção limitam a quantidade de cada item a ser produzido, dessa forma, foi necessário definir uma preferência de produção.

### 4.2 Decodificação

O método utilizado para desenvolvimento do Algoritmo 2, responsável por decodificar os cromossomos, foi baseado no método proposto por Haase e Kohlmorgen [3], no qual o preenchimento da matriz de produção é feito a partir do último período. O autor utiliza de dois conceitos importantes, os quais foram utilizados neste trabalho, quais sejam: **Capacidade Disponível**, que verifica quanto ainda é possível produzir no período corrente; e **Demanda Acumulada**, que é uma matriz que controla a quantidade de itens que não foi possível produzir em um período com o propósito de definir a quantidade que deveria ter sido produzida no período anterior.

### 4.3 Avaliação da população

Para avaliar a população foi considerado que a função de aptidão é inversamente proporcional ao resultado da função objetivo, ou seja, quando o valor encontrado

**Algoritmo 2:** Definição da produção

```
Data: ordens, d, b, s, CAP
dac ← d; /* ajusta a demanda acumulada                          */
ACAP ← CAP; /* ajusta a capacidade disponível de cada período  */
t ← T ;
/* preenche a partir do último período                         */
while t ≥ 1 do
    ordem ← ordens[t]; /* ordem de produção do período         */
    foreach i in ordem do
        /* tenta produzir toda a demanda                       */
        if b_i * dac_it + s_i ≤ ACAP_t then
            X_it ← dac_it;
        else
            /* produz o máximo possível                        */
            max ← floor((ACAP_t − s_i)/b_i);
            if max < 0 then max ← 0;
            X_it ← max;
        end
        if X_it > 0 then Y_it ← 1;
        /* atualiza dados                                      */
        dac_{i,t−1} ← dac_{i,t−1} + dac_it − X_it;
        dac_it ← 0;
        ACAP_t ← b_i * X_it + s_i * Y_it;
        /* balanceamento de estoque                            */
        I_it = I_{i,t+1} − X_{i,t+1} + d_{i,t+1};
        t ← t − 1;
    end
end
```

na função objetivo for muito alto, a aptidão do indivíduo será pequena. Isto significa que este indivíduo não produzirá uma boa solução para o problema, uma vez que o objetivo é minimizar o custo total do planejamento de produção e estocagem de itens.

### 4.4 Operadores

Como foi desenvolvida uma nova solução para o problema, foi necessário criar operadores de seleção, cruzamento e mutação próprios. Esses operadores foram baseados em outros já existentes e são apresentados a seguir.

**Seleção** O algoritmo de seleção tem como objetivo selecionar dois indivíduos da população para sofrer cruzamento e gerar dois outros indivíduos que serão inseridos na população. O algoritmo em questão foi baseado no Torneio, um algoritmo de seleção bastante conhecido e utilizado.

O algoritmo de torneio implementado (Algoritmo 3) realiza quatro duelos entre dois indivíduos, escolhidos aleatoriamente. O indivíduo que possui a maior

aptidão vence o duelo e é inserido em uma subpopulação de vencedores. Posteriormente, os 2 indivíduos da subpopulação com maior aptidão são escolhidos como os vencedores do torneio, passando para a próxima etapa do algoritmo genético, o cruzamento.

**Algoritmo 3:** Seleção

```
Data: populacao
torneio ← ∅;
for i ← 0 to 4 do
    c1 ← SelecionaRand(populacao);
    c2 ← SelecionaRand(populacao);
    /* Armazena o melhor entre c1 e c2                    */
    torneio ← torneio + Melhor(c1, c2);
end
/* retorna os dois melhores indivíduos entre os quatro    */
return Melhores(torneio)
```

**Cruzamento** O algoritmo de cruzamento tem como finalidade gerar um ou dois indivíduos a partir das características de dois outros indivíduos selecionados anteriormente. A taxa de cruzamento utilizada neste trabalho foi de $txCruz = 0.75$.

O algoritmo implementado foi baseado no operador *Ordered Crossover* (OX), proposto por Davis [1]. Como a Figura 2 apresenta, o operador OX produz dois indivíduos filhos utilizando uma subsequência dos indivíduos pais. Neste exemplo, o Filho 1 recebe a subsequência do Pai 2 enquanto o Filho 2 recebe a subsequência do Pai 1. Após receber a subsequência, as outras posições dos vetores filhos são preenchidas com relação aos seus respectivos pais. Caso um um elemento do pai já estiver no vetor do filho, o próximo elemento do vetor pai será considerado, respeitando a ordem em que aparece no indivíduo.

Um detalhe interessante dessa implementação é que como o indivíduo é composto por um vetor de períodos, que possui uma sequência de itens a ser produzida, o cruzamento entre os indivíduos pais ocorre período a período, ou seja, o primeiro período do primeiro indivíduo pai é cruzado com o primeiro período do segundo indivíduo, e assim por diante. Esse processo é ilustrado pelo Algoritmo 4.

**Mutação** No algoritmo genético, a Mutação tem por finalidade manter a diversidade da população gerando perturbações aleatórias nos indivíduos. A taxa de mutação utilizada neste trabalho foi de $txMut = 0.025$.

Assim como no Algoritmo de Cruzamento, a mutação em cada indivíduo ocorre período a período, sendo realizada somente em apenas $qntMut = 10\%$ dos

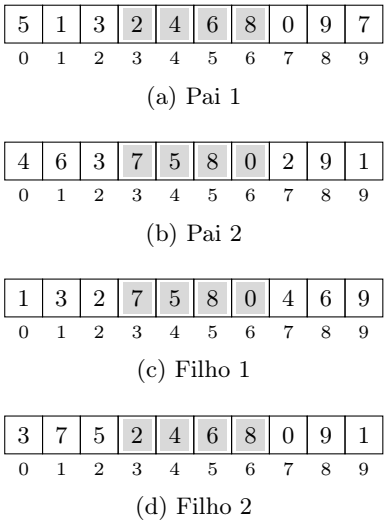


**Figura 2.** Exemplo do cruzamento OX (*Ordered Crossover*)

períodos, escolhidos aleatoriamente. Como ilustrado no Algoritmo 5, a mutação realiza apenas a trocar da ordem de produção de dois itens do período.

## 5 Testes computacionais

Para a realização dos testes foram geradas instâncias de acordo com o que foi proposto por Trigeiro et al.[10], levando em consideração seus parâmetros para número de períodos, itens, intervalo de demanda, custo, etc. Foi definido um fator F na geração das instâncias que determina a relação da capacidade de produção com a demanda, ou seja, um fator igual a 1.0 representa uma capacidade de produção igual à demanda, enquanto que F igual a 0.9, representa uma folga de 10% na capacidade de produção. Desta forma, assim como ilustra a Tabela 1, foram geradas 32 instâncias para serem testadas.

Neste trabalho, foi utilizado a *API* do CPLEX para *Python* na versão 12.7.1 e o algoritmo genético foi implementado utilizando a linguagem *Python* 2.7. Os testes foram realizados em uma máquina com sistema operacional Ubuntu, 8GB de RAM e processador i7-75000U (2.70GHz x 2).

Neste trabalho o maior tempo de execução para o CPLEX foi de 30 min (1800 segundos), pois foi estipulado que este seria o tempo máximo de execução para que o CPLEX encontre a solução ótima para o problema. Quando esta solução não é encontrada no tempo estipulado, é gerado um *Gap* que representa o quanto faltou para que o CPLEX encontrasse a solução ótima. Este *Gap* está ilustrado na oitava coluna da Tabela 1, que também apresenta o *Gap* de Relaxamento Linear do CPLEX (*Gap* RL) e o *Gap* do algoritmo genético, que representa o quanto o resultado do AG se aproximou do melhor valor encontrado pelo CPLEX.

**Algoritmo 4:** Cruzamento

```
Data: pai1, pai2, txCruz
if random(0, 1) ≤ txCruz then
    filho1 ← pai1 ;
    filho2 ← pai2 ;
    for t ← 0 to T do
        o1 ← filho1.ordem[t];
        o2 ← filho2.ordem[t];
        /* crossover para o período t                    */
        CruzamentoOX(o1, o2);
    end
    return filho1, filho2
end
```

**Algoritmo 5:** Mutação

```
Data: populacao, T, txMut, qntMut
foreach c ∈ populacao do
    /* checa a taxa de mutação                           */
    if random(0, 1) ≤ txMut then
        /* define número de trocas                       */
        q ← qntMut ∗ T;
        for i ← 0 to q do
            t ← random(0, T);
            /* troca dois itens de ordem                 */
            TrocaOrdem(c, t);
        end
    end
end
```

Para cada uma das instâncias geradas, o algoritmo genético foi executado 11 vezes e o resultado, apresentado na Tabela 1, foi obtido por meio da média dos resultados obtidos em cada execução. A convergência do algoritmo genético pode ser observada na Figura 3, utilizada como exemplo, para uma instância específica.

Como é possível observar na Tabela 1 e na Figura 4, quando comparado ao CPLEX, o tempo gasto pelo algoritmo genético para encontrar a solução do problema se aproxima do tempo gasto pelo CPLEX e, em alguns momentos, quando o CPLEX não conseguiu encontrar a solução ótima para o problema, o tempo gasto foi significativamente menor. No entanto, as soluções encontradas pelo CPLEX, em geral, podem ser consideradas melhores que as soluções encontradas pelo algoritmo genético.

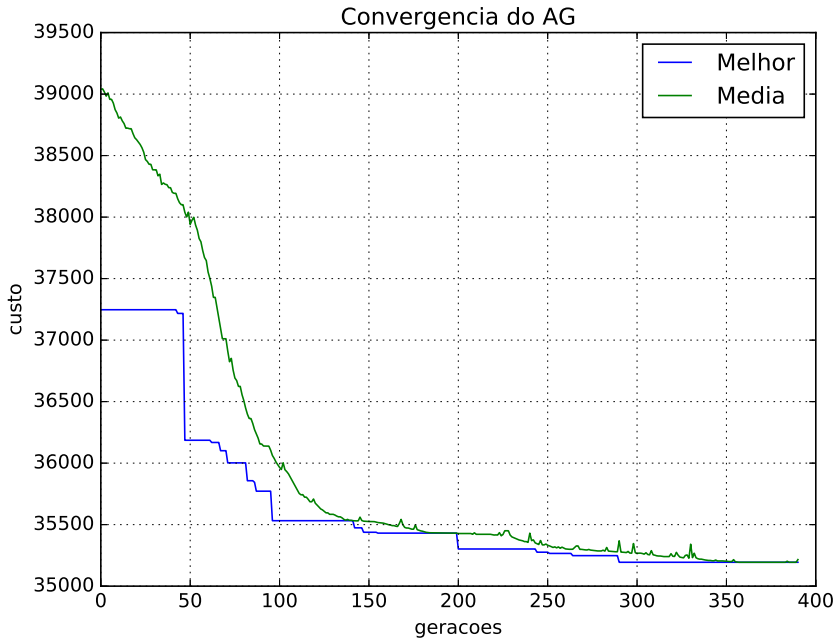


**Figura 3.** Gráfico de convergência do algoritmo genético para a instância com 6 itens, 15 períodos e fator 1.0

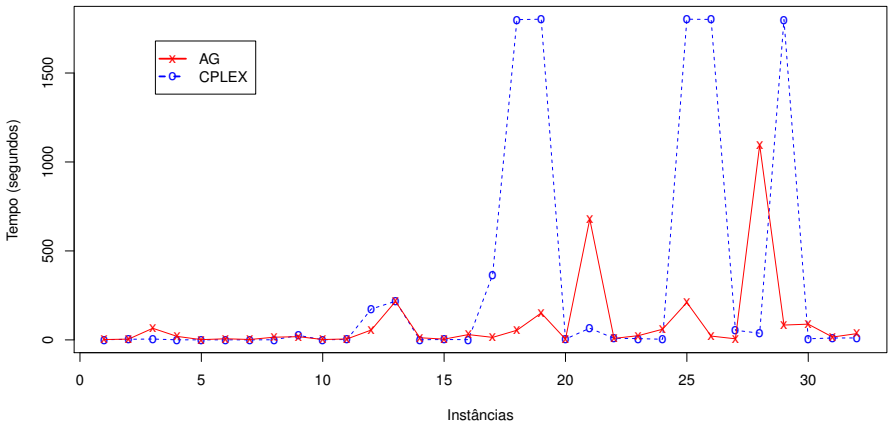


**Figura 4.** Comparação dos tempos de execução do AG e do CPLEX para cada instância

| N | T | F | Valor AG | Valor CPLEX | Tempo AG | Tempo CPLEX | Gap CPLEX | Gap RL | Gap AG |
|---|---|---|---|---|---|---|---|---|---|
| 6 | 15 | 0.9 | 38049.7 | 31479.0 | 1.33 | 0.16 | 0 | 62% | 17% |
| 6 | 15 | 1.0 | 35152,6 | 31046.0 | 1.93 | 0.78 | 0 | 61% | 11% |
| 6 | 30 | 0.9 | 75987.0 | 66913.0 | 2.09 | 0.97 | 0 | 61% | 12% |
| 6 | 30 | 1.0 | 75618,9 | 66494.0 | 5.25 | 2.13 | 0 | 60% | 12% |
| 6 | 60 | 0.9 | 158582.0 | 136239.0 | 4.10 | 4.50 | 0 | 61% | 14% |
| 6 | 60 | 1.0 | 151529,8 | 133463.0 | 14.10 | 366.35 | 0 | 60% | 12% |
| 6 | 90 | 0.9 | 238480.0 | 206168.0 | 6.09 | 54.00 | 0 | 61% | 14% |
| 6 | 90 | 1.0 | 235347,9 | 203492.0 | 21.41 | 1801.18 | 0,34% | 61% | 13% |
| 12 | 15 | 0.9 | 71434.0 | 57325.0 | 2.27 | 0.16 | 0 | 58% | 20% |
| 12 | 15 | 1.0 | 65483,7 | 56865.0 | 4.35 | 3.50 | 0 | 63% | 13% |
| 12 | 30 | 0.9 | 155473.0 | 128978.0 | 4.22 | 2.10 | 0 | 61% | 17% |
| 12 | 30 | 1.0 | 144816.0 | 129285.0 | 18.12 | 25.50 | 0 | 61% | 11% |
| 12 | 60 | 0.9 | 310151.0 | 264312.0 | 8.34 | 8.83 | 0 | 60% | 15% |
| 12 | 60 | 1.0 | 308864.2 | 267413.0 | 54.13 | 1800.01 | 0,05% | 60% | 13% |
| 12 | 90 | 0.9 | 478580.0 | 403929.0 | 15.99 | 1.0.66 | 0 | 61% | 16% |
| 12 | 90 | 1.0 | 475496,7 | 407403.0 | 83.35 | 1800.01 | 0,02% | 60% | 14% |
| 24 | 15 | 0.9 | 144705,8 | 117042.0 | 6.29 | 0.38 | 0 | 58% | 19% |
| 24 | 15 | 1.0 | 140838,8 | 116706.0 | 12.64 | 0.41 | 0 | 59% | 17% |
| 24 | 30 | 0.9 | 307752.0 | 257045.0 | 12.09 | 0.76 | 0 | 60% | 16% |
| 24 | 30 | 1.0 | 295638.1 | 256205.0 | 55.51 | 173.46 | 0 | 59% | 13% |
| 24 | 60 | 0.9 | 624407.0 | 530046.0 | 23.74 | 7.00 | 0 | 59% | 15% |
| 24 | 60 | 1.0 | 628995.3 | 541554.0 | 151.06 | 1802.44 | 0,01% | 60% | 14% |
| 24 | 90 | 0.9 | 961230.0 | 808864.0 | 35.20 | 1.0.83 | 0 | 61% | 16% |
| 24 | 90 | 1.0 | 935312.2 | 810103.0 | 210.19 | 1802.49 | 0,03% | 60% | 13% |
| 48 | 15 | 0.9 | 292869.0 | 232134.0 | 15.15 | 0.60 | 0 | 60% | 20% |
| 48 | 15 | 1.0 | 278649,8 | 231654.0 | 65.25 | 4.75 | 0 | 60% | 17% |
| 48 | 30 | 0.9 | 628585.0 | 520055.0 | 29.79 | 1.22 | 0 | 60% | 17% |
| 48 | 30 | 1.0 | 593767,9 | 507526.0 | 216.19 | 217.17 | 0 | 60% | 14% |
| 48 | 60 | 0.9 | 1244645.0 | 1.054600.0 | 59.85 | 3.50 | 0 | 60% | 15% |
| 48 | 60 | 1.0 | 1242838.2 | 1.066552.0 | 678.57 | 65.64 | 0 | 60% | 14% |
| 48 | 90 | 0.9 | 1907492.0 | 1616532.0 | 88.41 | 6.20 | 0 | 60% | 15% |
| 48 | 90 | 1.0 | 1886558,7 | 1628269.0 | 1095.21 | 37.64 | 0 | 60% | 14% |

**Tabela 1.** Comparação entre os resultados obtidos, os tempos de execução (em segundos) e os *gaps* utilizando o CPLEX e o algoritmo genético

## 6 Conclusão

A partir do que foi analisado na seção anterior, o algoritmo genético desenvolvido para o problema se mostrou eficiente, em relação ao tempo de execução do CPLEX, quando testado em instâncias grandes, uma vez que o seu tempo de execução foi significativamente menor que o tempo de execução do CPLEX. No entanto, com relação a solução encontrada, o algoritmo genético gerou soluções que são no mínimo 11% piores que as soluções encontradas pelo CPLEX, mesmo que esse tenha um tempo de execução muito maior em alguns momentos. Portanto, é possível concluir que o algoritmo genético proposto não foi mais eficiente que o CPLEX em encontrar a solução ótima para o Problema de Dimensionamento de Lotes.

Faz-se necessário ressaltar a relevância desse estudo com relação a representação da solução em formato de ordem de produção dos itens em cada período, que foi baseada na representação de outros problemas, mas pode ser uma forma pioneira para o Problema de Dimensionamento de Lotes tratado neste artigo. Visto os resultados que foram obtidos, essa nova representação se mostrou eficiente e também promissora, já que é possível aprimorar esse estudo inicial procurando melhorar os resultados.

Como trabalhos futuros, novos operadores para o algoritmo genético serão desenvolvidos de forma a verificar se algum se adapta melhor ao PDLC. Além disso, novos testes serão realizados em outros tipos de instâncias, tanto geradas aleatoriamente quanto em instâncias reais.

## Referências

1. Davis, L.: Applying adaptive algorithms to epistatic domains. In: IJCAI. vol. 85, pp. 162–164 (1985)
2. Goren, H.G., Tunali, S., Jans, R.: A review of applications of genetic algorithms in lot sizing. Journal of Intelligent Manufacturing 21(4), 575–590 (2010)
3. Haase, K., Kohlmorgen, U.: Parallel genetic algorithm for the capacitated lot-sizing problem. In: Operations Research Proceedings 1995, pp. 370–375. Springer (1996)
4. Hajipour, V., Fattahi, P., Nobari, A.: A hybrid ant colony optimization algorithm to optimize capacitated lot-sizing problem. Journal of Industrial and Systems Engineering 7(1), 1–20 (2014)
5. Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)
6. IBM: IBM ILOG CPLEX Optimization Studio CPLEX User's Manual
7. Karimi, B., Ghomi, S.F., Wilson, J.: The capacitated lot sizing problem: a review of models and algorithms. Omega 31(5), 365–378 (2003)
8. Lustosa, L.J., de Mesquita, M.A., Oliveira, R.J.: Planejamento e controle da produção. Elsevier Brasil (2008)
9. Rocha Junior, W.R.d.: Programação de pedidos e dimensionamento de lotes em uma indústria de móveis (2016)
10. Trigeiro, W.W., Thomas, L.J., McClain, J.O.: Capacitated lot sizing with setup times. Management science 35(3), 353–366 (1989)