UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS DE RIO PARANAÍBA
SISTEMAS DE INFORMAÇÃO

GLEIDSON VINÍCIUS GOMES BARBOSA – 6331

PESQUISA OPERACIONAL APLICADA À GESTÃO LOGÍSTICA: DESENVOLVIMENTO DE UM MODELO DE CROSS-DOCKING COM ROTEAMENTO DE VEÍCULOS

RIO PARANAÍBA
2022

Lista de ilustrações

Sumário

1 Introdução

O cross docking é um conceito logístico funciona de forma que quando um produto é comprado, ele será encaminhado a um centro de distribuição que o envia ao cliente por meio de um sistema de redistribuição.

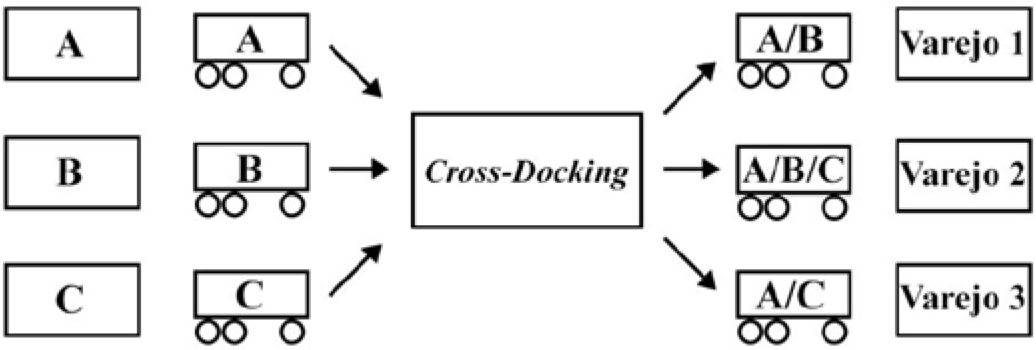

Figura 1: Esquema básico de funcionamento de um *Cross-Docking*

Figura 1 – Esquema de cross docking.

O modelo sobre o qual trabalhamos Ananias et al. (2021) estimou dados, variáveis e restrições de forma a modelar um problema que ao ser solucionado nos dará os melhores valores para trabalhar em torno de minimizar o atraso com base na importância de um produto. Tais variáveis e dados serão especificados nas próximas seções deste documento onde abordaremos e detalharemos a função de cada um dos componentes desse problema e tentaremos chegar a um resultado satisfatório que irá melhorar o funcionamento desse sistema.

2 Arquivo de entrada de dados

Introduzimos aqui nosso arquivo de entrada de dados, arquivo que fornecerá os dados iniciais para nosso programa trabalhar, ele será melhor explicado no próximo tópico.

```
26 3 3 11
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
CaminhaoEntrada1  0
CaminhaoEntrada2  1
CaminhaoEntrada3  2
CaminhaoSaida1
CaminhaoSaida2
CaminhaoSaida3
Cliente           0.00 0.0 00.00 00 00 0000
Smartphone        1.25 2.5 00.20 10 10 0300
Smartwatch        1.00 2.0 00.10 10 05 0400
Capinha           1.00 2.0 00.05 20 01 5000
Controle          1.00 2.0 00.20 20 02 0500
VideoGame         2.00 4.0 05.00 10 05 1500
Televisao         2.50 5.0 20.00 10 06 1000
Headset           1.25 2.5 00.40 20 05 1000
Notebook          2.25 4.5 06.00 10 10 0350
MouseGamer        1.00 2.0 00.10 05 04 1500
Teclado           1.25 2.5 01.50 05 04 1500
3 3 3 3 500.0 140703128616957LL
```

Figura 2 – Arquivo de entrada de dados.

3 Dados de entrada e variáveis de decisão

3.1 Conjuntos

Para essa implementação trabalharemos com 4 conjuntos sendo representados em nosso arquivo de entrada de dados na primeira linha com seus respectivos tamanhos, sendo eles:

- **T - tamanho 26** – Conjunto discreto dos períodos de tempo, baseando-se em nosso arquivo de entrada e contém os tempos, ele irá variar de 0 a 25.
- **J1 – tamanho 3** – é o conjunto que representa os caminhões do estágio 1, ele contém os caminhões CaminhaoEntrada1, CaminhaoEntrada2 e CaminhaoEntrada3 e os dados dependentes deles.
- **J2 – tamanho 3** – é o conjunto que representa os caminhões do estágio 2, ele contém os caminhões CaminhaoSaida1, CaminhaoSaida2 e CaminhaoSaida3 e os dados dependentes deles.
- **P – tamanho 11** – é o conjunto que representa os produtos, nele o primeiro dado, ou seja, o dado na posição 0 VariavelCliente é usado apenas para comparações envolvendo clientes e suas ordens de entrega. Além deste estarão contidos os produtos Smartphone, Smartwatch, Capinha, Controle, VideoGame, Televisao, Headset, Notebook, Mouse, Teclado todos com seus respectivos dados.

Também Serão utilizados dois subconjuntos A e B que são subconjuntos de P mas são definidos por J1 e J2 respectivamente.
Declaração e leitura dos conjuntos:

```
// Declarando conjuntos de entrada
int T;      //Conjunto discreto dos períodos de tempo. Se inicia em 0
int J1;     //Conjunto dos caminhões do estágio 1. Se inicia em 1
int J2;     //Conjunto dos caminhões do estágio 2. Se inicia em 1
int P;      //Conjunto dos produtos. Se inicia em 1
```

```
// Leitura dos conjuntos
fscanf(fp, "%d", &T);   //Leitura do conjunto T
fscanf(fp, "%d", &J1);  //Leitura do conjunto J2
fscanf(fp, "%d", &J2);  //Leitura do conjunto J1
fscanf(fp, "%d", &P);   //Leitura do conjunto P
```

Figura 3 – Declaração e leitura dos conjuntos.

3.2 Dados de entrada

Definiremos aqui o significado de cada dado de entrada e especificaremos a coluna correspondente a ele no arquivo de dados de entrada, lembrando que as linhas dependentes de cada conjuntos são definidas em nossa primeira linha, o que significa que as após a primeira linha, 26 serão dependentes do conjunto T, 3 do conjunto J1, 3 do conjunto J2 e 11 do conjunto P.

- **Ligados ao conjunto T:**
  - tempo: Tempo para o conjunto dos períodos de tempo (1ª coluna)

- **Ligados ao conjunto J1:**
  - **nomeCaminhaoJ1**: Nome para identificar os caminhões do primeiro estágio (1ª coluna)
  - **r**: Determina a data de início de cada caminhão do estágio 1 (2ª coluna)
  - **A**: Um subconjunto do conjunto P, dado por J2. (Não consegui identificar o local para introduzi-lo)
  - **p1**: Representa o tempo de processamento, apesar de ser dependente, utilizaremos um valor fixo o qual será definido por um somatório dos tempos de carregamento de cada produto do conjunto A. (Definido como variável independente dos índices do conjunto)
  - **n1**: número de caminhões no estágio 1(Definido como variável independente dos índices do conjunto)

- **Ligados ao conjunto J2:**
  - **nomeCaminhaoJ2**: Nome para identificar os caminhões do segundo estágio (1ª coluna)
  - **B**: Um subconjunto do conjunto P, dado por J2. (Não consegui identificar o local para introduzi-lo)
  - **p2**: Representa o tempo de processamento, apesar de ser dependente, utilizaremos um valor fixo o qual será definido por um somatório dos tempos de carregamento de cada produto do conjunto B. (Definido como variável independente dos índices do conjunto)
  - **k**: Capacidade dos caminhões de saída (Definido como variável independente dos índices do conjunto)
  - **n2**: número de caminhões no estágio 2(Definido como variável independente dos índices do conjunto)

- **Ligados ao conjunto P:**
  - **nomeProduto**: Nome para identificar os produtos (1ª coluna)
  - **pd**: É o tempo necessário para o descarregamento do produto no estágio 1 (2ª coluna)

- **pc**: É o tempo necessário para o carregamento do produto no estágio 2 (3ª coluna)
- **d**: Tempo limite para entrega daquele produto (4ª coluna)
- **s**: Peso do produto (5ª coluna)
- **w**: Importância (6ª coluna)
- **n**: Número de produtos (7ª coluna)
- **dc**: Tempo de viagem por cliente, utilizando o conjunto P para determinar clientes. (não incluso)
- **ht**: Horizonte de tempo, é determinado por um somatório de tempo de descarregamento+tempo de carregamento de cada produto (Definido como variável independente dos índices do conjunto)

• **Independentes de conjuntos**
- **m1**: número de docas no estágio 1 (Definido como variável independente dos índices do conjunto)
- **m2**: número de docas no estágio 2 (Definido como variável independente dos índices do conjunto)
- **m**: Número suficientemente grande para comparações, sendo este declarado como long long int (Definido como variável independente dos índices do conjunto)

Declaração, inicialização e leitura dos dados de entrada:

```
int* tempo;//tempo para o conjunto dos períodos de tempo

char** nomeCaminhaoJ1;//Nome dos caminhões do estágio 1
int* r;         //data de início por caminhão do estágio 1
int* A;         //é subconjunto de P, porém se dá por J1;
float p1 = 0;   //tempo de processamento por j1. Na prática é o somatório por produto  pertencente a A do tempo de descarregamento

char** nomeCaminhaoJ2;//Nome dos caminhões do estágio 2
int* B;         //é subconjunto de P, porém se dá por j2;
float p2 = 0;   //tempo de processamento por j2. Na prática é o somatório por produto  pertencente a B do tempo de carregamento

char** nomeProduto; //nome do produto
float* pd;      //Tempo de descarregamento por produto
float* pc;      //Tempo de carregamento por produto
int* d;         //Limite de tempo para entrega por produto
float* s;       //Peso por produto
int* w;         //Importância por produto
int* n;         //numero de produtos por produto
float** dc;     //tempo de viagem por client i, j. Em que i,j pertencem ao conjunto P União com 0

float ht;       // Horizonte de tempo, na prática é o somatório por produto de tempo de carregamento+tempo de descarregamento
int m1;         //Número de docas no estágio 1
int m2;         //Número de docas no estágio 2
int n1;         //Número de caminhões no estágio 1
int n2;         //Número de caminhões no estágio 2
float k;        //capacidade dos caminhões de saída
long long int m;        //Número suficientemente grande
```

Figura 4 – Declaração dos dados de entrada

```
//por T
tempo = new int[T];

//por J1
nomeCaminhaoJ1 = new char*[J1];
for (int j1 = 0; j1 < J1; j1++)
{
    nomeCaminhaoJ1[j1] = new char[51];
}
A = new int[J1];
r = new int[J1];

//por J2
nomeCaminhaoJ2 = new char*[J2];
for (int j2 = 0; j2 < J2; j2++)
{
    nomeCaminhaoJ2[j2] = new char[51];
}
B = new int[J2];

//Por produto
nomeProduto = new char*[P];
for (int p = 0; p < P; p++)
{
    nomeProduto[p] = new char[51];
}
pd = new float[P];
pc = new float[P];
d = new int[P];
s = new float[P];
w = new int[P];
n = new int[P];
dc = new float*[P];
for (int i = 0; i < P; i++)
{
    dc[i] = new float[P];
}
```

Figura 5 – Inicialização dos dados de entrada

```c
// Conjunto T
for (int t = 0; t < T; t++)
{
    fscanf(fp, "%d", &tempo[t]);
}

// Conjunto J1
for (int j1 = 0; j1 < J1; j1++)
{
    fscanf(fp, "%s", nomeCaminhaoJ1[j1]);
    fscanf(fp, "%d", &r[j1]);
}

// Conjunto J2
for (int j2 = 0; j2 < J2; j2++)
{
    fscanf(fp, "%s", nomeCaminhaoJ2[j2]);
}

// Conjunto P
for (int p = 0; p < P; p++) //fscanf(fp, "%f", dc[p]); //Variável comentada por não saber como lidar com a mesma
{
    fscanf(fp, "%s", nomeProduto[p]);
    fscanf(fp, "%f", &pd[p]); //Lendo tempo de descarregamento
    fscanf(fp, "%f", &pc[p]); //Lendo tempo de carregamento
    fscanf(fp, "%f", &s[p]); //Lendo peso do produto
    fscanf(fp, "%d", &d[p]); //Lendo Limite de tempo para entrega
    fscanf(fp, "%d", &w[p]); //Lendo importância
    fscanf(fp, "%d", &n[p]); //Lendo Número de produtos
}

//Variáveis independentes de conjuntos

for (int p = 1; p < P; p++)
{
    ht = pc[p] + pd[p];
}
for (int p = 1; p < sizeof(B); p++)
{
    p1 += pd[p];
}
for (int p = 1; p < sizeof(B); p++)
{
    p2 += pc[p];
}
fscanf(fp, "%d", &m1);
fscanf(fp, "%d", &m2);
fscanf(fp, "%d", &n1);
fscanf(fp, "%d", &n2);
fscanf(fp, "%f", &k);
fscanf(fp, "%lld", &m);
```

Figura 6 – Leitura e definição dos dados de entrada

3.3 Variáveis de decisão

Abordaremos aqui as variáveis de decisão e sua implementação, faremos em ordem de dimensões:

- **rd**: Variável de uma dimensão, representa o tempo que um produto p termina de ser descarregado no estágio 1. Declaração:

```
IloNumVarArray rd(env, P, 0, IloInfinity, ILOFLOAT); // tipo float

// adicionar as variáveis ao modelo
for (int p = 1; p < P; p++) //P só inclui o 0 em caso de tratar com clientes
{
    stringstream var;
    var << "rd[" << p << "]";
    rd[p].setName(var.str().c_str());
    modelo.add(rd[p]);
}
```

Figura 7 – Variável rd

- **atraso**: Variável de uma dimensão, representa o tempo de atraso do produto p. Declaração:

```
IloNumVarArray atraso(env, P, 0, IloInfinity, ILOFLOAT); // tipo float

// adicionar as variáveis ao modelo
for (int p = 0; p < P; p++)// P só inclui o 0 em caso de representar clientes
{
    stringstream var;
    var << "atraso[" << p << "]";
    atraso[p].setName(var.str().c_str());
    modelo.add(atraso[p]);
}
```

Figura 8 – Variável atraso

- **C**: Variável de uma dimensão, representa o tempo de conclusão do carregamendo no estágio 2: Declaração:

```
IloNumVarArray C(env, J2, 0, IloInfinity, ILOFLOAT); // tipo float

// adicionar as variáveis ao modelo
for (int j2 = 0; j2 < J2; j2++)
{
    stringstream var;
    var << "C[" << j2 << "]";
    C[j2].setName(var.str().c_str());
    modelo.add(C[j2]);
}
```

Figura 9 – Variável C

- tentrega: Variável de uma dimensão, tempo em que o produto p foi entregue ao cliente. Declaração:

```
IloNumVarArray tentrega(env, P, 0, IloInfinity, ILOFLOAT); // tipo float

// adicionar as variáveis ao modelo
for (int p = 1; p < P; p++)// P só inclui o 0 em caso de representar clientes
{
    stringstream var;
    var << "tentrega[" << p << "]";
    tentrega[p].setName(var.str().c_str());
    modelo.add(tentrega[p]);
}
```

Figura 10 – Variável tentrega

- **x**: Variável de duas dimensões, variável binária que representa se o caminhão do primeiro estágio começa o processamento no tempo t =0. Declaração:

```
IloNumVarMatrix x(env, J1);
for (int j1 = 0; j1 < J1; j1++)
{
    x[j1] = IloNumVarArray(env, T, 0, 1, ILOINT); // Variável Binária
}
// adicionar as variáveis no modelo
for (int j1 = 0; j1 < J1; j1++)
{
    for (int t = 0; t < T; t++)
    {
        stringstream var;
        var << "x[" << j1 << "][" << t << "]";
        x[j1][t].setName(var.str().c_str());
        modelo.add(x[j1][t]);
    }
}
```

Figura 11 – Variável x

- **y**: Variável de duas dimensões, variável binária que representa se o caminhão do segundo estágio começa o processamento no tempo t =0. Declaração:

```
IloNumVarMatrix y(env, J2);
for (int j2 = 0; j2 < J2; j2++)
{
    y[j2] = IloNumVarArray(env, T, 0, 1, ILOINT);// Variável binária
}
// adicionar as variáveis no modelo
for (int j2 = 0; j2 < J2; j2++)
{
    for (int t = 0; t < T; t++)
    {
        stringstream var;
        var << "y[" << j2 << "][" << t << "]";
        y[j2][t].setName(var.str().c_str());
        modelo.add(y[j2][t]);
    }
}
```

Figura 12 – Variável y

- **tempos**: Variável de duas dimensões, representa o tempo de viagem do caminhão do segundo estágio de um cliente ao outro, sendo o conjunto P utilizando a variavelCliente utilizado para representar os clientes. Declaração:

```
IloNumVarMatrix tempos(env, J2);
for (int j2 = 0; j2 < J2; j2++)
{
    tempos[j2] = IloNumVarArray(env, P, 0, IloInfinity, ILOFLOAT); // tipo float
}
// adicionar as variáveis no modelo
for (int j2 = 0; j2 < J2; j2++)
{
    for (int n = 0; n < P; n++) // P se inicia em 0 em caso de P representar clientes
    {
        stringstream var;
        var << "tempos[" << j2 << "][" << n << "]";
        tempos[j2][n].setName(var.str().c_str());
        modelo.add(tempos[j2][n]);
    }
}
```

Figura 13 – Variável tempos

- **rotas**: Variável de três dimensões, variável binária que representa se o caminhão do segundo estágio faz o percurso entre os clientes indicados, sendo o conjunto P utilizando a variavelCliente utilizado para representar os clientes. Declaração:

```
IloNumVar3Matrix rotas(env, J2);
for (int j2 = 0; j2 < J2; j2++)
{
    rotas[j2] = IloNumVarMatrix(env, P);
    for (int i = 0; i < P; i++)
    {
        rotas[j2][i] = IloNumVarArray(env, P, 0, 1, ILOINT); // Variável Binária
    }
}
```

Figura 14 – Variável

4 Função Objetivo e Restrições

4.1 Função Objetivo

Mesmo com tantas variáveis e dados a serem analisados, nossa função objetivo é extremamente simples, ela se resume a minimizar o somatório do produto entre atraso e importância para cada produto.

$$MIN : \sum_{p \in P} atraso_p * w_p \tag{4.1}$$

Declaração da função Objetivo:

```
IloExpr fo(env);

for (int p = 1; p < P; p++) //Somatório por p de P, atraso+importância, P só inclui o 0 em caso de representar com clientes
{
    fo += atraso[p] * w[p];
}


//IloMinimize e IloMaximize
modelo.add(IloMinimize(env, fo));
```

Figura 15 – Declaração da Função Objetivo

4.2 Restrições

Aqui abordaremos nossas várias restrições para alcançarmos o objetivo de nossa função:

$$(1) \sum_{t=0}^{T-p1_j} t * x_{jt} \geq r_j, \forall j \in J_1 \tag{4.2}$$

Essa restrição garante que os caminhões do primeiro estágio só podem ser alocados após chegarem ao centro de distribuição. Declaração:

```
//Restrição(1)

for (int j1; j1 < J1; j1++)//para todo
{
    IloExpr soma(env);
    for (int t; t < (T - p1);)
    {
        soma += t*x[j1][t];
    }
    //declarar a restição
    IloRange rest_um(env, r[j1], soma, IloInfinity);
    //nomeando a restrição
    stringstream rest;
    rest << "um: ";
    rest_um.setName(rest.str().c_str);
    //adicionar ao modelo
    modelo.add(rest_um);
}
```

Figura 16 – Primeira restrição

$$(2)\ \sum_{t=0}^{T-p1_j} t * x_{jt} = 1, \forall j \in J_1 \tag{4.3}$$

Essa restrição garante que cada caminhão do primeiro estágio só seja alocado em uam data no horizonte de tempo. Declaração:

```
//Restrição(2)

for (int j1; j1 < J1; j1++)//para todo
{
    IloExpr soma(env);
    for (int t; t < (T - p1);)
    {
        soma += t*x[j1][t];
    }
    //declarar a restição
    IloRange rest_dois(env, 1, soma, 1);
    //nomeando a restrição
    stringstream rest;
    rest << "dois: ";
    rest_dois.setName(rest.str().c_str);
    //adicionar ao modelo
    modelo.add(rest_dois);
}
```

Figura 17 – Segunda restrição

$$(3)\ \sum_{t=0}^{T-p2_j} t * y_{jt} = 1, \forall j \in J_2 \tag{4.4}$$

Essa restrição garante que cada caminhão do segundo estágio só seja alocado em uma data do horizonte de tempo. Declaração:

```
//Restrição(3)

for (int j2; j2 < J2; j2++)//para todo
{
    IloExpr soma(env);
    for (int t; t < (T - p2);)
    {
        soma += t*y[j2][t];
    }
    //declarar a restição
    IloRange rest_tres(env, 1, soma, 1);
    //nomeando a restrição
    stringstream rest;
    rest << "tres: ";
    rest_tres.setName(rest.str().c_str);
    //adicionar ao modelo
    modelo.add(rest_tres);
}
```

Figura 18 – Terceira restrição

$$(4) \sum_{j \in J_1} \sum_{s=max(0;t-p1_j+1)}^{t} x_{js} \leq m_1, \forall t \in T \tag{4.5}$$

Essa restrição garante que o número de caminhões sendo processados no primeiro estágio é menor ou igual ao número de docas. Declaração:

```
//Restrição(4)
for (int t = 0; t < T; t++)//para todo
{
    IloExpr soma(env);
    for (int j1 = 0; j1 < J1; j1++)
    {
        for (int  s = fmax(0,(t-p1+1)); s < t; s++)
        {
            soma += x[j1][s];
        }
    }
    //declarar a restição
    IloRange rest_quatro(env, -IloInfinity, soma, m1);
    //nomeando a restrição
    stringstream rest;
    rest << "quatro: ";
    rest_quatro.setName(rest.str().c_str);
    //adicionar ao modelo
    modelo.add(rest_quatro);
}
```

Figura 19 – Quarta restrição

$$(5)\sum_{j\in J_2}\sum_{s=max(0;t-p2_j+1)}^{t} y_{js} \leq m_2, \forall t \in T \tag{4.6}$$

Essa restrição garante que o número de caminhões sendo processados no segundo estágio é menor ou igual ao número de docas. Declaração:

```
//Restrição(5)

for (int t = 0; t < T; t++)//para todo
{
    IloExpr soma(env);
    for (int j2 = 0; j2 < J2; j2++)
    {
        for (int s = fmax(0, (t - p2 + 1)); s < t; s++)
        {
            soma += y[j2][s];
        }
    }
    //declarar a restrição
    IloRange rest_cinco(env, -IloInfinity, soma, m2);
    //nomeando a restrição
    stringstream rest;
    rest << "cinco: ";
    rest_cinco.setName(rest.str().c_str);
    //adicionar ao modelo
    modelo.add(rest_cinco);
}
```

Figura 20 – Quinta restrição

$$(6) rd_p \geq \sum_{t\in T}(t + p1_j) * x_{jt}, \forall p \in A_j, \forall j \in J_1 \tag{4.7}$$

Essa restrição garante que a data de disponibilidade de cada produto para ser carregado no segundo estágio é maior ou igual ao tempo de conclusão de processamento do caminhão no qual o produto estava carregado no primeiro estágio. Declaração:

```
//Restrição(6)

for (int j1 = 0; j1 < J1; j1++)//para todo
{
    for (int p; p < A[j1]; p++)//para todo
    {
        IloExpr soma(env);
        for (int t = 0; t < T; t++)
        {
            soma += (t + p1)*x[j1][t];
        }
        //declarar a restrição
        IloRange rest_seis(env, 0, rd[p]-soma, IloInfinity);
        //nomeando a restrição
        stringstream rest;
        rest << "seis: ";
        rest_seis.setName(rest.str().c_str());
        //adicionar ao modelo
        modelo.add(rest_seis);
    }
}
```

Figura 21 – Sexta restrição

$$(7)\ \sum_{t=0}^{T-p2_j} t * y_{jt} \geq rd_p, \forall p \in B_j, \forall j \in J_2 \tag{4.8}$$

Essa restrição garante que os caminhões no segundo estágio só podem iniciar seu processamento após seus produtos estarem liberados para o carregamento. Declaração:

```
//Restrição(7)

    for (int j2 = 0; j2 < J2; j2++)//para todo
    {
        for (int p; p < B[j2]; p++)//para todo
        {
            IloExpr soma(env);
            for (int t = 0; t < (T-p2); t++)
            {
                soma += t*y[j2][t];
            }
            //declarar a restrição
            IloRange rest_sete(env, 0, soma-rd[p], IloInfinity);
            //nomeando a restrição
            stringstream rest;
            rest << "sete: ";
            rest_sete.setName(rest.str().c_str());
            //adicionar ao modelo
            modelo.add(rest_sete);
        }
    }
```

Figura 22 – Sétima restrição

$$(8)C_j \geq p2_j + \sum_{t=0}^{T-p2_j} t * y_{jt}, \forall p \in B_j, \forall j \in J_2 \tag{4.9}$$

Essa restrição garante que a data de conclusão de um caminhão do segundo estágio é maior ou igual a data de início do mesmo, mais o tempo de processamento de um caminhão j no segundo estágio. Declaração:

```
//Restrição(8)

    for (int j2 = 0; j2 < J2; j2++)//para todo
    {
        for (int p = 0; p < (sizeof(B)/sizeof(B[0])); p++)//para todo
        {
            IloExpr soma(env);
            for (int t = 0; t < (T - p2); t++)
            {
                soma += t*y[j2][t];
            }
            soma += p2;
            //declarar restrição
            IloRange rest_oito(env, 0, C[j2]-soma, IloInfinity);
            //nomeando a restrição
            stringstream rest;
            rest << "oito: ";
            rest_oito.setName(rest.str().c_str());
            //adicionar ao modelo
            modelo.add(rest_oito);

        }
    }
```

Figura 23 – Oitava restrição

$$(9) \sum_{p \in P} rotas_{j,0,p} = 1, \forall j \in J_2 \tag{4.10}$$

Essa restrição garante que o trajeto dos caminhões do segundo estágio inicia seus trajetos no centro de Cross-Docking (CD), que é o ponto de origem, onde ocorre o carregamento dos produtos. Declaração:

```
//Restrição(9)

for (int j2 = 0; j2 < J2; j2++)//para todo
{
    IloExpr soma(env);
    for (int p = 1; p < P; p++)
    {
        soma += rotas[j2][0][p];
    }
    //declarar restrição
    IloRange rest_nove(env, 1, soma, 1);
    //nomeando a restrição
    stringstream rest;
    rest << "nove: ";
    rest_nove.setName(rest.str().c_str);
    //adicionar ao modelo
    modelo.add(rest_nove);
}
```

Figura 24 – Nona restrição

$$(10) \sum_{i \in \{0\} \cup P | p \neq i} rotas_{jip} = 1, \forall p \in B_j, \forall j \in J_2 \tag{4.11}$$

$$(11) \sum_{p \in \{0\} \cup P | p \neq i} rotas_{jip} = 1, \forall i \in B_j, \forall j \in J_2 \tag{4.12}$$

Ambas as restrições juntas garantem que todos os clientes de cada caminhão serão visitados uma única vez. Declaração:

```
//Restrição(10)
for (int j2 = 0; j2 < J2; j2++)//para todo
{
    for (int p = 0; p < B[j2]; j2++)//para todo
    {
        IloExpr soma(env);
        for (int i = 0; i < P; i++)
        {
            if (p != i)
            {
                soma += rotas[j2][i][p];
            }
        }
        //declarar restrição
        IloRange rest_dez(env, 1, soma, 1);
        //nomeando a restrição
        stringstream rest;
        rest << "dez: ";
        rest_dez.setName(rest.str().c_str);
        //adicionar ao modelo
        modelo.add(rest_dez);

    }
}
```

```
//Restrição(11)
for (int j2 = 0; j2 < J2; j2++)//para todo
{
    for (int i = 0; i < B[j2]; j2++)//para todo
    {
        IloExpr soma(env);
        for (int p = 0; p < P; i++)
        {
            if (p!=i)
            {
                soma += rotas[j2][i][p];
            }
        }
        //declarar restrição
        IloRange rest_onze(env, 1, soma, 1);
        //nomeando a restrição
        stringstream rest;
        rest << "onze: ";
        rest_onze.setName(rest.str().c_str);
        //adicionar ao modelo
        modelo.add(rest_onze);

    }
}
```

Figura 25 – Décima e décima primeira restrições

$$(12)tentrega_P \geq tempos_{jp} + C_j, \forall p \in P, \forall j \in J_2 \tag{4.13}$$

Essa restrição garante que o tempo de entrega do produto "p" é maior ou igual ao tempo de conclusão de carregamento do caminhão que contém esse produto (Cj,) somado ao tempo de viagem até o ponto de entrega. Declaração:

```
//Restrição(12)

for (int j2; j2 < J2; j2++) // para todo j2
{
    for (int p; p < P; p++)//para todo p
    {
        IloExpr soma(env);
        soma = tempos[j2][p] + C[j2];
        //declarar restrição
        IloRange rest_doze(env, 0 , tentrega[p]- soma, IloInfinity);
        //nomeando a restrição
        stringstream rest;
        rest << "doze: ";
        rest_doze.setName(rest.str().c_str());
        //adicionar ao modelo
        modelo.add(rest_doze);
    }
}
```

Figura 26 – Décima segunda restrição

$$(13)atraso_p \geq tentrega_p - d_p, \forall p \in P \tag{4.14}$$

Essa restrição garante que o atraso de entrega do produto p é maior ou igual a diferença do tempo de entrega do produto "p" pelo tempo esperado de entrega do produto "p". Declaração:

```
/Restrição(13)

    for (int p=0; p < P; p++)//para todo
    {
        IloExpr sub(env);
        sub += tentrega[p] - d[p];
        //declarar a restrição
        IloRange rest_treze(env, 0, atraso[p] - sub, IloInfinity);
        //nomeando a restrição
        stringstream rest;
        rest << "treze: ";
        rest_treze.setName(rest.str().c_str());
        //adicionar ao modelo
        modelo.add(rest_treze);
    }
```

Figura 27 – Décima terceira restrição

$$(14) tempos_{jn} \geq tempos_{ji} + dc_{in} - M * (1 - rotas_{jin}), \forall n, i \in N \wedge n \neq 0, \forall j \in J_2 \quad (4.15)$$

A restrição (14) garante que o tempo de chegada no próximo cliente deve ser maior do que o tempo de chegada ao cliente anterior somado ao tempo de viagem até o cliente atual. Declaração:

```
//Restrição(14)

for (int j2 = 0; j2 < J2; j2++)//para todo
{
    for (int n; n < N; n++)//para todo
    {
        for (int i; i < N; i++)
        {
            IloExpr soma(env);
            if (n != 0)
            {
                soma = tempos[j2][i] + dc[i][n] - m*(1 - rotas[j2][i][n]);
            }
            IloRange rest_quatorze(env, -IloInfinity, soma, tempos[j2][n]);
            //nomeando a restrição
            stringstream rest;
            rest << "quatorze: ";
            rest_quatorze.setName(rest.str().c_str);
            //adicionar ao modelo
            modelo.add(rest_quatorze);

        }
    }
}
```

Figura 28 – Décima quarta restrição

As demais restrições de nosso modelo são restrições de não negatividade e de variáveis binárias, todas essas são garantidas ao definir as variáveis de decisão.

Não Negatividade

$$(15)\ rd_p \geq 0, \forall p \in P$$

$$(16)\ atraso_p \geq 0, \forall p \in P$$

$$(17)\ tentrega_p \geq 0, \forall p \in P$$

$$(18)\ c_j \geq 0, \forall j \in J_2$$

$$(19)\ tempos_{jn} \geq 0, \forall j \in J_2, \forall n \in \{0\} \cup P$$

Variáveis binárias

$$(20)\ x_{jt} \in \{0,1\}, \forall j \in J_1, \forall t \in T$$

$$(21)\ y_{jt} \in \{0,1\}, \forall j \in J_2, \forall t \in T$$

$$(22)\ rotas_{jik} \in \{0,1\}, \forall j \in J_2, \forall i, k \in P.$$

5 Finalizando a implementação

5.1 Impressão dos dados gerados

Apesar de nosso problema não ter sido viável por alguns problemas do modelo utilizado, todo o procedimento foi feito inclusive para impressão das variáveis e valor ótimo da FO. Implementação:

```
for (int p = 0; p < P; p++)//Variáveis de uma dimensão dependentes de P
{
    float variavel1 = cplex.getValue(rd[p]);
    float variavel2 = cplex.getValue(atraso[p]);
    float variavel4 = cplex.getValue(tentrega[p]);
    printf("********\n");
    printf("Tempo em que o produto %s termina de ser descarregado no estagio 1: %f\n", nomeProduto[p], variavel1);
    printf("********\n");
    printf("Tempo de atraso do produto %s: %f\n", nomeProduto[p], variavel2);
    printf("********\n");
    printf("Tempo em que o produto %s foi entregue: %f\n", nomeProduto[p], variavel4);
}

for (int j2 = 0; j2 < J2; j2++)//C
{

    float variavel = cplex.getValue(C[j2]);
    printf("********\n");
    printf("Tempo em que o caminhao %s termina de ser carregado no estagio 2: %f\n", nomeCaminhaoJ2[j2], variavel);
}

for (int j1 = 0; j1 < J1; j1++)//X
{
    for (int t = 0; t < T; t++)
    {
        int variavel = cplex.getValue(x[j1][t]);
        printf("********\n");
        printf("1- Verdadeiro  0- Falso\nO caminhao %s comeca o processamento no tempo 0: %d\n", nomeCaminhaoJ1[j1], variavel);
    }
}

for (int j2 = 0; j2 < J2; j2++)//Y
{
    for (int t = 0; t < T; t++)
    {
        int variavel = cplex.getValue(y[j2][t]);
        printf("********\n");
        printf("1- Verdadeiro  0- Falso\nO caminhao %s comeca o processamento no tempo 0: %d\n", nomeCaminhaoJ2[j2], variavel);
    }
}

for (int j2 = 0; j2 < J2; j2++)//tempos
{
    for (int p = 0; p < P; p++)
    {
        float variavel = cplex.getValue(tempos[j2][p]);
        printf("********\n");
        printf("Tempo de viagem de um cliente ao outro do caminhao %s: %.2f\n", nomeCaminhaoJ2[j2], variavel);
    }
}
```

```
for (int j2 = 0; j2 < J2; j2++)//rotas
{
    for (int i = 0; i < P; i++)
    {
        for (int k = 0; k < P; k++)
        {
            int variavel = cplex.getValue(rotas[j2][i][k]);
            printf("********\n");
            printf("1- Verdadeiro  0- Falso\nO caminhao %s faz o percurso entre os clientes %d e %d: %d\n", nomeCaminhaoJ2[j2], i, k, variavel);
        }

    }
}


printf("Funcao objetivo: %.2f\n", cplex.getObjValue());
```

Figura 29 – Exibição dos valores das variáveis de decisão e solução ótima.

Referências

ANANIAS, L. F. N. et al. **Pesquisa operacional aplicada à gestão logística: desenvolvimento de um modelo de Cross-Docking com roteamento de veículos**. 2021. Disponível em: <https://proceedings.science/sbpo-series/sbpo-2021/papers/pesquisa-operacional-aplicada-a-gestao-logistica--desenvolvimento-de-um-modelo-de-cross-docking-c/>. Acesso em: 18 mar. 2022. Citado na página 3.