INSTITUTO TECNOLÓGICO VALE

Programa de Pós Graduação em Instrumentação, Controle e Automação de Processos de Mineração - PROFICAM
Universidade Federal de Ouro Preto - Escola de Minas
Associação Instituto Tecnológico Vale - ITV

Trabalho Final

PROGRAMAÇÃO APLICADA À MINERAÇÃO

Edmar Magela do Carmo Brito
Jéssica Cristina Teixeira da Costa
João Paulo Estevão Morais
Rafael Pires Mota

Ouro Preto
Abril de 2024

Edmar Magela do Carmo Brito
Jéssica Cristina Teixeira da Costa
João Paulo Estevão Morais
Rafael Pires Mota

PROGRAMAÇÃO APLICADA À MINERAÇÃO

Trabalho Final apresentada ao curso de Mestrado Profissional em Instrumentação, Controle e Automação de Processos de Mineração da Universidade Federal de Ouro Preto e do Instituto Tecnológico Vale, como parte dos requisitos para obtenção do título de Mestre em Engenharia de Controle e Automação.

Disciplina: Programação Aplicada à Mineração

Professor: D.Sc. Marcone Jamilson Freitas Souza

Ouro Preto, MG – Brasil
Abril de 2024

Sumário

1 Bin Packing Problem **1**
1.1 Heurística Construtiva Gulosa 1
1.2 Implementação via Método Matemático 4
1.3 Implementação via Heurística Gulosa 6
2 Sequenciamento de frentes de lavra **9**
2.1 Adição das penalidades 9
2.2 Função Objetivo 9
2.3 Exportação para o Excel 10
2.4 Gráfico de Gantt 10
3 Alteração do código Simulated Annealing (SA) **12**

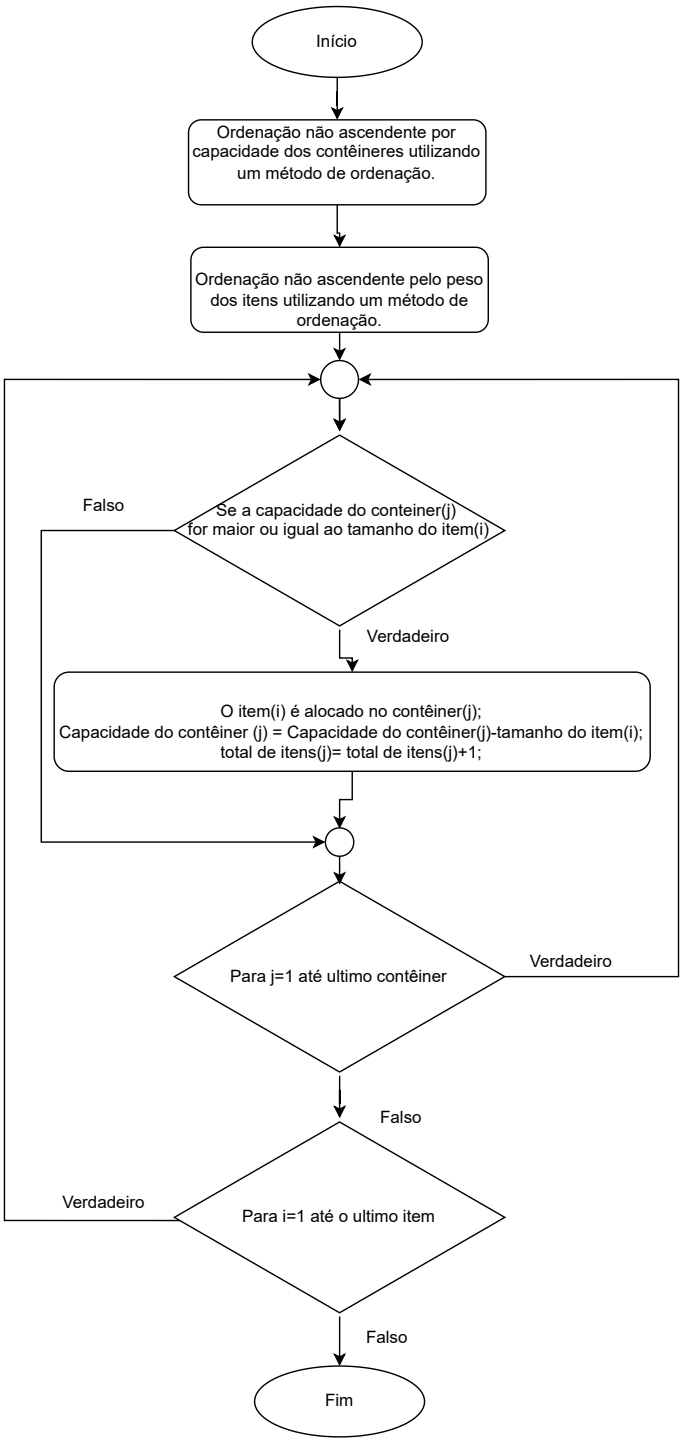
Referências Bibliográficas **13**
1 Implementação Bin Packing Problem **14**
2 Implementação heurística gulosa **15**
3 Implementação sequenciamento de frentes de lavra **16**

1 Bin Packing Problem

1.1 Heurística Construtiva Gulosa

Fluxograma

Figura 1: Fluxograma da Heurística Construtiva Gulosa



Fonte: Elaborado pelos Autores (2024)

Exemplo prático

Para a implementação do problema, foram definidos de forma arbitrária os pesos dos itens e a capacidade de cada contêiner, mantendo o número de itens e de contêineres conforme especificado no enunciado. Para tal, foram utilizados os itens conforme especificado na Tabela (1) e os contêineres conforme apresentado na Tabela (2).

Tabela 1: Relação de itens e seus respectivos pesos

Itens	Peso (kg)
1	6
2	12
3	17
4	8
5	5
6	20
7	8
8	15
9	5
10	18
11	7
12	10
13	15
14	19
15	6

Fonte: Elaborado pelos Autores (2024)

Tabela 2: Relação de contêineres e suas respectivas capacidades

Contêineres	Cap. (kg)
1	90
2	100
3	150
4	100
5	70

Fonte: Elaborado pelos Autores (2024)

A primeira parte da resolução do problema utilizando uma heurística construtiva gulosa é fazer a ordenação dos itens e contêineres em ordem não ascendente. E em caso de empate, escolhe-se o item de menor "ID". O resultado dessa ordenação pode ser vista na Tabela (3) e Tabela (4).

Como não há variação no valor de cada contêiner em relação ao seu tamanho, a heurística gulosa foi desenvolvida para selecionar os contêineres com base na ordem não

Tabela 3: Ordenação dos itens em ordem não ascendente de peso

Itens	Peso (kg)
6	20
14	19
10	18
3	17
8	15
13	15
2	12
12	10
4	8
7	8
11	7
1	6
15	6
5	5
9	5

Fonte: Elaborado pelos Autores (2024)

Tabela 4: Ordenação dos contêineres em ordem não ascendente de capacidades

Contêineres	Cap. (kg)
3	150
2	100
4	100
1	90
5	70

Fonte: Elaborado pelos Autores (2024)

ascendente de capacidade. Dessa forma, o contêiner com maior capacidade, que neste caso é de 150 kg, é escolhido primeiro.

O processo de alocação é iniciado com o item de maior peso sendo alocado ao contêiner 3. Após esse passo, os itens são distribuídos conforme a ordem estabelecida na Tabela (3), até que exista um item que exceda a capacidade restante do contêiner em uso, sendo alocado à um próximo contêiner disponível, de acordo com a sequência descrita na Tabela (4). Ou seja, a alocação é realizada atribuindo o item de maior peso ainda não designado ao contêiner de maior capacidade que possua espaço suficiente.

A cada iteração, verifica-se se o item pode ser acomodado em algum dos contêineres já em uso. Caso nenhum deles tenha capacidade disponível, um novo contêiner é aberto. Esse processo continua até que todos os itens sejam alocados adequadamente.

A distribuição dos itens foi feita da seguinte forma:

- **Item 6:** Alocado ao Contêiner 3 (Espaço restante C3: 130 e C2: 100)

- **Item 14:** Alocado ao Contêiner 3 (Espaço restante C3: 111 e C2: 100)
- **Item 10:** Alocado ao Contêiner 3 (Espaço restante C3: 93 e C2: 100)
- **Item 3:** Alocado ao Contêiner 3 (Espaço restante C3: 76 e C2: 100)
- **Item 8:** Alocado ao Contêiner 3 (Espaço restante C3: 61 e C2: 100)
- **Item 13:** Alocado ao Contêiner 3 (Espaço restante C3: 46 e C2: 100)
- **Item 2:** Alocado ao Contêiner 3 (Espaço restante C3: 34 e C2: 100)
- **Item 12:** Alocado ao Contêiner 3 (Espaço restante C3: 24 e C2: 100)
- **Item 4:** Alocado ao Contêiner 3 (Espaço restante C3: 16 e C2: 100)
- **Item 7:** Alocado ao Contêiner 3 (Espaço restante C3: 8 e C2: 100)
- **Item 11:** Alocado ao Contêiner 3 (Espaço restante C3: 1 e C2: 100)
- **Item 1:** Não cabe no Contêiner 3, foi alocado ao Contêiner 2 (Espaço restante C3: 1 e C2: 94)
- **Item 15:** Não cabe no Contêiner 3, foi alocado ao Contêiner 2 (Espaço restante C3: 1 e C2: 88)
- **Item 5:** Não cabe no Contêiner 3, foi alocado ao Contêiner 2 (Espaço restante C3: 1 e C2: 83)
- **Item 9:** Não cabe no Contêiner 3, foi alocado ao Contêiner 2 (Espaço restante C3: 1 e C2: 78)

Assim, de acordo com a heurística gulosa criada, foram utilizados dois contêineres, sendo o número 3 e 2, com capacidades 150 kg e 100 kg, respectivamente. A resposta final do problema foi: s = { 2 , 3 , 3 , 3 , 2 , 3 , 3 , 3 , 2 , 3 , 3 , 3 , 3 , 3 , 2 }, sendo a unidade armazenada nesse vetor o contêiner em que o item dessa posição é alocado. Além disso, o contêiner 3 ficou com espaço residual de 1 kg e o contêiner 2 com 78 kg.

1.2 Implementação via Método Matemático

Para a resolução do problema via implementação com método matemático, é necessário definir o que cada restrição e variável de decisão estabelece.

A Equação (1) define a função objetivo, cujo propósito é minimizar o número total de contêineres utilizados.

$$[fo]\min \sum_{i \in \text{Contêineres}} y_i \tag{1}$$

A Equação (2) assegura que cada item j seja alocado a um único contêiner i, impedindo que o mesmo item seja destinado à mais de um contêiner.

$$\sum_{i \in \text{Contêineres}} x_{ij} = 1 \qquad \forall j \in \text{Itens} \tag{2}$$

A Equação (3) garante que o peso total dos itens alocados a um contêiner i não ultrapasse a capacidade desse contêiner.

$$\sum_{j \in \text{Itens}} w_j x_{ij} \leq cap_i y_i \qquad \forall i \in \text{Contêineres} \tag{3}$$

Por fim, as Equações (4) e (5) definem o domínio das variáveis de decisão, que são binárias, assumindo valores 0 ou 1. A variável y_i assume valor 1 quando o contêiner i é utilizado e 0 caso contrário. De forma semelhante, a variável x_{ij} assume valor 1 se o item j for alocado ao contêiner i e 0 em caso contrário.

$$y_i \in \{0, 1\} \qquad \forall i \in \text{Contêineres} \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in \text{Contêineres}, \forall j \in \text{Itens} \tag{5}$$

Por meio da formulação do método matemático, foi possível obter a solução s = { 2 , 3 , 3 , 2 , 3 , 3 , 3 , 3 , 3 , 3 , 2 , 3 , 3 , 3 , 3 }, onde foram utilizados os contêineres 2 e 3, que possuem capacidade 150 e 100 kg. Como não foi especificado na modelagem matemática a diferença de preço entre um contêiner que possui capacidade diferente, o modelo escolheu os que possuíam maiores capacidades, caso fosse especificado e colocado na função objetivo, seriam escolhidos os contêineres de menores capacidades que acomodariam os itens. Assim, a modelagem matemática encontrou o valor de solução utilizando 2 contêineres, sendo este o valor ótimo global, que coincidiu com o valor da heurística gulosa.

Para analisar o código de outra forma, a matriz $X = (x_{ij})_{5\times15}$ é visualizada na Tabela (5).

Tabela 5: Matriz X

X	$J1$	$J2$	$J3$	$J4$	$J5$	$J6$	$J7$	$J8$	$J9$	$J10$	$J11$	$J12$	$J13$	$J14$	$J15$
$I1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$I2$	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0
$I3$	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1
$I4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$I5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Elaborado pelos Autores (2024)

Lembrando que x_{ij} assume valor 1 se o item j for alocado ao contêiner i e 0 em

caso contrário, neste caso, é visível quais itens foram alocados à cada um dos contêineres utilizados. Ainda, $y = (y_i)_5$ é visualizado na Tabela (6).

Tabela 6: Vetor Y

	Y
$I1$	0
$I2$	1
$I3$	1
$I4$	0
$I5$	0

Elaborado pelos Autores (2024)

Lembrando que a variável y_i assume valor 1 quando o contêiner i é utilizado e 0 caso contrário, neste caso, podemos observar que, neste caso, apenas os contêineres 2 e 3 foram utilizados.

A implementação completa do problema Bin Packing é apresentada no Apêndice (1).

1.3 Implementação via Heurística Gulosa

A implementação da heurística gulosa foi feita conforme raciocínio apresentado na Seção (1.1). Para a execução desse raciocínio, foi elaborado um código em Python que estrutura a alocação de itens em contêineres conforme as suas respectivas capacidades.

A primeira etapa do algoritmo é fazer a definição dos itens a serem alocados. Cada item é representado por um dicionário que contém um identificador único e seu peso correspondente. A lista de itens é organizada conforme segue:

```
itens = [
    {'id': 1, 'peso': 6},
    {'id': 2, 'peso': 12},
    {'id': 3, 'peso': 17},
    {'id': 4, 'peso': 8},
    {'id': 5, 'peso': 5},
    {'id': 6, 'peso': 20},
    {'id': 7, 'peso': 8},
    {'id': 8, 'peso': 15},
    {'id': 9, 'peso': 5},
    {'id': 10, 'peso': 18},
    {'id': 11, 'peso': 7},
    {'id': 12, 'peso': 10},
    {'id': 13, 'peso': 15},
    {'id': 14, 'peso': 19},
    {'id': 15, 'peso': 6}
]
```

Neste contexto, esses itens representam os objetos que serão alocados nos contêineres.

De forma semelhante, é necessário definir os contêineres que receberão os itens. Cada contêiner também é representado por um dicionário que inclui seu identificador e a capacidade máxima que pode armazenar. A estruturação dos contêineres é apresentada a seguir:

```
conteineres = [
    {'id': 1, 'capacidade': 90},
    {'id': 2, 'capacidade': 100},
    {'id': 3, 'capacidade': 150},
    {'id': 4, 'capacidade': 100},
    {'id': 5, 'capacidade': 70}
]
```

Neste caso, esse dicionário define a capacidade de cada contêiner, ou seja, é definido o peso que cada contêiner pode armazenar.

Para garantir uma alocação eficiente, os itens e contêineres são ordenados em ordem não ascendente, com base em seus pesos e capacidades, respectivamente. A ordenação é realizada através do seguinte código:

```
itens.sort(key=lambda x: x['peso'], reverse=True)
conteineres.sort(key=lambda x: x['capacidade'], reverse=True)
```

Foi criada uma lista vazia denominada "alocacao"para armazenar qual contêiner cada item foi alocado. Iniciamos uma lista vazia chamada alocação. Essa lista será preenchida com as alocações conforme os itens forem distribuídos nos contêineres.

```
alocacao = []
```

A alocação dos itens nos contêineres é realizada através de um loop que verifica se cada item pode ser alocado em um contêiner disponível. O código implementa essa lógica da seguinte maneira:

```
for item in itens:
    for conteiner in conteineres:
        if conteiner['capacidade'] >= item['peso']:  # Verifica se o
    item cabe no contêiner
            conteiner['capacidade'] -= item['peso']  # Atualiza a
    capacidade do contêiner
            alocacao.append({'item': item['id'], 'conteiner': conteiner[
    'id']})  # Adiciona a alocação
            break  # Vai para o próximo item
```

Por fim, a solução final da alocação é exibida ao usuário, utilizando o código a seguir:

```
print("Solução Final (Item -> Contêiner):")
for aloc in alocacao:
  print(f"Item {aloc['item']} alocado ao Contêiner {aloc['conteiner']}")
```

Deste modo, o código gerou a solução final s = { 2 , 3 , 3 , 3 , 2 , 3 , 3 , 3 , 2 , 3 , 3 , 3 , 3 , 3 , 2 }, também utilizando os contêineres 2 e 3. Assim, a solução deste problema foi feita utilizando dois contêineres, coincidindo com o valor da heurística gulosa implementada manualmente no Excel e também com o método matemático modelado via Lingo.

A implementação completa do problema Bin Packing utilizando a heurística gulosa é apresentada no Apêndice (2).

2 Sequenciamento de frentes de lavra

2.1 Adição das penalidades

Para garantir que possa existir um atraso t_j ou uma antecipação e_j em relação à data de entrega, foi verificado se a restrição $C_j = d_j \ \forall j \in$ Frentes já tinha sido relaxada, conforme apresentado no Lingo:

```
@for(frentes(j):
  C(j) - t(j) + e(j) = d(j));
```

Essa restrição pode ser escrita pela Equação (6).

$$c_j - t_j + e_j = d_j \qquad \forall j \in \text{Frentes} \tag{6}$$

Esta restrição já estava implementada no código. Deste modo, com essa restrição é necessário adicionar as penalidades por antecipação e por atraso na função objetivo. Inicialmente é necessário definir o tamanho das variáveis we e wt no ambiente *"sets"*, conforme segue:

```
sets:
  carregadeiras / @ole('SFLDS-JIT.xlsx','carregadeiras')/:;
  frentes / @ole('SFLDS-JIT.xlsx','frentes')/: C, d, e, t, x, we, wt;
  matriz_fcf(carregadeiras,frentes,frentes): y;
  matriz_ff(frentes,frentes): s;
  matriz_cf(carregadeiras,frentes): p, z;
endsets
```

Além disso, foi necessário fazer as leitura das penalidade da planilha do Excel no ambiente *"data"*, conforme segue:

```
data:
  p = @ole('SFLDS-JIT.xlsx','p'); ! Tempo de processamento;
  s = @ole('SFLDS-JIT.xlsx','preparacao'); ! Tempo de setup;
  M = 100000; ! Número arbitrariamente grande;
  d = @ole('SFLDS-JIT.xlsx','d'); ! Due date;
  we = @ole('SFLDS-JIT.xlsx','we'); ! Penalidade por antecipação;
  wt = @ole('SFLDS-JIT.xlsx','wt'); ! Penalidade por atraso;
enddata
```

2.2 Função Objetivo

A função objetivo para essa questão é apresentada na Equação (7).

$$[fo]\min \sum_{j \in \text{Frentes}} we_j e_j + wt_j t_j \tag{7}$$

A implementação dessa função objetivo no Lingo foi:

```
[JIT] min  = @sum(Frentes(j): wt(j)*t(j) + we(j)*e(j));
```

2.3 Exportação para o Excel

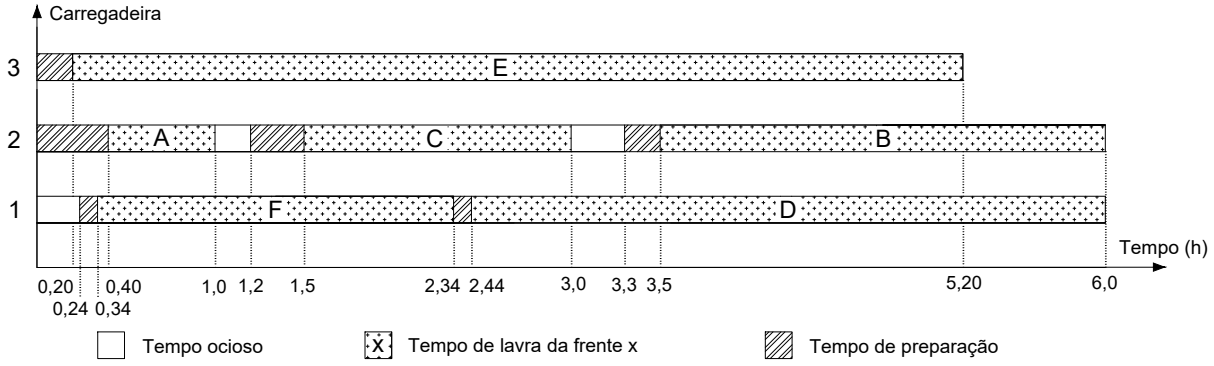
Para exportar os valores da soma ponderada das antecipações e atrasos, e ainda, as antecipações e atrasos, foi feita a seguinte alteração em *"data"*:

```
data:
@ole('SFLDS-JIT.xlsx','conclusao','JIT','z','t', 'e') = C, JIT, z, t, e;
enddata
```

2.4 Gráfico de Gantt

O Gráfico de Gantt para o problema de sequenciamento de frentes de lavra é apresentado na Figura (2).

Figura 2: Gráfico de Gantt para o problema de sequenciamento de frentes de lavra



Fonte: Elaborado pelos Autores (2024)

No diagrama acima, o eixo vertical representa as carregadeiras (Carregadeira 1, Carregadeira 2 e Carregadeira 3). Já o eixo horizontal representa o tempo, medido em horas.

De acordo com o diagrama, podem ser feitas as seguintes observações sobre cada uma das frentes:

(A) A conclusão estava prevista para 1h e foi finalizada exatamente em 1h, sem atraso ou antecipação.

(B) A conclusão estava prevista para 6h e foi finalizada exatamente em 6h, sem atraso ou antecipação.

(C) A conclusão estava prevista para 3h e foi finalizada exatamente em 3h, sem atraso ou antecipação.

(D) A conclusão estava prevista para 6h e foi finalizada exatamente em 6h, sem atraso ou antecipação.

(E) A conclusão estava prevista para 5h e foi finalizada em 5,20h, resultando em um atraso de 0,20h.

(F) A conclusão estava prevista para 2,5h e foi finalizada em 2,34h, resultando em uma antecipação de 0,16h.

Dessa forma, o total de atraso foi de 0,20h e a antecipação foi de 0,16h. Portanto, a função objetivo do sistema just in time é dada por:

$$JIT = 3 * 0,20 + 1 * 0,16 = 0,76 \tag{8}$$

A implementação completa do problema de sequenciamento de frentes de lavra é apresentada no Apêndice (3).

3 Alteração do código Simulated Annealing (SA)

Conforme solicitado no enunciado, os parâmetros do algoritmo foram alterados, conforme segue:

```
fo = SimulatedAnnealing(n,s,d,0.998,10*n,temp_inicial,0.001);
```

Onde, $\alpha = 0.998$ representa a taxa de resfriamento, $SAmax = 10n$ o número máximo de iterações em uma dada temperatura e ($temp\_final = 0.001$ a temperatura final. Além disso, *temp_inicial* representa a temperatura inicial determinada por simulação.

Na implementação do método *Simulated Annealing (SA)* foi inserido:

```
f_star = descida_primeiro_melhora(n, s_star, d);
```

Dete modo, o código original e o código ajustado foram executados 10 vezes utilizando a instância C50.txt do Problema do Caixeiro Viajante. Em cada uma dessas execuções do algoritmo foi coletado o valor da função objetivo e o tempo computacional, conforme Tabela (7).

Tabela 7: Relação entre a função objetivo e tempo computacional do Simulated Annealing (SA) original e refinado com *First Improvement*

Exec	**Fo do SA original (km)**	**Tempo (s)**	**Fo do SA adaptado (km)**	**Tempo (s)**
1	467.17	0.25	462.05	0.39
2	460.00	0.25	443.31	0.35
3	469.25	0.25	463.47	0.37
4	456.75	0.25	470.43	0.37
5	454.94	0.25	448.99	0.38
6	451.46	0.25	448.21	0.36
7	445.18	0.25	448.03	0.39
8	450.24	0.25	455.55	0.37
9	446.52	0.25	453.49	0.37
10	461.51	0.25	452.28	0.35
Média	456.30	0.25	454.48	0.37

Elaborado pelos Autores (2024)

Notou-se que, ao executar o código do Simulated Annealing modificado, quando comparado com o original, a distância percorrida foi reduzida em, aproximadamente, 2 km. Em contrapartida, o tempo de execução médio aumentou em 0.12 segundos. Para um problema de minimização, como é a casa da distância percorrida pelo caixeiro viajante, a melhora foi observada. Trocando-se os parâmetros solicitados, e considerando que não é caro computacionalmente, foi benéfico realizar as adaptações.

Pensando em um número maior de execuções (centenas ou até milhares), a tendência é que a diferença entre as distâncias percorridas seja maior. Isso porque, a busca pelo espaço de soluções irá aumentar e pode-se ter um panorama do quão eficaz foi o resultado do SA alterado. Assim, o valor final da função objetivo (Fo) para o presente problema, seria ainda mais satisfatória.

Referências Bibliográficas

ARENALES, M., ARMENTANO, V. A., MORABITO, R., et al.. *Pesquisa operacional*. Campus/elsevier, 2007.

SOUZA, M. J. F. "Inteligência computacional para otimização", *Departamento de Computaçao, Universidade Federal de Ouro Preto*, 2024. Disponível em: `http://www.decom.ufop.br/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.pdf`. Acesso em: 28 de setembro de 2024.

SOUZA, M. J. F. "Otimização combinatória", *Departamento de Computaçao, Universidade Federal de Ouro Preto*, 2009. Disponível em: `http://www.decom.ufop.br/marcone/Disciplinas/OtimizacaoCombinatoria/OtimizacaoCombinatoria.pdf`. Acesso em: 28 de setembro de 2024.

SOUZA, M., MARTINS, A., COSTA, T., et al.. "Manual do LINGO: com exercícios resolvidos de Programação Matemática", *Departamento de Computaçao, Universidade Federal de Ouro Preto*, 2012. Disponível em: `http://www.decom.ufop.br/prof/marcone/Disciplinas/OtimizacaoCombinatoria/LINGO_ListaExercicios`. Acesso em: 28 de setembro de 2024.

1 Implementação Bin Packing Problem

```
sets:
  Itens/@ole('bin-packing.xlsx','Itens')/: w;
  Conteineres/@ole('bin-packing.xlsx','Conteineres')/: cap, y;
  Matriz(Conteineres, Itens): x;
endsets

data:
  w = @ole('bin-packing.xlsx','w');
  cap = @ole('bin-packing.xlsx','cap');
enddata

! Minimizar número de contêineres utilizados;
[fo] min = @sum(Conteineres(i): y(i));

! Cada item j só pode ser alocado a um único conteiner i;
@for(Itens(j):
   @sum(Conteineres(i): x(i,j)) = 1);

! O total de itens j em um contêiner i não pode ultrapassar a capacidade
    desse mesmo contêiner;
@for(Conteineres(i):
  @sum(Itens(j): w(j)*x(i,j)) <= cap(i)*y(i));

! As variáveis y e x são binárias;
@for(Conteineres(i):
   @bin(y(i));
   @for(Itens(j): @bin(x(i,j))));

data:
  @ole('bin-packing.xlsx','y','x','fo') = y, x, fo;
enddata
```

2 Implementação heurística gulosa

```
itens.sort(key=lambda x: x['peso'], reverse=True)
conteineres.sort(key=lambda x: x['capacidade'], reverse=True]
itens = [
    {'id': 1, 'peso': 6},
    {'id': 2, 'peso': 12},
    {'id': 3, 'peso': 17},
    {'id': 4, 'peso': 8},
    {'id': 5, 'peso': 5},
    {'id': 6, 'peso': 20},
    {'id': 7, 'peso': 8},
    {'id': 8, 'peso': 15},
    {'id': 9, 'peso': 5},
    {'id': 10, 'peso': 18},
    {'id': 11, 'peso': 7},
    {'id': 12, 'peso': 10},
    {'id': 13, 'peso': 15},
    {'id': 14, 'peso': 19},
    {'id': 15, 'peso': 6}]

conteineres = [
    {'id': 1, 'capacidade': 90},
    {'id': 2, 'capacidade': 100},
    {'id': 3, 'capacidade': 150},
    {'id': 4, 'capacidade': 100},
    {'id': 5, 'capacidade': 70}]

alocacao = []

itens.sort(key=lambda x: x['peso'], reverse=True)
conteineres.sort(key=lambda x: x['capacidade'], reverse=True)

for item in itens:
    for conteiner in conteineres:
        if conteiner['capacidade'] >= item['peso']:  # Verifica se o
   item cabe no contêiner
            conteiner['capacidade'] -= item['peso']  # Atualiza a
   capacidade do contêiner
            alocacao.append({'item': item['id'], 'conteiner': conteiner[
   'id']})  # Adiciona a alocação
            break  # Vai para o próximo item

print("Solução Final (Item -> Contêiner):")
for aloc in alocacao:
    print(f"Item {aloc['item']} alocado ao Contêiner {aloc['conteiner']}
   ")
```

3 Implementação sequenciamento de frentes de lavra

```
! Formulação de Zhu e Heady;
sets:
  carregadeiras / @ole('SFLDS-JIT.xlsx','carregadeiras')/:;
  frentes / @ole('SFLDS-JIT.xlsx','frentes')/: C, d, e, t, x,we,wt;
  matriz_fcf(carregadeiras,frentes,frentes): y;
  matriz_ff(frentes,frentes): s;
  matriz_cf(carregadeiras,frentes): p, z;
endsets

data:
  p = @ole('SFLDS-JIT.xlsx','p'); ! Tempo de processamento;
  s = @ole('SFLDS-JIT.xlsx','preparacao'); ! Tempo de setup;
  M = 100000; ! Número arbitrariamente grande;
  d = @ole('SFLDS-JIT.xlsx','d'); ! Due date;
  we = @ole('SFLDS-JIT.xlsx','we'); !Penalidade por antecipação;
  wt = @ole('SFLDS-JIT.xlsx','wt'); ! Penalidade por atraso;
enddata

! Minimizar a soma ponderada do tempo de antecipação e atraso.
Aqui estão faltando ler os pesos de antecipação e atraso e usá-los
na função objetivo;

[JIT] min  = @sum(Frentes(j): wt(j)*t(j) + we(j)*e(j));


! Em cada frente j pode-se não atender à data devida d_j;
@for(frentes(j):
  C(j) - t(j) + e(j) = d(j));


! Em cada frente só pode ser alocada uma única carregadeira;
@for(frentes(j) | j #NE# @index(0):
  @sum(carregadeiras(k): z(k,j)) = 1);

! Para cada frente j, exceto a origem, há uma única frente
  imediatamente predecessora em cada carregadeira;
@for(frentes(j) | j #NE# @index(0):
  @for(carregadeiras(k):
    @sum(frentes(i) | i #NE# j: y(k,i,j)) = z(k,j)));

! De cada frente i, há uma única frente j imediatamente
  sucessora em cada carregadeira;
@for(frentes(i):
  @for(carregadeiras(k):
     @sum(frentes(j) | j #NE# i:
```

```
        y(k,i,j)) <= z(k,i)));

! Uma frente j só é concluída depois de terminada a lavra
  da frente i imediatamente anterior, mais o processamento da frente j
  na carregadeira k, mais o tempo de preparação da carregadeira para
   sair da
  frente i e ir para a frente j;
@for(frentes(j) | j #NE# @index(0):
  @for(frentes(i) | i #NE# j:
    @for(carregadeiras(k):
      C(j) - C(i) - M*y(k,i,j) >= p(k,j) + s(i,j) - M)));

! As variáveis z são binárias;
@for(frentes(i):
  @for(carregadeiras(k):
    @bin(z(k,i))));

! As variáveis y são binárias;
@for(carregadeiras(k):
  @for(frentes(i):
    @for(frentes(j):
      @bin(y(k,i,j)))));

! Exporte o tempo de conclusão, o valor da soma ponderada da antecipação
  e atraso, o vetor de alocação, a antecipação e o atraso. No comando
   abaixo estão faltando o
  valor da soma ponderada, a antecipação e o atraso;
data:
 !   @ole('SFLDS-JIT.xlsx','conclusao','z') = C, z;
@ole('SFLDS-JIT.xlsx','conclusao','JIT','z','t', 'e') = C, JIT, z, t, e;
enddata
```