

UNIVERSIDADE FEDERAL DE VIÇOSA  
*CAMPUS* DE RIO PARANAÍBA  
SISTEMAS DE INFORMAÇÃO

GLEIDSON VINÍCIUS GOMES BARBOSA – 6331  
MARCELA ROCHA SOUZA - 5443  
BRUNO CAETANO DE OLIVEIRA - 6683

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

RIO PARANAÍBA

2022

# Lista de ilustrações

Figura 1 – Esquema de cross docking. . . . .	4
Figura 2 – Dados de entrada. . . . .	5
Figura 3 – Arquivo de entrada de dados parte 1. . . . .	6
Figura 4 – Arquivo de entrada de dados parte 2. . . . .	7
Figura 5 – Declaração e leitura dos conjuntos. . . . .	9
Figura 6 – Declaração dos dados de entrada . . . . .	11
Figura 7 – Inicialização dos dados de entrada . . . . .	12
Figura 8 – Leitura e definição dos dados de entrada parte 1. . . . .	13
Figura 9 – Leitura e definição dos dados de entrada parte 2. . . . .	14
Figura 10 – Variável rd . . . . .	14
Figura 11 – Variável atraso . . . . .	15
Figura 12 – Variável C . . . . .	15
Figura 13 – Variável tentrega . . . . .	15
Figura 14 – Variável x . . . . .	16
Figura 15 – Variável y . . . . .	16
Figura 16 – Variável tempos . . . . .	17
Figura 17 – Variável . . . . .	17
Figura 18 – Declaração da Função Objetivo . . . . .	18
Figura 19 – Funções necessárias para auxiliar nas restrições. . . . .	18
Figura 20 – Primeira restrição . . . . .	19
Figura 21 – Segunda restrição . . . . .	19
Figura 22 – Terceira restrição . . . . .	20
Figura 23 – Quarta restrição . . . . .	20
Figura 24 – Quinta restrição . . . . .	21
Figura 25 – Sexta restrição . . . . .	22
Figura 26 – Sétima restrição . . . . .	22
Figura 27 – Oitava restrição . . . . .	23
Figura 28 – Nona restrição . . . . .	23
Figura 29 – Décima e décima primeira restrições . . . . .	24
Figura 30 – Décima segunda restrição . . . . .	24
Figura 31 – Décima terceira restrição . . . . .	25
Figura 32 – Décima quarta restrição . . . . .	25
Figura 33 – Execução do programa parte 1. . . . .	27
Figura 34 – Execução do programa parte 2. . . . .	28

Figura 35 – Execução do programa parte 3. . . . .	29
Figura 36 – Execução do programa modificado. . . . .	29
Figura 37 – Exibição dos valores das variáveis de decisão e solução ótima. . . . .	30

# Sumário

<b>1</b>	<b>Introdução . . . . .</b>	<b>4</b>
<b>2</b>	<b>Arquivo de entrada de dados . . . . .</b>	<b>5</b>
<b>3</b>	<b>Dados de entrada e variáveis de decisão . . . . .</b>	<b>8</b>
3.1	Conjuntos . . . . .	8
3.2	Dados de entrada . . . . .	9
3.3	Variáveis de decisão . . . . .	14
<b>4</b>	<b>Função Objetivo e Restrições . . . . .</b>	<b>18</b>
4.1	Função Objetivo . . . . .	18
4.2	Restrições . . . . .	18
<b>5</b>	<b>Rodando o programa . . . . .</b>	<b>27</b>
5.1	Tentativa com todas as restrições . . . . .	27
5.2	Excluindo as restrições Um e Dois . . . . .	29
<b>6</b>	<b>Finalizando a implementação . . . . .</b>	<b>30</b>
6.1	Impressão dos dados gerados . . . . .	30
	<b>Referências . . . . .</b>	<b>31</b>

# 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.

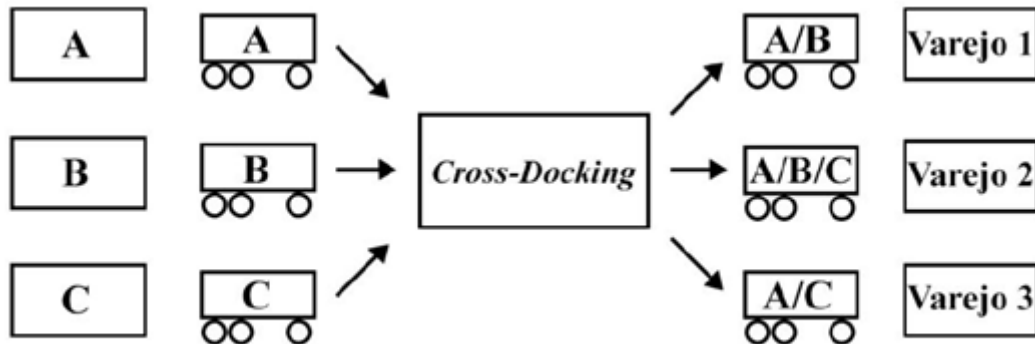


Figura 1: Esquema básico de funcionamento de um *Cross-Docking*

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.

Dados de Entrada											
T			Definido pelo horizonte de tempo HT.								
Período	Definidos por um for de 1 em 1 até o horizonte de tempo										
Tempo											
J1			3								
CaminhãoEntrada 1	P1j 345	Data de Início 0									
Aj Produto por caminhão de entrada	Smartphone 20	Smartwatch 10	Capinha 100	Controle 25	VideoGame 50	Televisão 30	Headset 30	Notebook 12	Mouse Gamer 20	Teclado 40	
CaminhãoEntrada2	P1j 553	Data de Início 1									
Aj Produto por caminhão de entrada	Smartphone 8	Smartwatch 15	Capinha 300	Controle 20	VideoGame 50	Televisão 35	Headset 40	Notebook 13	Mouse Gamer 60	Teclado 30	
CaminhãoEntrada3	P1j 322	Data de Início 2									
Aj Produto por caminhão de entrada	Smartphone 2	Smartwatch 15	Capinha 100	Controle 5	VideoGame 50	Televisão 35	Headset 30	Notebook 10	Mouse Gamer 70	Teclado 80	
J2			3								
CaminhãoSaída1	P2j 650										
Bj Produto por caminhão de saída	Smartphone 10	Smartwatch 15	Capinha 100	Controle 25	VideoGame 40	Televisão 30	Headset 40	Notebook 10	Mouse Gamer 30	Teclado 20	
CaminhãoSaída2	P2j 910										
Bj Produto por caminhão de saída	Smartphone 10	Smartwatch 15	Capinha 200	Controle 20	VideoGame 60	Televisão 30	Headset 30	Notebook 10	Mouse Gamer 60	Teclado 80	
CaminhãoSaída3	P2j 870										
Bj Produto por caminhão de saída	Smartphone 10	Smartwatch 10	Capinha 200	Controle 5	VideoGame 50	Televisão 40	Headset 30	Notebook 15	Mouse Gamer 60	Teclado 50	
P			11								
Nome do produto	Tempo de descarregamento	Tempo de carregamento	Tempo limite para entrega	Peso do produto	Importância	Número de produtos					
Smartphone	1	2	10,00	0,2	10	30					
Smartwatch	1	2	10	0,1	5	40					
Capinha	1	2	20	0,05	1	500					
Controle	1	2	20	0,2	2	50					
VideoGame	2	4	10	5	5	150					
Televisao	2	5	10	20	6	100					
Headset	1	2	20	0,4	5	100					
Notebook	2	4	10	6	10	35					
MouseGamer	1	2	5	0,1	4	150					
Teclado	1	2	5	1,5	4	150					

Cliente i j	Tempo de Viagem do cliente i ao J										
0	1	2	3	4	5	6	7	8	9	10	
1	0	1	2	3	4	5	6	7	8	9	
2	1	0	1	2	3	4	5	6	7	8	
3	2	1	0	1	2	3	4	5	6	7	
4	3	2	1	0	1	2	3	4	5	6	
5	4	3	2	1	0	1	2	3	4	5	
6	5	4	3	2	1	0	1	2	3	4	
7	6	5	4	3	2	1	0	1	2	3	
8	7	6	5	4	3	2	1	0	1	2	
9	8	7	6	5	4	3	2	1	0	1	
10	9	8	7	6	5	4	3	2	1	0	

Número de caminhões do primeiro estágio	3										
Número de Caminhões do Segundo Estágio	3										
Número de Docas do primeiro estágio	3										
Número de Docas do segundo estágio	3										
Capacidade de caminhões de saída	500										
Número suficientemente grande para comparar	140703128516957										

Figura 2 – Dados de entrada.

```
3 3 11
CaminhaoEntrada1 0
20
10
100
25
50
30
30
12
20
40
CaminhaoEntrada2 1
8
15
300
20
50
35
40
13
60
30
CaminhaoEntrada3 2
2
15
100
5
50
35
30
10
70
80
CaminhaoSaida1
10
15
100
25
40
30
40
10
30
20
```

Figura 3 – Arquivo de entrada de dados parte 1.

```

CaminhaoSaida2
10
15
200
20
60
30
30
10
60
80
CaminhaoSaida3
10
10
200
5
50
40
30
15
60
50
Cliente          00.00 0 0 00 00 000
Smartphone       00.20 1 2 10 10 030
Smartwatch       00.10 1 2 10 05 040
Capinha          00.05 1 2 20 01 500
Controle         00.20 1 2 20 02 050
VideoGame        05.00 2 4 10 05 150
Televisao        20.00 2 4 10 06 100
Headset          00.40 1 2 20 05 100
Notebook         06.00 2 4 10 10 035
MouseGamer       00.10 1 4 05 04 150
Teclado          01.50 1 4 05 04 150
0      1      2      3      4      5      6      7      8      9      10
1      0      1      2      3      4      5      6      7      8      9
2      1      0      1      2      3      4      5      6      7      8
3      2      1      0      1      2      3      4      5      6      7
4      3      2      1      0      1      2      3      4      5      6
5      4      3      2      1      0      1      2      3      4      5
6      5      4      3      2      1      0      1      2      3      4
7      6      5      4      3      2      1      0      1      2      3
8      7      6      5      4      3      2      1      0      1      2
9      8      7      6      5      4      3      2      1      0      1
10     9      8      7      6      5      4      3      2      1      0
3 3 3 3 500.0 140703128616957LL

```

Figura 4 – Arquivo de entrada de dados parte 2.



## 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 de 3 deles na primeira linha com seus respectivos tamanhos, com exceção do conjunto T que será dado por ht, sendo eles:

- **T - tamanho determinado pelo horizonte de tempo**– Conjunto discreto dos períodos de tempo, ele será definido pelos nossos próprios dados de entrada visto que ele se dará de 1 em 1 com o máximo do horizonte de tempo.
- **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", &J1); //Leitura do conjunto J2
fscanf(fp, "%d", &J2); //Leitura do conjunto J1
fscanf(fp, "%d", &P);  //Leitura do conjunto P

for (int p = 1; p < P; p++)
{
    ht += n[p] * (pc[p] + pd[p]);
}
T = ht; // o maior tempo é definido pelo horizonte de tempo
```

Figura 5 – 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 (dados por ht, trabalhando de 1 em 1)
- **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. Este conjunto contem a quantidade de determinado produto carregado naquele caminhão.
  - **nA**: A quantidade de cada produto carregado naquele caminhão.
  - **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. Este conjunto contém a quantidade de determinado produto carregado naquele caminhão.
- **nB:** A quantidade de cada produto carregado naquele caminhão.
- **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. Incluso nos dados de entrada por meio de uma matriz.
- **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:

```

// Dados de entrada dependentes dos conjuntos

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
int** nA; //numero de produtos carregados em A
int* p1; //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
int** nB; //numero de produtos carregados em B
int* p2; //tempo de processamento por j2. Na prática é o somatório por produto pertencente a B do tempo de carregamento

char** nomeProduto; //nome do produto
int* pd; //Tempo de descarregamento por produto
int* 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
int** dc; //tempo de viagem por client i, j. Em que i,j pertencem ao conjunto P União com 0

int ht = 0; // 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
int k; //capacidade dos caminhões de saída
long long int m; //Número suficientemente grande

```

Figura 6 – Declaração dos dados de entrada

```
//por T Definidos após a leitura de dados pois são dependentes deles

//por J1
nomeCaminhaoJ1 = new char* [J1];
for (int j1 = 0; j1 < J1; j1++)
{
    nomeCaminhaoJ1[j1] = new char[51];
}
A = new int* [J1];
for (int j1 = 0; j1 < J1; j1++)
{
    A[j1] = new int[P];
}
nA = new int* [J1];
for (int j1 = 0; j1 < J1; j1++)
{
    nA[j1] = new int[P];
}
r = new int[J1];
p1 = 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];
for (int j2 = 0; j2 < J2; j2++)
{
    B[j2] = new int[P];
}
nB = new int* [J2];
for (int j2 = 0; j2 < J2; j2++)
{
    nB[j2] = new int[P];
}
p2 = new int[J2];

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

Figura 7 – Inicialização dos dados de entrada

```
// Conjunto T será definido após a leitura de dados pois este depende dos dados de entrada

// Conjunto J1
for (int j1 = 0; j1 < J1; j1++)
{
    fscanf(fp, "%s", nomeCaminhaoJ1[j1]);
    fscanf(fp, "%d", &r[j1]);
    for (int p = 1; p < P; p++)
    {
        fscanf(fp, "%d", &nA[j1][p]); //Lendo Número de produtos carregados em A
    }
}

// Conjunto J2
for (int j2 = 0; j2 < J2; j2++)
{
    fscanf(fp, "%s", nomeCaminhaoJ2[j2]);
    for (int p = 1; p < P; p++)
    {
        fscanf(fp, "%d", &nB[j2][p]); //Lendo Número de produtos carregados em B
    }
}

// Conjunto P
for (int p = 0; p < P; p++)
{
    fscanf(fp, "%s", nomeProduto[p]);
    fscanf(fp, "%f", &s[p]); //Lendo peso do produto
    fscanf(fp, "%d", &pd[p]); //Lendo tempo de descarregamento
    fscanf(fp, "%d", &pc[p]); //Lendo tempo de carregamento
    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
}
for (int i = 0; i < P; i++)
{
    for (int j = 0; j < P; j++)
    {
        fscanf(fp, "%d", &dc[i][j]); //Lendo a matriz de tempo de viagem do cliente i ao j
    }
}
```

Figura 8 – Leitura e definição dos dados de entrada parte 1.

```

//Variáveis independentes de conjuntos

for (int p = 1; p < P; p++)
{
    ht += n[p] * (pc[p] + pd[p]);
}
T = ht; // o maior tempo é definido pelo horizonte de tempo
tempo = new int[T]; // Inicializando neste local pois depende do valor do horizonte de tempo
for (int i = 0; i < ht; i++)
{
    tempo[i] = i; //o conjunto discreto de tempo é definido pelos tempo de 0 ao horizonte de tempo
}
for (int j1 = 0; j1 < J1; j1++)
{
    for (int p = 1; p < sizeof(B); p++)
    {
        p1[j1] += pd[p]*nA[j1][p]; //calculando p1
    }
}
for (int j2 = 0; j2 < J2; j2++)
{
    for (int p = 1; p < sizeof(B); p++)
    {
        p2[j2] += pc[p]*nB[j2][p]; //calculando p1
    }
}
fscanf(fp, "%d", &m1);
fscanf(fp, "%d", &m2);
fscanf(fp, "%d", &n1);
fscanf(fp, "%d", &n2);
fscanf(fp, "%d", &k);
fscanf(fp, "%lld", &m);

```

Figura 9 – Leitura e definição dos dados de entrada parte 2.

### 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 10 – 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 11 – Variável atraso

- C: Variável de uma dimensão, representa o tempo de conclusão do carregamento 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 12 – Variável C

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

```
IloNumVarArray entrega(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 << "entrega[" << p << "]";
    entrega[p].setName(var.str().c_str());
    modelo.add(entrega[p]);
}
```

Figura 13 – Variável entrega



- **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, ILOBOOL); // 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 14 – 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, ILOBOOL); // 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 15 – 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 variávelCliente 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 16 – 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, ILOBOOL); // Variável Binária
    }
}
// adicionar as variáveis no modelo
for (int j2 = 0; j2 < J2; j2++)
{
    for (int i = 0; i < P; i++) // P se inicia em 0 em caso de P representar clientes
    {
        for (int K = 0; K < P; K++)
        {
            stringstream var;
            var << "rotas[" << j2 << "][" << i << "][" << K << "];";
            rotas[j2][i][K].setName(var.str().c_str());
            modelo.add(rotas[j2][i][K]);
        }
    }
}

```

Figura 17 – 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.

$$MIN : \sum_{p \in P} atraso_p * w_p \quad (4.1)$$

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 18 – Declaração da Função Objetivo

### 4.2 Restrições

Aqui abordaremos nossas várias restrições para alcançarmos o objetivo de nossa função, utilizaremos duas funções declaradas para auxiliar nas restrições, são elas:

```
int tamanho(int arr[])//Função para definir o tamanho de um array
{
    return sizeof arr / sizeof arr[0];
}

int maior(int a, int b)//Função para determinar o maior dentre 2 números
{
    if (a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}
```

Figura 19 – Funções necessárias para auxiliar nas restrições.

Informadas as funções extras que utilizaremos, seguem as restrições:

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

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

```
for (int j1 = 0; j1 < J1; j1++)//para todo
{
    IloExpr soma(env);
    for (int t = 0; t < (ht - p1[j1]); t++)
    {
        soma += tempo[t] * x[j1][t];
    }
    //declarar a restrição
    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 20 – Primeira restrição

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

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

```
for (int j1 = 0; j1 < J1; j1++)//para todo
{
    IloExpr soma(env);
    for (int t = 0; t < (ht - p1[j1]); t++)
    {
        soma += tempo[t] * x[j1][t];
    }
    //declarar a restrição
    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 21 – Segunda restrição

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

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

```
for (int j2 = 0; j2 < J2; j2++)//para todo
{
    IloExpr soma(env);
    for (int t = 0; t < (ht - p2[j2]); t++)
    {
        soma += tempo[t] * y[j2][t];
    }
    //declarar a restrição
    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 22 – Terceira restrição

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

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

```
for (int t = 0; t < T; t++)//para todo
{
    IloExpr soma(env);
    for (int j1 = 0; j1 < J1; j1++)
    {
        for (int s = maior(0, (t - p1[j1] + 1)); s < t; s++)
        {
            soma += x[j1][s];
        }
    }
    //declarar a restrição
    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 23 – Quarta restrição

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

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

```
for (int t = 0; t < T; t++)//para todo
{
    IloExpr soma(env);
    for (int j2 = 0; j2 < J2; j2++)
    {
        for (int s = maior(0, (t - p2[j2] + 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 24 – Quinta restrição

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

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:

```

for (int j1 = 0; j1 < J1; j1++)//para todo
{
    for (int a = 1; a < tamanho(A[j1]); a++)//para todo
    {
        IloExpr soma(env);
        for (int t = 0; t < T; t++)
        {
            soma += (tempo[t] + p1[j1])*x[j1][t];
        }
        //declarar a restrição
        IloRange rest_seis(env, 0, rd[a]-soma, IloInfinity);
        //nomeando a restrição
        stringstream rest;
        rest << "seis: ";
        rest_seis.setName(rest.str().c_str());
        //adicionar ao modelo
        modelo.add(rest_seis);
    }
}

```

Figura 25 – Sexta restrição

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

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

```

for (int j2 = 0; j2 < J2; j2++)//para todo
{
    for (int b = 1; b < tamanho(B[j2]); b++)//para todo
    {
        IloExpr soma(env);
        for (int t = 0; t < (ht-p2[j2]); t++)
        {
            soma += tempo[t]*y[j2][t];
        }
        //declarar a restrição
        IloRange rest_sete(env, 0, soma-rd[b], IloInfinity);
        //nomeando a restrição
        stringstream rest;
        rest << "sete: ";
        rest_sete.setName(rest.str().c_str());
        //adicionar ao modelo
        modelo.add(rest_sete);
    }
}

```

Figura 26 – Sétima restrição

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

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:

```
for (int j2 = 0; j2 < J2; j2++)//para todo
{
    for (int b = 1; b < tamanho(B[j2]); b++)//para todo
    {
        IloExpr soma(env);
        for (int t = 0; t < (ht - p2[j2]); t++)
        {
            soma += tempo[t]*y[j2][t];
        }
        soma += p2[j2];
        //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 27 – Oitava restrição

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

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:

```
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);
}
```

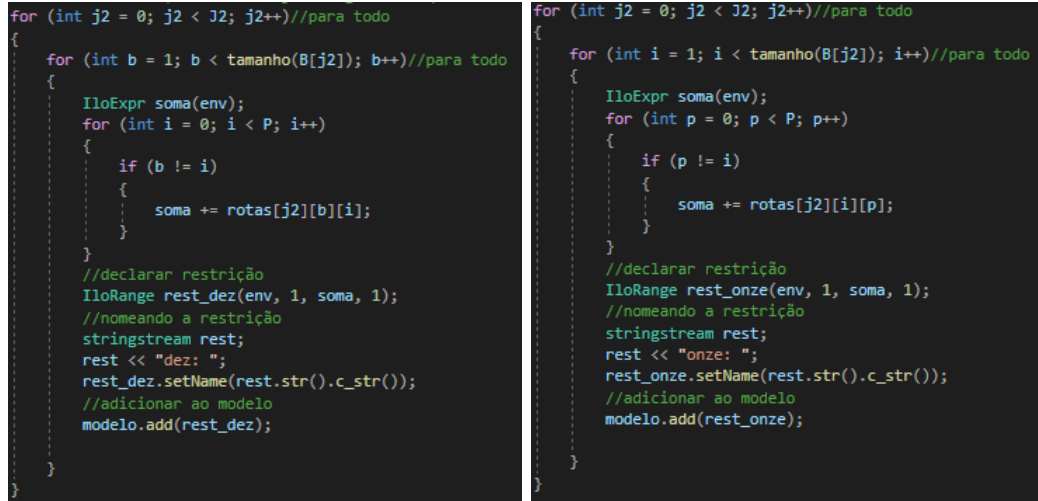
Figura 28 – Nona restrição



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

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

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



```

for (int j2 = 0; j2 < J2; j2++)//para todo
{
    for (int b = 1; b < tamanho(B[j2]); b++)//para todo
    {
        IloExpr soma(env);
        for (int i = 0; i < P; i++)
        {
            if (b != i)
            {
                soma += rotas[j2][b][i];
            }
        }
        //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);
    }
}

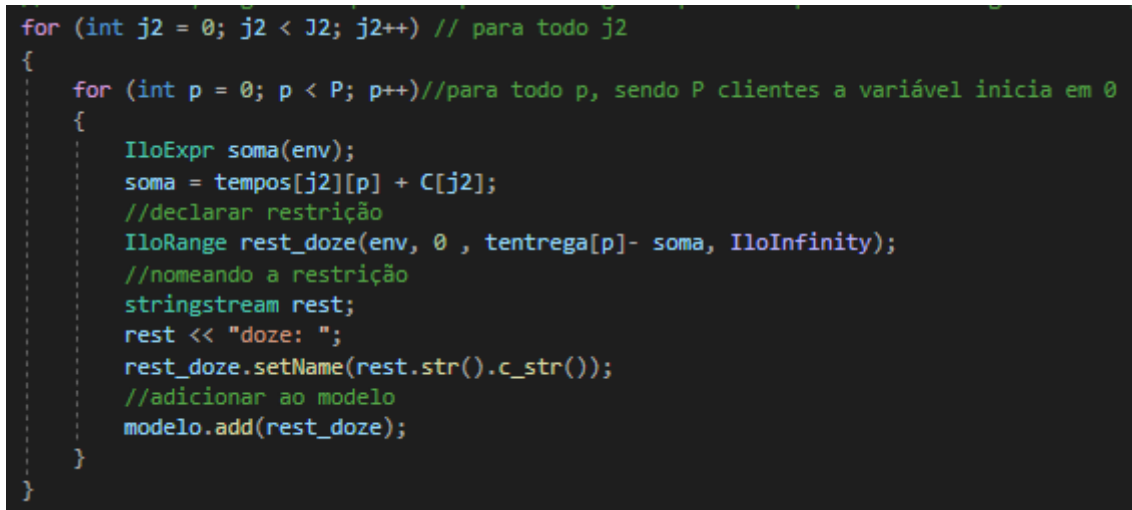
for (int j2 = 0; j2 < J2; j2++)//para todo
{
    for (int i = 1; i < tamanho(B[j2]); i++)//para todo
    {
        IloExpr soma(env);
        for (int p = 0; p < P; p++)
        {
            if (p != i)
            {
                soma += rotas[j2][i][p];
            }
        }
        //declarar 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 29 – Décima e décima primeira restrições

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

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 ( $C_j$ ) somado ao tempo de viagem até o ponto de entrega. Declaração:



```

for (int j2 = 0; j2 < J2; j2++) // para todo j2
{
    for (int p = 0; p < P; p++)//para todo p, sendo P clientes a variável inicia em 0
    {
        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 30 – Décima segunda restrição

$$(13) \text{atraso}_p \geq \text{tentrega}_p - d_p, \forall p \in P \quad (4.14)$$

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:

```
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 31 – Décima terceira restrição

$$(14) \text{tempos}_{jn} \geq \text{tempos}_{ji} + dc_{in} - M * (1 - \text{rotas}_{jin}), \forall n, i \in P \wedge n \neq 0, \forall j \in J_2 \quad (4.15)$$

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:

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

Figura 32 – 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

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

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

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

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

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

Variáveis binárias

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

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

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

## 5 Rodando o programa

### 5.1 Tentativa com todas as restrições

Após a implementação nos deparamos com problemas nas restrições, logo na primeira ela se apresentou como inviável, após diversas tentativas de solução não conseguimos uma forma de tratar o problema como viável e assumimos uma dificuldade de interpretação para o funcionamento do modelo utilizado. Seguem os resultados de execução do programa:

```

Numero de tempos: 5370
Numero de itens em J1: 3
Numero de itens em J2: 3
Numero de produtos: 11
Caminhoes do primeiro estagio:
CaminhaoEntrada1      345      0

Produtos carregados no caminhao CaminhaoEntrada1
Smartphone      20
Smartwatch      10
Capinha         100
Controle        25
VideoGame       50
Televisao       30
Headset         30
Notebook        12
MouseGamer      20
Teclado         40

CaminhaoEntrada2      553      1

Produtos carregados no caminhao CaminhaoEntrada2
Smartphone      8
Smartwatch      15
Capinha         300
Controle        20
VideoGame       50
Televisao       35
Headset         40
Notebook        13
MouseGamer      60
Teclado         30

CaminhaoEntrada3      322      2

Produtos carregados no caminhao CaminhaoEntrada3
Smartphone      2
Smartwatch      15
Capinha         100
Controle        5
VideoGame       50
Televisao       35
Headset         30
Notebook        10
MouseGamer      70
Teclado         80

```

Figura 33 – Execução do programa parte 1.

```

CaminhaoSaida1  660

Produtos carregados no caminhao CaminhaoSaida1
Smartphone      10
Smartwatch      15
Capinha         100
Controle        25
VideoGame       40
Televisao       30
Headset         40
Notebook        10
MouseGamer      30
Teclado         20

CaminhaoSaida2  910

Produtos carregados no caminhao CaminhaoSaida2
Smartphone      10
Smartwatch      15
Capinha         200
Controle        20
VideoGame       60
Televisao       30
Headset         30
Notebook        10
MouseGamer      60
Teclado         80

CaminhaoSaida3  870

Produtos carregados no caminhao CaminhaoSaida3
Smartphone      10
Smartwatch      10
Capinha         200
Controle        5
VideoGame       50
Televisao       40
Headset         30
Notebook        15
MouseGamer      60
Teclado         50

Produtos:
Smartphone      1      2      10      0.20      10      30
Smartwatch      1      2      10      0.10      5       40
Capinha         1      2      20      0.05      1      500
Controle        1      2      20      0.20      2       50
VideoGame       2      4      10      5.00      5      150
Televisao       2      4      10      20.00     6      100
Headset         1      2      20      0.40      5      100
Notebook        2      4      10      6.00     10      35
MouseGamer      1      4      5       0.10      4      150
Teclado         1      4      5       1.50      4      150

```

Figura 34 – Execução do programa parte 2.

```

Matriz dos tempos de viagem do cliente i ao j:
0 1 2 3 4 5 6 7 8 9 10
1 0 1 2 3 4 5 6 7 8 9
2 1 0 1 2 3 4 5 6 7 8
3 2 1 0 1 2 3 4 5 6 7
4 3 2 1 0 1 2 3 4 5 6
5 4 3 2 1 0 1 2 3 4 5
6 5 4 3 2 1 0 1 2 3 4
7 6 5 4 3 2 1 0 1 2 3
8 7 6 5 4 3 2 1 0 1 2
9 8 7 6 5 4 3 2 1 0 1
10 9 8 7 6 5 4 3 2 1 0
Horizonte de tempo: 5370
Numero de Caminhos do primeiro estagio: 3
Numero de Caminhos do segundo estagio: 3
Numero de docas do primeiro estagio: 3
Numero de docas do segundo estagio: 3
Capacidade dos caminheiros de saida: 500
Numero suficientemente grande para comparacoes: 30118247717077060
Warning: Output names have been modified due to duplicate names or characters invalid in LP format.
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
Row 'um': infeasible, all entries at implied bounds.
Presolve time = 1.00 sec. (816.62 ticks)

Root node processing (before b&c):
  Real time = 1.51 sec. (1398.46 ticks)
Parallel b&c, 12 threads:
  Real time = 0.00 sec. (0.00 ticks)
  Sync time (average) = 0.00 sec.
  Wait time (average) = 0.00 sec.
-----
Total (root+branch&cut) = 1.51 sec. (1398.46 ticks)
Infeasible

```

Figura 35 – Execução do programa parte 3.

## 5.2 Excluindo as restrições Um e Dois

Após diversos testes não conseguimos solucionar o problema de inviabilidade das restrições um e dois visto que estas tomam para si a variável  $X$  para tratar as duas restrições e se tornam incompatíveis, assim não foi possível chegar ao ótimo. Porém ao remover tais restrições, conseguimos um possível ótimo, porém este não consegue ser impresso devido a dependência dessas restrições para determinar o valor de todas as restrições. Como os dados de entrada são os mesmos, apresentaremos a única diferença que é na saída da solução:

```

Warning: Output names have been modified due to duplicate names or characters invalid in LP format.
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
Tried aggregator 1 time.
MIP Presolve eliminated 10868 rows and 32611 columns.
MIP Presolve modified 108 coefficients.
Reduced MIP has 300 rows, 40 columns, and 600 nonzeros.
Reduced MIP has 0 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.83 sec. (994.84 ticks)
Tried aggregator 1 time.
Detecting symmetries...
Reduced MIP has 300 rows, 40 columns, and 600 nonzeros.
Reduced MIP has 0 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.33 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 12 threads.
Root relaxation solution time = 0.00 sec. (0.20 ticks)

  Nodes
  Node Left   Objective  IInf Best Integer   Cuts/
                                Best Bound   ItCnt   Gap
*    0    0   integral    0   46812.0000   46812.0000    0   0.00%
Elapsed time = 1.06 sec. (1280.26 ticks, tree = 0.00 MB, solutions = 1)

Root node processing (before b&c):
  Real time = 1.06 sec. (1281.13 ticks)
Parallel b&c, 12 threads:
  Real time = 0.00 sec. (0.00 ticks)
  Sync time (average) = 0.00 sec.
  Wait time (average) = 0.00 sec.
-----
Total (root+branch&cut) = 1.06 sec. (1281.13 ticks)
Optimal

```

Figura 36 – Execução do programa modificado.

## 6 Finalizando a implementação

### 6.1 Impressão dos dados gerados

Apesar de nosso problema não ter sido viável por algumas dificuldades com o 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 começa 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 começa 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 37 – 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-com-roteamento-de-veicul#/>](https://proceedings.science/sbpo-series/sbpo-2021/papers/pesquisa-operacional-aplicada-a-gestao-logistica--desenvolvimento-de-um-modelo-de-cross-docking-com-roteamento-de-veicul#/)