**PRO5826 - Estudo de Meta-heurísticas para Problemas de Produção**
**RELATÓRIO DE PROJETO – HEURÍSTICA MELHORIA**

Filipe Aécio Alves de Andrade Santos NUSP 5695721
Igor Henrique de Freitas NUSP 12477186

**SUMÁRIO**



## 1 INTRODUÇÃO

O problema de programação de tarefas com data de entrega comum com penalização de atraso e adiantamento vem sendo estudados há décadas. De acordo com Jarboui et al (2013) este problema ganhou destaque com o surgimento do conceito de manufatura enxuta que surgiu no Japão. Como máquinas nos processos industriais frequentemente são gargalos nas linhas de produção o estudo para encontrar a melhor programação que minimize as penalizações de atraso e adiantamento são de grande importância (Jarboui et al, 2013).

Biskup & Feldmann (2001) realizaram um estudo para este problema de programação de tarefas com data de entrega comum com penalização de atraso e adiantamento analisando 280 diferentes problemas com tamanhos variados de trabalhos (10, 20, 50, 100, 200, 500, 1000). Os autores propuseram duas heurísticas construtivas que geraram benchmarks para futuras pesquisas com este problema.

A fim de encontrar resultados para comparar com os valores dos 280 problemas proposto por Biskup & Feldmann (2001) o presente relatório apresenta uma proposta de heurística construtiva e uma proposta de melhoria para uma das heurísticas proposta por Biskup & Feldmann (2001).

## 2 DESCRIÇÃO DO PROBLEMA

Existem **n** tarefas que devem ser processadas em uma única máquina com um prazo de entrega comum **d**. Cada tarefa necessita de apenas uma operação. São conhecidos os tempos de execução $\boldsymbol{p_i}$ de cada tarefa. O atraso da tarefa $\boldsymbol{T_i}$ é dado por $\max(C_i - d, 0)$ e o adiantamento $\boldsymbol{E_i}$ é dado por $\max(d - C_i,\ 0)$. A penalidade por unidade de tempo para tarefas realizadas antes do prazo é $\boldsymbol{a_i}$ e para tarefas realizadas após o prazo é $\boldsymbol{b_i}$. Sejam $\boldsymbol{s_i}$ o tempo de início da tarefa i, $\boldsymbol{x_{ik}} = 1$ se a tarefa i ocorre antes da tarefa k e 0 caso contrário e **R** um número grande. Biskup & Feldmann (2001) apresentam a formulação a seguir em que a função objetivo busca minimizar as penalidades e é dada pela equação (1).

$$\min \sum_{i=1}^{n} a_i * E_i + \sum_{i=1}^{n} b_i * T_i \tag{1}$$

$$T_i \geq s_i + p_i - d, \forall\ i \tag{2}$$

$$E_i \geq d - s_i - p_i, \forall\ i \tag{3}$$

$$s_i + p_i \leq s_k + R(1 - x_{ik}), i = 1, \dots, n-1;\ \ k = i+1, \dots, n \tag{4}$$

$$s_k + p_k \leq s_i + R * x_{ik}, i = 1, \dots, n-1; k = i+1, \dots, n \tag{5}$$

$$T_i, E_i, s_i \geq 0\ \forall\ i \tag{6}$$

$$x_{ik} \in \{1\ , 0\}, \qquad i = 1, \dots, n-1;\ \ k = i+1, \dots, n \tag{7}$$

As restrições (2) e (3) definem os valores de $T_i$ e $E_i$. As restrições (4) e (5) estabelecem o temo de início das tarefas. E as restrições (6) e (7) definem o espaço das variáveis.

Para Biskup & Feldmann (2001) como o problema possui grande complexidade a formulação apresentada só é útil para problemas com pequenos números de tarefas. Jarboui et al (2013) e Biskup & Feldmann (2001) apresentam três propriedades para a solução ótima do problema de programação de tarefas com data de entrega comum com penalização de atraso e adiantamento.

**Propriedade 1:** Há uma solução ótima em que os trabalhos que são completados antes da data de entrega são ordenados em ordem decrescente de p/a e os trabalhos que são completados após a data de entrega são ordenados em ordem crescente de p/b.

**Propriedade 2:** Existe uma solução ótima em que ou o primeiro trabalho inicia no tempo zero ou um trabalho termina exatamente na data de entrega.

**Propriedade 3:** Em uma solução ótima não existe intervalo entre e execução de duas tarefas sucessivas.

Analisando essas três propriedades podemos dizer que o problema consiste em determinar quais tarefas serão finalizadas antes da data de entrega e quais serão finalizadas após, pois definindo esses dois grupos só existirá uma solução que respeite as três propriedades.

# 3 HEURÍSTICA PROPOSTA

## 3.1 HEURÍSTICA CONSTRUTIVA

A heurística construtiva proposta foi construída a partir do entendimento do problema e das instâncias, e posteriormente afinada de acordo com sugestões e informações presentes no artigo escrito por Biskup & Feldmann (2001) e em Jarboui et al (2013).

Foram feitas pequenas mudanças em cima da heurística proposta previamente, pois a heurística construtiva proposta anteriormente era muito rígida, e apresentava resultados pouco satisfatórios o que dificultaria as possibilidades de melhoria coma a implementação de uma heurística de melhoria. Com as alterações foi possível reduzir a diferença do benchmark de 25,1% para 3,1%.

Seguindo as propriedades mencionadas anteriormente, é proposto uma sequência que pode ter uma tarefa começando em tempo zero ou terminando no prazo de entrega (propriedade 2), sem intervalos entre as tarefas (propriedade 3) e mantendo a ordenação informada na propriedade 1.

Seguem os passos:

- Primeiro, as tarefas são separadas em dois conjuntos de acordo com as penalidades de término antecipado ou atrasado, portanto se $a < b$, a tarefa irá para o primeiro conjunto (chamado de *early candidates*, que são as tarefas candidatas a serem antecipadas), e caso a regra proposta seja falsa, ou seja $a > b$, a tarefa irá para o segundo conjunto de tarefas (chamado de *late candidates*, que são as tarefas candidatas a serem atrasadas);
- As tarefa do conjunto *early candidates* são ordenadas pela pena proporcional, *(b-a)/a*, de ordem decrescente, para que as tarefas que tem um peso de atraso muito maior do que a pena de antecipação sejam priorizadas nas etapas seguintes e se mantenham no conjunto *early candidates*;
- Com o grupo *early candidates* ordenado pela etapa anterior, é feito a verificação se o término da execução das tarefas deste grupo é maior do que o prazo, $\sum pe > \sum p*h$ (onde pe é o prazo das tarefas do grupo de *early candidates*), se o prazo for estourado, a última tarefa do conjunto é movida para o conjunto *late candidates* recursivamente até o prazo ser respeitado;

- Com a atribuição das tarefas nos dois grupos finalizada, é feito a ordenação em ordem decrescente do conjunto *early candidates* de acordo com *p/a*, e a ordenação do conjunto *late candidates* é executada de forma crescente por *p/b*. Nesta etapa respeitamos a propriedade 1;
- Se a primeira tarefa não inicia no tempo zero todas as tarefas são deslocadas para permitir que a primeira tarefa inicie em zero. O custo desta nova sequência é comparado com da sequência inicial, e se for menor as tarefas são reordenadas obedecendo a propriedade 1 e essa nova sequência é considerada.
- Agora concatena-se os dois conjuntos (*early candidates* seguido consecutivamente do *late candidates*) e tem-se a ordenação final proposta.

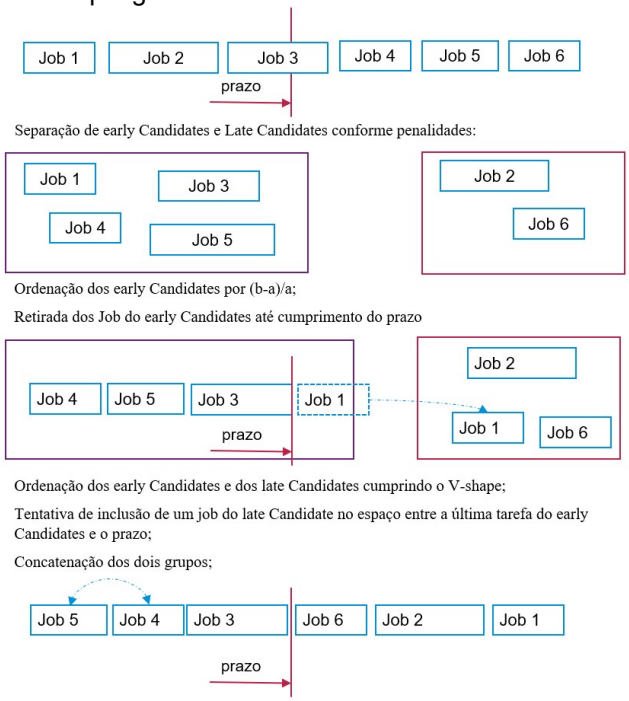
Segue pseudocódigo do fluxo proposto acima:

**Algoritmo Heurística Construtiva**

**begin**
S ← *initialize tasks;*
h ← *factor;*
$d = h * \sum_{i=0}^{n} p_i$
**for** i ∈ S **do:**
**if** $a_i < b_i$ **then:**
$E \leftarrow E + \{i\}$;
**else:**
$T \leftarrow T + \{i\}$;
sort (E ; $(b_i - a_i)/a_i$ ; decreasing);
**while** $\sum_{i=0}^{n} p_i > d \; \forall \; i \in E$ **do**:
$E \leftarrow E - \{i\}$;
$T \leftarrow T + \{i\}$;
sort (E ; $p_i/a_i$ ; decreasing);
sort (T ; $p_i/b_i$ ; increasing);
$d_0 = d - \sum_{i=0}^{n} p_i > d \; \forall \; i \in E$;
$S' \leftarrow E + T$ *such that* $d_i = p_i + p_{i-1} + d_0 \; \forall \; i \in S'$;
$S'' \leftarrow E + T$ *such that* $d_i = p_i + p_{i-1} \; \forall \; i \in S''$;
**if** $f(S') < f(S'')$ **then:**
$d_0 = 0$
**end**

A figura abaixo é apresentada com o intuito de entender o pseudo-código de maneira gráfica.

Imagem 1 – Exemplo gráfico do funcionamento da heurística construtiva.




## 3.2 HEURÍSTICA DE MELHORIA

A heurística de melhoria proposta foi pensada como uma mistura a partir do algoritmo de Inserção, na qual se escolhe um item e faz a inserção deste item em todas as posições possíveis, e do algoritmo API, onde se troca os vizinhos de posição. Esta mistura é obtida de forma que possamos continuar respeitando as propriedades já expostas.

A busca local apresentada tem como características a escolha da vizinhança como as tarefas atrasadas que possam ser movidas para o conjunto adiantado e das tarefas adiantadas que possam ser movidas para o conjunto das atrasadas. O movimento a ser feito é o de inserção destas tarefas no outro conjunto. São realizados movimentos até a quantidade limite de metade do número de tarefas.

Seguem os passos:

- Percorre as tarefas atrasadas que tenham tempo de processamento $p < d_0$, sendo $d_0$ a data de início das tarefas (equivalente ao espaço livre entre a data zero e o inicio das atividades). Para cada tarefa testa se colocá-la no grupo das tarefas adiantadas (respeitando a propriedade 1) há melhoria dos custos. Quando encontra a primeira tarefa que melhora os custos interrompe a busca e retorna a nova solução.

- Percorre as tarefas adiantadas e testa para cada tarefa testa se colocá-la no grupo das tarefas atrasadas (respeitando a propriedade 1) há melhoria dos custos. Quando encontra a primeira tarefa que melhora os custos interrompe a busca e retorna a nova solução.
- Se a primeira tarefa não inicia no tempo zero todas as tarefas são deslocadas para permitir que a primeira tarefa inicie em zero. O custo desta nova sequência é comparado com a sequência inicial, e se for menor as tarefas são reordenadas obedecendo a propriedade 1 e essa nova sequência é considerada.
- Repete-se os passos anteriores até que não se tenham vizinhos que melhoram o custo.

Segue pseudocódigo do fluxo proposto acima:

**Algoritmo Heurística de Melhoria**

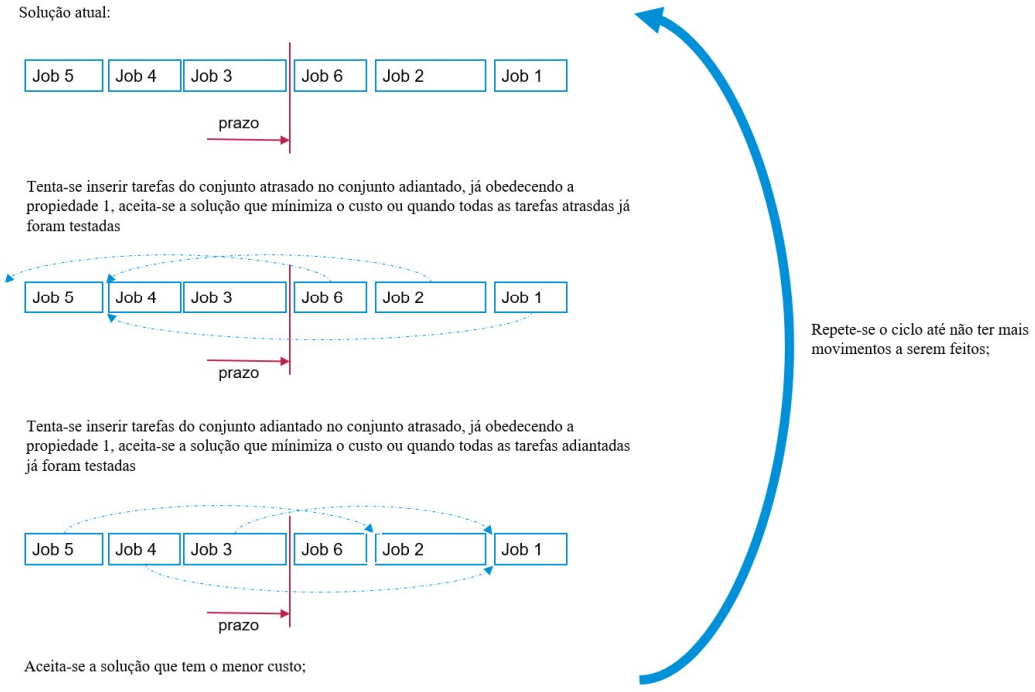
**begin**
S ← *initialize tasks;*
h ← *factor;*
$d = h * \sum_{i=0}^{n} p_i$;
$E \leftarrow i \; \forall \; i \; \in S \mid d_i \leq d$;
$T \leftarrow S - E$;
$cost = f(S)$;
**While** TRUE **do**:
**for** i ∈ T **do:**
$E' \leftarrow E + i$;
sort (E ; $p_i/a_i$ ; decreasing);
$T' \leftarrow T - i$;
$S' \leftarrow E' + T'$;
$cost' = f(S')$;
**if** $cost' < cost$ **then:**
$E \leftarrow E'$
$T \leftarrow T'$
$S \leftarrow T + E$
$cost = cost'$;
Break;
**for** i ∈ E **do:**
$T' \leftarrow T + i$;
sort (T ; $p_i/b_i$ ; increasing);
$E' \leftarrow E - i$;
$S' \leftarrow E' + T'$;
$cost' = f(S')$;
**if** $cost' < cost$ **then:**
$E \leftarrow E'$
$T \leftarrow T'$

$S \leftarrow T + E$
$cost = cost'$;
Break;
**if** cost'=cost **then**
Break;

$d_0 = d - \sum_{i=0}^{n} p_i > d \; \forall \; i \; \in E$;
$S' \leftarrow E + T \; such \; that \; d_i = p_i + p_{i-1} + d_0 \; \forall \; i \; \in S'$;
$S'' \leftarrow E + T \; such \; that \; d_i = p_i + p_{i-1} \; \forall \; i \; \in S''$;
**if** $f(S') < f(S'')$ **then:**
$d_0 = 0$
**end**

Para exemplificar melhor, um exemplo gráfico é apresentado abaixo:

Imagem 2 – Representação gráfica da busca local (heurística de melhoria).




## 3.3 META-HEURÍSTICA – ALGORITMO GENÉTICO

A meta-heurística proposta foi baseada em Algoritmo Genético (AG) e foi escolhida por ser uma meta-heurística popular na área de engenharia, o seu mecanismo de seleção natural é muito interessante para várias aplicações.

De forma resumida AG usa a solução da heurística construtiva para criar uma população de soluções iniciais a partir de mutações, que são trocas aleatórias do status das tarefas de adiantada para atrasada e vice-versa. Com a população inicial, é feita uma avaliação recursiva do custo de cada indivíduo dessa população, as melhores soluções são utilizadas para gerar uma nova população a partir do *crossover* – mistura de duas ou mais soluções com correção de factibilidade – e novamente uma mutação. Este ciclo se repete por um número máximo de iterações ou espera-se a convergência, o que pode ser computacionalmente custoso.

Para a implementação, foi adotada a idéia de genes, onde cada tarefa *i* tem seu gene $g_i$ e cada gene pode ter valor de 0, para status atrasada, ou 1, para status de adiantado.

Imagem 3 – Exemplo da representação da programação via genes.


A implementação do AG possui alguns parâmetros e nuances específicos que serão abordados nos próximos itens.

### *3.3.1 Cross-over*

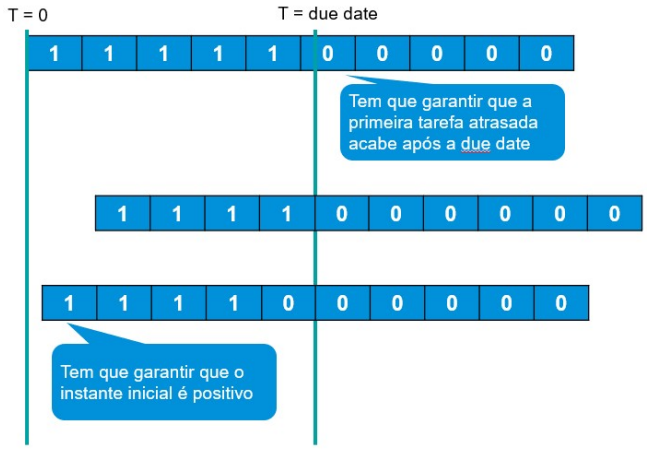
A etapa de geração de nova população (*cross-over)*, onde iniciamos uma nova geração, inicialmente foi feita utilizando as duas soluções da geração anterior, onde a solução resultante seria metade de cada pai. Dessa forma a pior solução da geração anterior é substituido pela nova solução, resultado do *cross-over*.

Após a geração da solução resultante, é necessário uma correção de factibilidade para que não ocorra casos onde há estouro do prazo devido a excesso de tarefas adiantadas. Portanto, para toda tarefa *i* adiantada é calculado o tempo atual do conjunto adiantado *E* e ao atingir o prazo, todas as tarefas posteriores que deveriam ser adiantadas pela solução atual passam a ser atrasadas, modificando a solução resultante. Importante notar que as tarefas podem ser adiantadas ou atrasadas porém a programação sempre se dá respeitando a propriedade 1, portanto a correção de factibilidade se da com preferência para manter as tarefas com a razão *p/a* baixos no conjunto adiantado e as tarefas com *p/a* alto serão movidas para o conjunto atrasado, na ordem de *p/b* crescente, se não houver tempo suficiente.

Para aproveitamento de processo, durante a correção de factibilidade do *cross-over* também é checado o instante inicial que melhor acomoda a solução, tendo três possibilidades, como mostra a figura abaixo:

- Programação iniciando em *t*=0 (pode existir *strandling jobs* – que são tarefas que iniciam antes do prazo e terminam após o prazo, e portanto não entram na regra da propriedade 1);
- Tarefas do conjunto adiandatas terminando exatamente no prazo;
- Primeira tarefa do conjunto de atrasadas terminando exatamente no prazo.

Imagem 4 – Possibilidades de inicio existentes para uma mesma solução que atendem todas as propriedade necessárias para garatia de solução ótima.



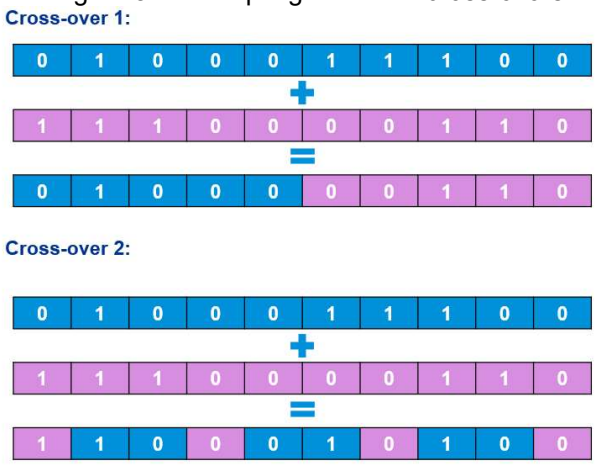

Apesar deste tipo de *cross-over* resultar em soluções de boa qualidade, foi percebido em testes preliminares que havia pouca diversidade no decorrer das gerações, pois este *cross-over* foi feito somente um vez por geração, o que resultava em um processo ágil, porém que gerava pouca diversidade e tendia a necessitar de um número grande de gerações para ter uma convergência em minímo local aceitável. Além disso, a baixa diversidade faz com que seja difícil escapar destes mínimos locais, mesmo com as posteriores mutações.

Visto isso, foi inserido um novo tipo de *cross-over*, onde para cada tarefa, a solução resultante é obtida com base na escolha entre o status da mesma tarefa nas soluções pai, ou seja, com probabilidade uniforme escolhe-se para a tarefa $i$ de qual das soluções pais, ela deve herdar o status (adiantado ou atrasado). Dessa forma é o espaço de busca é diversificado.

Com os testes, comentados a fundo em 3.3.5, foi identificado que mesmo com estas táticas havia uma baixa diversidade e resultavam em uma convergência prematura. Após a 10 iteração não haviam mais grandes melhorias e eram necessárias iterações na ordem de 50.000 à 100.000 para permitir pequenas melhoras posteriores, com isso foi necessário aumentar o *cross-over*, que é na implmementação final ficou sendo feito substituindo 50% da população atual, e sempre escolhendo os pais como os melhores dentro o grupo controlado por um parâmetro $p_p$ (Tamanho da elite de soluções), por exemplo, para um problema de tamanho 100, serão escolhidos com probabilidade uniforme, dois das $100 * p_p$ melhores soluções para passarem pelo processo de *cross-over* e a solução resultante substituirá a pior solução da população atual.

A figura abaixo demonstra com um exemplo o uso das duas técnicas de *cross-over*.

Imagem 5 – Exemplo gráfico dos *cross-overs*.




### 3.3.2 Mutação

Para a etapa de mutação das soluções encontradas, a implementação altera de forma aleatória o status de cada uma das tarefas com probabilidade *p*, ou seja, para cada tarefa *i* de uma solução, o status da tarefa é alterado de acordo com uma probabilidade uniforme de valor *p*, definida posteriormente.

Dessa forma, foi compensado a pouca diversidade criada via *cross-over* com a uma mutação que pode, por chance, gerar uma nova solução totalmente diferente da anterior.

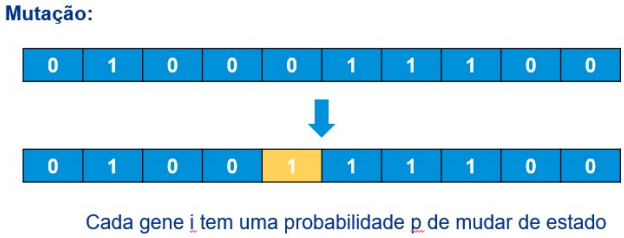
Para impedir que cada geração tenha uma população pouco divergente, devido a um baixo *cross-over* e uma alta taxa de mutação porém executada em poucas soluções, foi feito a contrapartida de escolher com uma probabilidade uniforme a chance de uma solução ser mutada.

Dessa forma, há dois parâmetros de controle de mutação:

- Probabilidade de uma solução vinda do *cross-over* sofrer mutação em qualquer dos seus genes;
- Probabilidade de cada tarefa (ou gene) sofrer uma mutação.

A figura abaixo demonstra a alteração feita, sendo que para cada tarefa abaixo, foi sorteada a possibilidade de se alterar o gene, e apenas a 5° tarefa desta instância teve seu gene alterado.

Imagem 6 – Exemplo gráfico da mutação.




### 3.3.3 Geração da população inicial

Métodos de meta-heurística são altamente dependentes da solução inicial, sendo métodos populacionais mais sensíveis à essa escolha devido a falta de mecanismo de escape de ótimo local, sendo nesse caso, necessário uma taxa de mutação muito alta para escapar dos ótimos locais o que ocasiona uma convergência demorada devido a uma população extremamente diversa.

Devido a estes problemas, não é possível utilizar a solução da heurística de melhoria, pois já se espera ser um ótimo local, para compor a solução inicial. Como só há a solução da heurística construtiva, temos apenas uma solução que comporá a geração inicial. Para popular esta geração inicial, a heurística construtiva passa pelo processo de mutação *n* vezes, até que se tenha a quantidade de soluções estabelecidas pelo parâmetro de tamanho da população.

### 3.3.4 Parâmetros da solução

Parâmetros da solução adodata:

- Tamanho da população:
  Após alguns testes em instâncias pequenas, foi concluido que uma alternativa boa seria manter o tamanho da população variável de acordo com o tamanho da instância. Esta saída se justifica pela facilidade com que instâncias maiores caiam em mínimos locais, dessa forma é possível imaginas que instâncias maiores necessitam de uma maior diversidade da população para escapar destes mínimos locais, diferentemente do que ocorre com instâncias menores.

- Número de Gerações:
  O número de gerações é uma abstração para tempo de execução, onde explicita-se quantos grupos de soluções serão gerados pelas etapas consecutivas de *cross-over*, mutação, e avaliação das soluções. Quanto maior o número de gerações, se espera que seja obtida uma solução melhor, respeitando a natureza de cada problema e implementação para que não haja uma convergência prematura em mínimo local ou uma execução demasiadamente morosa devido a um grande número de execuções.

- Probabilidade de mutação da população inicial:
  Probabilidade de uma solução atual sofrer uma mutação, de acordo com o mecanismo citado acima.

- Tamanho do grupo de elite de escolha dos pais:
  Controla o tamanho do subset de soluções $S^* \in S$, na qual será escolhido as soluções geradores para *cross-over*, quanto maior este valor, maior o espaço de escolha dos pais, não sendo fixo sempre as melhores soluções. A escolha de cada solução geradora (pai), é feita com probabilidade uniforme dentro do subconjunto $S^*$.

- Probabilidade de mutação da geração inicial:
  Controla a probabilidade de cada tarefa $i$ da solução $s$ ter seu gene $g_i$ (status) alterado de adiantado para atrasado, e vice-versa, para gerar a população inicial.

- Probabilidade de mutação da geração atual:
  Controla a probabilidade de cada tarefa $i$ da solução $s$ ter seu gene $g_i$ (status) alterado de adiantado para atrasado, e vice-versa, para gerar uma nova população.

### 3.3.5 Otimização e escolha de parâmetros

Para otimização e escolha de parâmetros, foram feitos testes iniciais com uma implementação inicial, após estes testes foram escolhidos aleatóriamente 1 instância de teste para cada tamanho, com um fator h aleatório dentre as opções. Para estas foi feito o teste rodando várias combinações em um esquema de busca em grade (também chamado de busca fatorial) gerando 1.620 testes. Apartir destes testes em conjunto com o resultado de todas as instâncias para a primeira implementação, foi possível avaliar quais instâncias poderiam ser utilizadas para otimização da implementação (como alterações e maiores modificações) e também para comparação dos valores utilizados nos parâmetros acima.

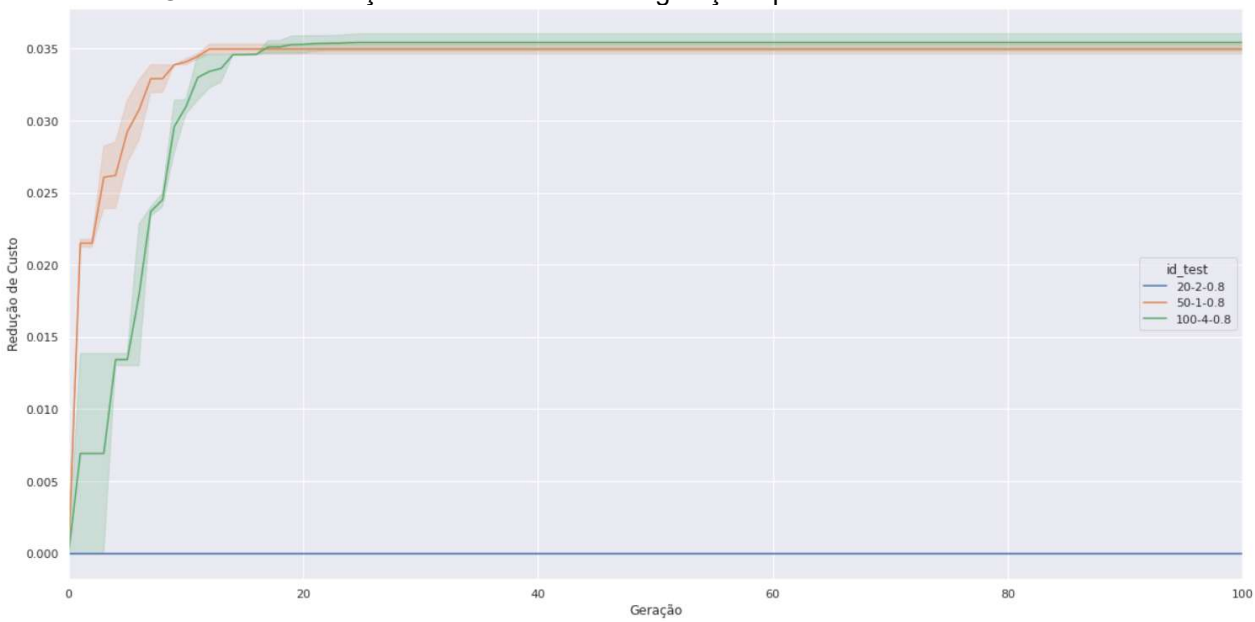
Devido a menor capacidade da solução inicial em melhorar problemas com prazo grande, sua maior variância nos resultados da busca em grade e tempo, foi escolhida as combinações da tabela abaixo, para ajustar a implementação e parâmetros.

Tabela 1 – Instâncias escolhidas para testes de melhoria na implementação e otimização de parâmetros

| Tamanho | ID | Fator h |
|---|---|---|
| **20** | 2 | 0,8 |
| **100** | 4 | 0,8 |
| **50** | 1 | 0,8 |

O Gráfico 1 demonstra a redução de custo atingida pela meta-heurística quando comparada com a heurística construtiva, com o número de gerações da solução:

Gráfico 1 – Evolução do custo durante as gerações para as instâncias de teste.



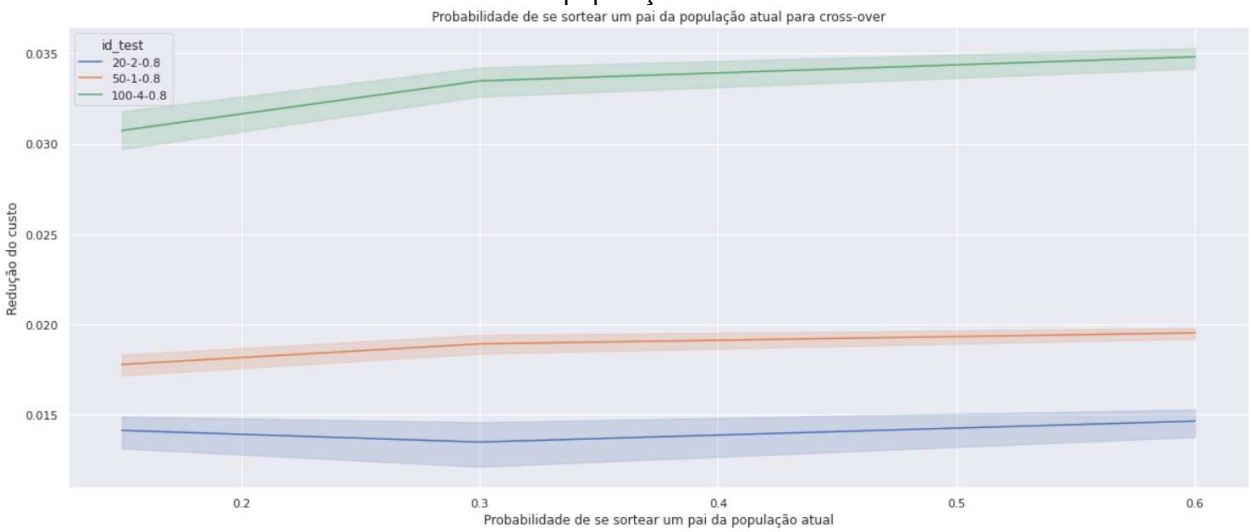

Como é possível de verificar, há uma convergência para um mínimo local por volta das 20 gerações, não tendo grande mudança posterior, o desvio padrão dos testes se demonstram no sombreado da cor de cada curva, o que demonstra baixo impacto de outros parâmetros. Isso é interessante pois inicialmente, se suspeitava que devido a implementação ter baixa diversidade, seria necessário um grande número de gerações para se convergir a um mínimo global o que não se comprovou com o gráfico acima. Porém quando expandimos o gráfico para 10.000 iterações foi verificado que há uma leve melhora na geração 1.000, com isso foi entendido que:

- Não há necessidade de ter um número de iterações extremamente alto visto que o custo computacional será alto e o retorno da solução seria baixo;
- Podemos interromper a meta-heurística antes do término de iterações pois há um momento (nos exemplos acima ~20 iterações) em que há uma convergência, basta escolher um valor alto o suficiente para que garanta que continue tendo melhorias no decorrer do processo e só seja interrompido quando houver uma maior garantia de convergência;

Como este parâmetro se demonstrou o mais importante para a solução final e o conjunto de teste utilizado para otimizar os parâmetros e implementação foi pequeno, foi adotada por segurança para não impactar negativamente nas demais instâncias, a estratégia de aplicar um número grande de interações, em torno de 1.000, e usar a ferramenta de *early stop*, vinda de outras meta-heurísticas e de redes neurais profundas, onde se não houver uma melhora no custo em 100 interações, mata-se o processo de melhoria e aceita-se a solução atual como incumbente. Está solução se mostrou eficiente pois em instâncias menores rápidamente a convergência é atingida e perder mais tempo com novas gerações não seria eficiente, e em instâncias maiores houveram casos em que o número de gerações finais foram entre 300 e 550 o que demonstra que apesar do *early stop* houveram casos de escape de convergências prematuras. Nenhuma instância atingiu as 1.000 iterações.

O outro parâmetro que se mostrou muito eficiente para melhorar o custo final foi a probabilidade de sorteio das soluções pais para *cross-over*. Como mostra-se o gráfico abaixo, para soluções pequenas, 20 e 50 tarefas, o custo final teve pouca variação em relação a mudança do parâmetro, porém sofreu um impacto significativo em relação ao problema de 100 tarefas, o que é interessante pois isso indica uma melhora nos pontos os a implementação se mostrou menos eficiente, que são instâncias grandes.

Gráfico 2 – Evolução do custo em relação a probabilidade de sorteio dos pais para geração de uma nova população.




Os demais parâmetros não tiveram um efeito grande, muito provavelmente devido aos nuances da implementação feita.

Os parâmetros finais foram:

1. Tamanho da população = 1.000, com early stop de 100.
2. Número de Gerações = Max ( 3*[número de tarefas], 200)
3. Probabilidade de um individuo sofre mutação = 0.8
4. Tamanho do subset de escolha dos pais = 0.6
5. Probabilidade de mutação de um gene da geração inicial = 0.3
6. Probabilidade de mutação de um gene = 0.6

### 3.3.6 Pseudo-código

**Algoritmo Genético**

```
begin
    generate initial population;
    j = 1.000;
    early_stop = 100;
    elite_size=0.6;
    prob_enter_mutation=0.8
    set cost_min_atual;
```

**set** size_population_to_mutation;
**While** n < j **or** k<*early_stop* **do:**
n=n+1;
**for** i < size_population_to_mutation **then:**
i=i+1
$S^* \leftarrow sort(S; cost; increasing) * elite\_size$
$pai_1, pai_2 \leftarrow random\ (s \in S^*)$
**If** $n\ \boldsymbol{MOD}\ 2 = 0$ **then:**
$filho$ = **func** crossover_1($pai_1, pai_2$)
**else:**
$filho$ = **func** crossover_2($pai_1, pai_2$)
**if** random(0,1) < *prob_enter_mutation* **then:**
$filho$ = **func** mutation($filho$)
*replace worst current solution from S with filho*
**if** min(cost_population)<cost_min_atual **then**:
cost_min_atual = min(cost_population)
**else**:
k=k+1
**end while**
**end**

## 4 RESULTADOS COMPUTACIONAIS

Biskup & Feldmann (2001) criaram 280 estâncias para o problema de programação de tarefas com data de entrega comum com penalização de atraso e adiantamento. Para problemas com 10, 20, 50, 100, 200, 500 e 1000 tarefas os autores geraram 10 grupos de dados para cada um. Para cada um desses problemas foram geradas 4 estâncias com prazos de entrega definido pelo maior inteiro que é menor que h*(soma de todos os P das tarefas do problema), para os seguintes valores de h 0,2; 0,4; 0,6 e 0,8. Os dados dos problemas gerados pelos autores estão disponibilizados em Beasley, J. E. (2010) assim como resultados para os 280 problemas, sendo alguns melhores dos que encontrados por Biskup & Feldmann (2001).

A seguir apresentamos os valores comparativos entre as heurísticas propostas e os valores obtidos por Biskup & Feldmann (2001) e os disponíveis em Beasley, J. E. (2010).

As tabelas apresentam a soma das diferenças para cada problema definidas por:

$$\%DIF = 100 * (FO^{HC} - FO^{BF})/FO^{BF}$$

Onde $FO^{HC}$ é o valor da função objetivo encontrado com as heurísticas propostas e $FO^{BF}$ é o valor da função objetivo encontrado por Biskup & Feldmann (2001) ou disponível em Beasley, J. E. (2010) dependendo da tabela, importante mencionar que os valores de Beasley, J. E. (2010) são os tidos como ótimos para instâncias com 10 tarefas.

A Tabela 2 apresenta a comparação da heurística construtiva proposta com os valores obtidos por Biskup & Feldmann (2001), a Tabela 3 apresenta a comparação da heurística construtiva com os

valores disponíveis em Beasley, J. E. (2010) e a Tabela 3 apresenta os tempos de processamento para a heurística construtiva.

Tabela 2 – Comparação da heurística construtiva com os valores encontrados por Biskup & Feldmann (2001)

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | 26,33% | 8,47% | 6,40% | 2,72% | 1,79% | 0,64% | 0,72% | **6,72%** |
| **0,4** | 25,91% | 9,16% | 5,33% | 2,20% | 1,07% | 0,66% | 0,40% | **6,39%** |
| **0,6** | 7,69% | 3,54% | 1,43% | 1,40% | 1,84% | 2,76% | 2,21% | **2,98%** |
| **0,8** | 5,79% | 10,44% | 7,53% | 6,96% | 8,19% | 6,60% | 8,07% | **7,65%** |
| **Média** | **16,43%** | **7,90%** | **5,17%** | **3,32%** | **3,22%** | **2,66%** | **2,85%** | **5,94%** |

Tabela 3 – Comparação da heurística construtiva com os valores disponíveis em Beasley, J. E. (2010)

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | 34,82% | 8,47% | 6,40% | 2,72% | 1,79% | 0,64% | 0,72% | **7,94%** |
| **0,4** | 28,94% | 9,16% | 5,33% | 2,20% | 1,07% | 0,66% | 0,40% | **6,82%** |
| **0,6** | 7,71% | 3,54% | 1,43% | 1,40% | 1,84% | 2,76% | 2,21% | **2,98%** |
| **0,8** | 5,94% | 10,44% | 7,53% | 6,96% | 8,19% | 6,60% | 8,07% | **7,68%** |
| **Média** | 19,35% | 7,90% | 5,17% | 3,32% | 3,22% | 2,66% | 2,85% | 6,35% |

Tabela 4 – Tempos de processamento da heurística construtiva

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | 0,0000 | 0,0001 | 0,0001 | 0,0001 | 0,0003 | 0,0005 | 0,0010 | **0,0003** |
| **0,4** | 0,0001 | 0,0001 | 0,0001 | 0,0001 | 0,0002 | 0,0006 | 0,0010 | **0,0003** |
| **0,6** | 0,0000 | 0,0001 | 0,0001 | 0,0001 | 0,0002 | 0,0005 | 0,0009 | **0,0003** |
| **0,8** | 0,0000 | 0,0000 | 0,0001 | 0,0001 | 0,0002 | 0,0005 | 0,0009 | **0,0003** |
| **Média** | 0,0001 | 0,0001 | 0,0001 | 0,0001 | 0,0002 | 0,0005 | 0,0010 | 0,0003 |

As Tabela 5 e Tabela 6 apresentam as mesmas comparações que as Tabela 2 e Tabela 3, respectivamente, porém para a heurística de melhoria, assim como a Tabela 7 apresenta os tempos de processamento.

Tabela 5 – Comparação com os valores encontrados por Biskup & Feldmann (2001) e a heurística de melhoria

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | -1,2% | 0,0% | 1,1% | 0,4% | 0,8% | 0,2% | 0,4% | **0,2%** |
| **0,4** | 6,5% | 2,3% | 1,6% | 0,2% | 0,4% | 0,3% | 0,3% | **1,7%** |
| **0,6** | 3,2% | 0,1% | 0,3% | **-0,1%** | **-0,1%** | **-0,1%** | **-0,1%** | **0,5%** |
| **0,8** | 1,5% | 0,0% | 0,0% | **-0,1%** | **-0,1%** | **-0,1%** | **-0,1%** | **0,2%** |
| **Média** | **2,5%** | **0,6%** | **0,8%** | **0,1%** | **0,2%** | **0,1%** | **0,1%** | **0,6%** |

Tabela 6 – Comparação com os valores disponíveis em Beasley, J. E. (2010) e a heurística de melhoria

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | 5,3% | 0,0% | 1,1% | 0,4% | 0,8% | 0,2% | 0,4% | **1,2%** |
| **0,4** | 8,9% | 2,3% | 1,6% | 0,2% | 0,4% | 0,3% | 0,3% | **2,0%** |
| **0,6** | 3,2% | 0,1% | 0,3% | **-0,1%** | **-0,1%** | **-0,1%** | **-0,1%** | **0,5%** |
| **0,8** | 1,6% | 0,0% | 0,0% | **-0,1%** | **-0,1%** | **-0,1%** | **-0,1%** | **0,2%** |
| **Média** | **4,7%** | **0,6%** | **0,8%** | **0,1%** | **0,2%** | **0,1%** | **0,1%** | **0,9%** |

Tabela 7 – Tempos de processamento da heurística construtiva e da heurística de melhoria

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | 0,0007 | 0,0008 | 0,0023 | 0,0060 | 0,0208 | 0,1151 | 0,5656 | **0,1016** |
| **0,4** | 0,0004 | 0,0009 | 0,0032 | 0,0089 | 0,0366 | 0,1655 | 0,8884 | **0,1577** |
| **0,6** | 0,0005 | 0,0013 | 0,0057 | 0,0441 | 0,3970 | 6,0641 | 37,4574 | **6,2814** |
| **0,8** | 0,0005 | 0,0017 | 0,0120 | 0,0784 | 0,6553 | 10,9611 | 63,8145 | **10,7891** |
| **Média** | 0,0005 | 0,0012 | 0,0058 | 0,0344 | 0,2774 | 4,3265 | 25,6815 | 4,3325 |

A Tabela 8 apresenta a comparação direta entra a heurística de melhoria com a heurística construtiva.

Tabela 8 – Comparação entre heurística de melhoria e heurística construtiva.

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | -27,53% | -8,47% | -5,30% | -2,32% | -0,99% | -0,44% | -0,32% | **-6,52%** |
| **0,4** | -19,41% | -6,86% | -3,73% | -2,00% | -0,67% | -0,36% | -0,10% | **-4,69%** |
| **0,6** | -4,49% | -3,44% | -1,13% | -1,50% | -1,94% | -2,86% | -2,31% | **-2,48%** |
| **0,8** | -4,29% | -10,44% | -7,53% | -7,06% | -8,29% | -6,70% | -8,17% | **-7,45%** |
| **Média** | **-13,93%** | **-7,30%** | **-4,37%** | **-3,22%** | **-3,02%** | **-2,56%** | **-2,75%** | **-5,34%** |

As Tabela 9 e Tabela 10 apresentam as mesmas comparações que as Tabela 2 e Tabela 3, respectivamente, porém para a implementação da meta-heurística AG, assim como a Tabela 7 apresenta os tempos de processamento.

Tabela 9 – Comparação com os valores encontrados por Biskup & Feldmann (2001) e AG

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | -6,2% | -3,8% | -5,5% | -5,3% | -2,9% | -2,6% | -0,6% | -3,8% |
| **0,4** | -2,2% | -1,6% | -4,5% | -4,3% | -2,4% | -2,3% | -1,0% | -2,6% |
| **0,6** | 0,0% | -0,7% | -0,3% | -0,1% | -0,1% | -0,1% | 0,1% | -0,2% |
| **0,8** | -0,1% | -0,4% | -0,2% | 0,1% | 0,2% | 0,0% | 0,6% | 0,0% |
| **Média** | **-2,1%** | **-1,7%** | **-2,6%** | **-2,4%** | **-1,3%** | **-1,2%** | **-0,2%** | **-1,7%** |

Tabela 10 – Comparação com os valores disponíveis em Beasley, J. E. (2010) e AG

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | 0,0% | -3,8% | -5,5% | -5,3% | -2,9% | -2,6% | -0,6% | -3,0% |
| **0,4** | 0,0% | -1,6% | -4,5% | -4,3% | -2,4% | -2,3% | -1,0% | -2,3% |
| **0,6** | 0,0% | -0,7% | -0,3% | -0,1% | -0,1% | -0,1% | 0,1% | -0,2% |
| **0,8** | 0,0% | -0,4% | -0,2% | 0,1% | 0,2% | 0,0% | 0,6% | 0,0% |
| **Média** | **0,0%** | **-1,7%** | **-2,6%** | **-2,4%** | **-1,3%** | **-1,2%** | **-0,2%** | **-1,3%** |

Conforme os dados mostrados na Tabela 10, é válido salientar o atingimento dos mínimos globais para as instâncias de 10 tarefas, mostrando que a implementação é funcional. A dificuldade continua sendo em instâncias maiores combinadas com fatores h altos, que dão uma maior flexibilidade na escolha da solução.

Tabela 11 – Tempos de processamento da AG

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | 6,51 | 8,63 | 15,36 | 30,37 | 58,22 | 128,90 | 274,74 | **74,68** |
| **0,4** | 6,43 | 8,56 | 15,34 | 31,46 | 58,67 | 128,79 | 280,99 | **75,75** |
| **0,6** | 6,66 | 9,08 | 16,48 | 33,11 | 64,24 | 156,49 | 298,69 | **83,53** |
| **0,8** | 6,73 | 9,33 | 17,49 | 33,26 | 64,43 | 159,36 | 295,54 | **83,73** |
| **Média** | **6,58** | **8,90** | **16,17** | **32,05** | **61,39** | **143,39** | **287,49** | **79,42** |

A Tabela 12 apresenta a comparação direta entra a solução encontrada pela implementação da meta-heurística AG com a heurística construtiva.

Tabela 12 – Comparação entre AG e heurística construtiva.

| h | 10 | 20 | 50 | 100 | 200 | 500 | 1000 | Média |
|---|---|---|---|---|---|---|---|---|
| **0,2** | -24,5% | -10,8% | -9,3% | -5,5% | -3,8% | -3,3% | -2,9% | **-8,6%** |
| **0,4** | -18,9% | -9,4% | -8,3% | -6,2% | -4,0% | -2,8% | -2,7% | **-7,5%** |
| **0,6** | -6,2% | -3,9% | -1,6% | -1,5% | -1,9% | -2,8% | -2,2% | **-2,9%** |
| **0,8** | -5,3% | -9,2% | -7,0% | -6,4% | -7,6% | -6,3% | -7,5% | **-7,0%** |
| **Média** | -13,7% | -8,3% | -6,6% | -4,9% | -4,3% | -3,8% | -3,8% | -6,5% |

Ambas as heurísticas foram implementadas em usando python 3.7, rodando em WSL (Windows Subsistem for Linux v. 1) com imagem Ubuntu 7.5.0-3ubuntu1~18.04, o computador utilizado possui Windows 10, 16GB de RAM, Processador AMD Ryzen 5 PRO 4650U with Radeon Graphics  2.10 GHz.

## 5 CONCLUSÃO

Como apresentado na Tabela 2 e na Tabela 3 os valores da heurística construtiva são em média 5,9% e 6,3% piores que os obtidos por Biskup & Feldmann (2001) e disponíveis em Beasley, J. E. (2010), respectivamente. Com a implementação da busca local, os resultados melhoraram e partiram para 0,6% e 0,9%, respectivamente.

Com a implementação da meta-heurística AG, foi possível melhorar significativamente os resultados obtidos pela heurística construtiva, superando os resultados obtidos pela heurística de melhoria. Como *trade-off* temos o tempo de ajuste do algoritmo, pois devidos os parâmetros e diferentes opções de implementação, há um custo de desenvolvimento que deve ser levado em consideração em implementações em ambientes produtivos. Porém mesmo com essa contraparte, é entendido que na maioria dos casos vale a pena a implementação da solução de meta-heurística proposto pois há uma redução significativa do custo.

Como popostas de melhoria para a implementação ficam:

- Implementação de mutações para o *strandling job* (tarefa que inicia antes do prazo e termina após o prazo), pois a solução é fixa após a mutação e apenas o inicio da programação que é testado algumas opções (sem alterarmos a solução), conforme mostrado na Imagem 4;
- Taxa de mutação dinâmica em conjunto com *early-stop*, de forma que se a solução imbumbente não for alterada em *n* iterações, aumeta-se a taxa de mutação para dar mais diversidade e escapar de mínimos locais;
- Otimização do algoritmo com linguagens de baixo nível e paralelismo, pois a implementação na linguagem escolhida impactou no tempo de processamento. Apesar do tempo não ter ficado alto, foi um fator limitante para que fossem testados mais opções.

## 6 REFERÊNCIAS

Beasley, J. E. (2010). *OR-LIBRARY*. http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Biskup, D., & Feldmann, M. (2001). Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers and Operations Research*, *28*(8). https://doi.org/10.1016/S0305-0548(00)00008-3

Jarboui, B., Siarry, P., & Teghem, J. (2013). Metaheuristics for Production Scheduling. In *Metaheuristics for Production Scheduling*. https://doi.org/10.1002/9781118731598