U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Trabalho prático I

Logística Urbana para Entrega de Mercadorias

Trabalho realizado por:

- Adam Nogueira – 202007519 – up202007519@fe.up.pt
- Ana Sofia Costa – 202007602 – up202007602@fe.up.pt
- José Artur Assunção – 202000163 – up202000163@fe.up.pt

Descrição do problema

No contexto do problema, uma empresa pretende criar uma plataforma de entregas de mercadorias em zonas urbanas.
Aos pedidos de transporte estão associados o id, o peso e o volume do pacote a transportar e a recompensa pela realização do serviço.
Em entregas normais, a empresa recorre à subcontratação de estafetas com viaturas próprias para o transporte das mercadorias. Cada estafeta terá definido a sua capacidade de transporte (volume máximo e peso máximo) que pode transportar e o custo do serviço.

Em relação às entregas expresso, a empresa utiliza uma viatura capaz de transportar apenas um pedido de cada vez, independemente do seu peso e/ou volume, estando unicamente associado um tempo de transporte estimado a este serviço.

Cenário I - otimização do número de estafetas

- Formalização do problema

Objetivo principal : distribuir os pedidos de entregas normais acumulados num dia, de modo a que todos ou grande parte deles sejam realizados, minimizando o número de estafetas usados.

Contexto : Cada estafeta realiza apenas uma viagem por dia, logo é necessário maximizar o número de pedidos a transportar por um estafeta.

Variáveis de decisão :

- Conjunto de estafetas registados : E
  - capacidade de volume de cada estafeta : v_e
  - capacidade de peso de cada estafeta : w_e
- Conjunto de pedidos a entregar : P
  - volume de cada pedido : v_p
  - peso de cada pedido : w_p

Restrições :

- O somatório do volume de todos os pedidos realizados por um estafeta deve ser menor ou igual à sua capacidade;
- O somatório do peso de todos os pedidos realizados por um estafeta deve ser menor ou igual à sua capacidade;
- O número de pedidos de entregas normais realizadas toma valores inteiros não negativos.

Cenário I - otimização do número de estafetas

- Descrição de algoritmos relevantes

De modo a encontrar uma solução que respondesse ao problema do cenário 1, foi implementado um algoritmo ganancioso. Isto é, um algoritmo que tenta realizar uma escolha ótima local em todo e cada estágio da solução.

Este tipo de algoritmo tem como bases :

1. - Escolher o melhor objeto que se pode obter no exato momento, sem considerar as consequências futuras para o resultado final;

2. - Por se ter escolhido um ótimo local a cada passo, espera-se encontrar um resultado ótimo global.

No contexto do cenário 1 :

Na tentativa de minimizar o número de estafetas usados, foi escolhido um algoritmo que ao longo do seu processo vai otimizando as suas escolhas. Para tal, serão escolhidos desde início, para realizar as entrega, os estafetas com as maiores capacidades.

Cenário I - otimização do número de estafetas

- Análise de complexidade

```
vector<Truck> Controller::scenery1(){
    chrono::steady_clock sc;
    auto start = sc.now();
    //SORT ORDERS
    auto cmpRak = [this]  (Order order1, Order order2){return (order1.getRankingWei()+order1.getRankingVol())/2.0<(order2.getRankingWei()+order2.getRankingVol())/2.0 ;};
    vector<Order> orderdb = orderDB;
    SetOrderByWeight( & orderdb);
    SetOrderByVolume( & orderdb);
    sort(orderdb.begin(),orderdb.end(),cmpRak);
    //*SORT TRUCKS
    auto cmpTruckRak = [this]  (Truck truck1, Truck truck2){return (truck1.getRankingWei()+truck1.getRankingVol())/2.0<(truck2.getRankingWei()+truck2.getRankingVol())/2.0 ;};
    vector<Truck> truckByRanking = truckDB;
    SetTruckRankingWeight( & truckByRanking);
    auto vec3 = truckByRanking;
    SetTruckRankingVol( & truckByRanking);
    auto vec2 = truckByRanking;
    sort(truckByRanking.begin(),truckByRanking.end(),cmpTruckRak);

    int n = orderdb.size();
    int j = 0;
    set<int> a;
    set<int> trucksIDS;
    vector<Truck> trucksUsed;
    for (int i = 0; i < n; i++) {
        if(orderdb[i].getWeight()>vec3[0].getWeightMax() || orderdb[i].getVol()> vec2[0].getVolMax()){
            continue;
        }
        if (orderdb[i].getWeight() > truckByRanking[j].getWeightMax() ||orderdb[i].getVol() > truckByRanking[j].getVolMax()  ) {
            j=(j!=(truckByRanking.size()-1)) ? ++j:(truckByRanking.size()-1);
            if(j != (truckByRanking.size() - 1) ){
                j++;
                i--;
            }
            else {
```

```
                j = (truckByRanking.size()-1);
            }
        }
        else {
            a.insert(j);
            trucksIDS.insert(truckByRanking[j].getId());
            truckByRanking[j].setWeightMax( weightMax: truckByRanking[j].getWeightMax() - orderdb[i].getWeight());
            truckByRanking[j].setVolMax( volMax: truckByRanking[j].getVolMax() - orderdb[i].getVol());
            truckByRanking[j].addOrder( & orderdb[i]);
            j = 0;
        }
    }
    for(auto x:a){
        trucksUsed.push_back(truckByRanking[x]);
    }

    auto end = sc.now();

    auto time_span = static_cast<chrono::duration<double>>(end - start);
    cout<<"Operation took: "<<time_span.count()<<" seconds !!!\n";

    return trucksUsed;
}
```

Para resolver o problema do cenário 1, foi criada uma função com diferentes etapas para garantir a minimização do número de estafetas usados. Primeiramente, quer para os pedidos de entregas, quer para os estafetas foram realizados processos de ordenação que dependiam da média entre o seu volume e o seu peso. Para tal, foi utilizada a função template sort da biblioteca algorithm.

Cenário I - otimização do número de estafetas

- Análise de complexidade

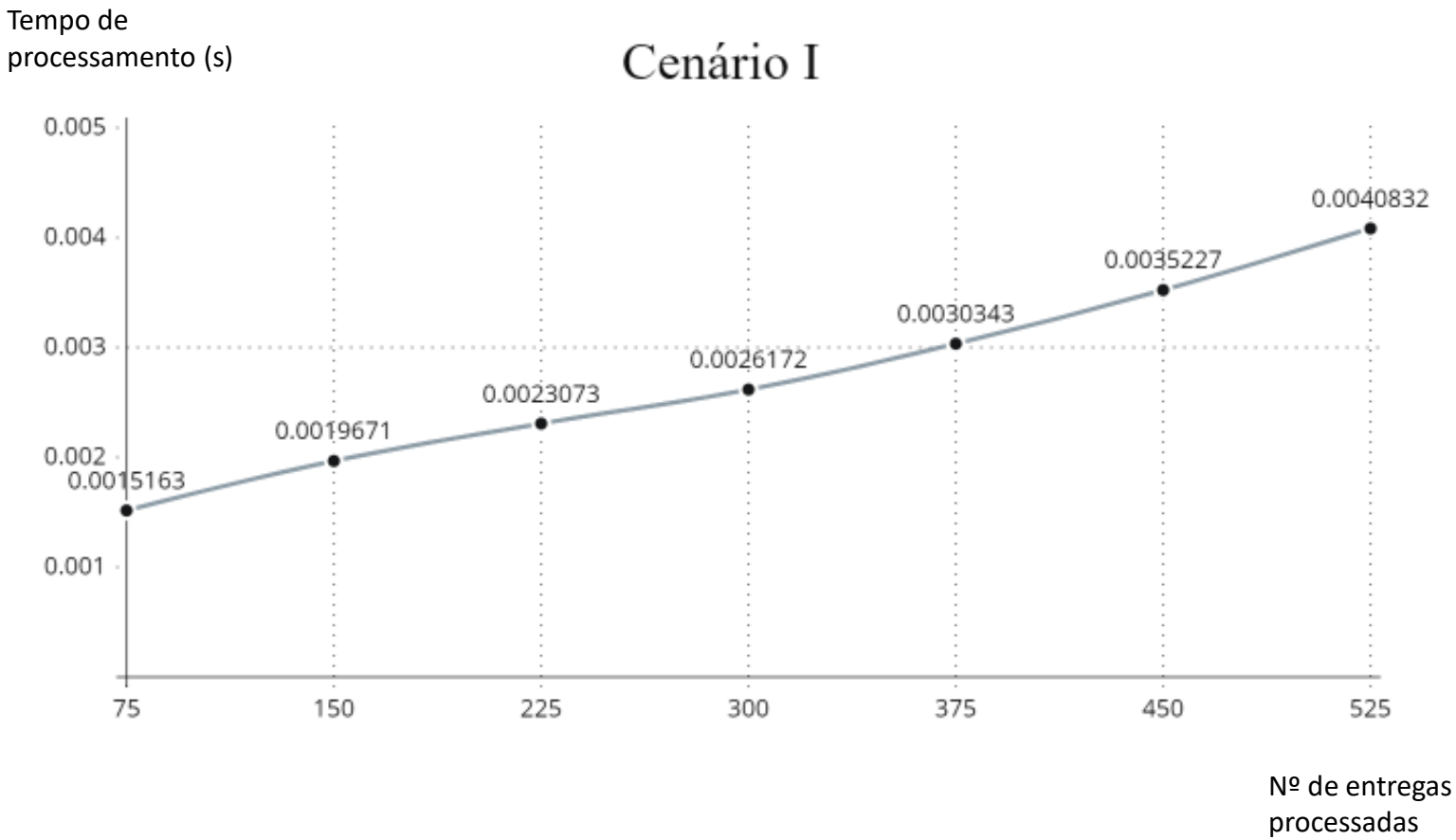
```
sort(truckByRanking.begin(),truckByRanking.end(),cmpTruckRak);
int n = orderdb.size();
int j = 0;
set<int> ids;
set<int> trucksIDS;
vector<Truck> trucksUsed;
for (int i = 0; i < n; i++) {
    if(orderdb[i].getWeight()>vec3[0].getWeightMax() || orderdb[i].getVol()> vec2[0].getVolMax()){
        continue;
    }
    if (orderdb[i].getWeight() > truckByRanking[j].getWeightMax() ||orderdb[i].getVol() > truckByRanking[j].getVolMax()  ) {
        j=(j!=(truckByRanking.size()-1)) ? ++j:(truckByRanking.size()-1);
        if(j != (truckByRanking.size() - 1) ){
            j++;
            i--;
        }
        else {
            j = (truckByRanking.size()-1);
        }
    }
    else {
        ids.insert(j);
        trucksIDS.insert(truckByRanking[j].getId());
        truckByRanking[j].setWeightMax( weightMax: truckByRanking[j].getWeightMax() - orderdb[i].getWeight());
        truckByRanking[j].setVolMax( volMax: truckByRanking[j].getVolMax() - orderdb[i].getVol());
        truckByRanking[j].addOrder( &: orderdb[i]);
        j = 0;
    }
}
for(auto x:ids){
    trucksUsed.push_back(truckByRanking[x]);
}

auto end = sc.now();
```

Por fim, através de um ciclo (com n iterações, n = nº de pedidos de entrega) será feita a planificação dos estafetas e das respetivas entregas que irão realizar.
Concluindo, o algoritmo desenvolvido passará a ter a complexidade da função sort, sendo esta, O (N*log2(N)) em que N é igual ao tamanho do vetor de todos os pedidos registados no sistema.

Cenário I - otimização do número de estafetas

- Resultados da avaliação empírica



Cenário II - otimização do lucro da empresa

- Formalização do problema

Objetivo principal : Maximizar o lucro diário da empresa na realização de pedidos de entregas normais.

Contexto : O lucro da empresa corresponde à diferença entre a receita total dos pedidos entregues e a despesa correspondente ao custo total a ser pago aos estafetas utilizados.

Variáveis de decisão :

- Conjunto de estafetas registados : E
  - capacidade de volume de cada estafeta : v_e
  - capacidade de peso de cada estafeta : w_e
  - custo de cada estafeta : c_e
- Conjunto de pedidos a entregar : P
  - volume de cada pedido : v_p
  - peso de cada pedido : w_p
  - recompensa de cada pedido : r_p

Restrições :

- O somatório do volume de todos os pedidos realizados por um estafeta deve ser menor ou igual à sua capacidade;
- O somatório do peso de todos os pedidos realizados por um estafeta deve ser menor ou igual à sua capacidade;
- O número de pedidos de entregas normais realizadas, o lucro total e individual de cada estafeta tomam valores inteiros não negativos.

Cenário II - otimização do lucro da empresa

- Descrição de algoritmos relevantes

De modo a encontrar uma solução que respondesse ao problema do cenário 2, foi implementado um algoritmo de programação dinâmica. Isto é, um algoritmo que divide o problema em subproblemas e resolve-os uma vez. Com recurso ao método de tabulation, são guardadas informações úteis para que não sejam feitas repetições de cálculos.

Este tipo de algoritmo tem como bases :

1. - Ao chegar a um ponto de escolha o problema subdivide-se em diferentes partes, sendo resolvidas uma parte de cada vez;

2. - Chegando a um "beco sem saída", retroceder até ao ponto de escolha mais próximo com subproblemas a resolver, e processar os mesmos;

3. – Ao resolver um subproblema guardar a solução para evitar repetições de cálculos;

4. - Retornar o subproblema com a melhor solução.

No contexto do cenário 2 :

Através de um algoritmo baseado nos métodos programação dinâmica e tabulation, serão calculados todos os lucros totais possiveís para cada estafeta individual, para que seja entregue a combinação cujo lucro da empresa com as encomendas seja máximizado.

Cenário II - otimização do lucro da empresa

- Análise de complexidade

```
vector<Truck> Controller::scenery2(int& getProfit){
    chrono::steady_clock sc;
    auto start = sc.now();
    sortTruckDBforS2(); //sort based on volume*weight/cost
    int total = 0; //profit counter
    vector<Truck> usedTrucks; //used trucks, returned for interface processing
    int i = 0; //current truck index

    while(!orderDB.empty() && i<truckDB.size()) {
        int reward = processTruck( & truckDB[i]);
        if (reward == 0) { //no reward: skip truck
            i++;
            continue;
        }
        int cost = truckDB[i].getCost();
        int profit = reward - cost;
        if (profit <= 0) { //negative profit: return all orders and go to next truck
            for( auto order : truckDB[i].getOrdersInside() )
                orderDB.push_back(order);
                truckDB[i].emptyTruck();
                continue;
        }
        total += profit;
        usedTrucks.push_back(truckDB[i]);
        i++;
    }
    getProfit = total; // setting given int as Total profit to be shown to user
    auto end = sc.now();
    auto time_span = static_cast<chrono::duration<double>>(end - start);
    cout<<"Operation took: "<<time_span.count()<<" seconds !!!\n";
    return usedTrucks;
}
```

Para resolver o segundo cenário, começamos por ordenar os estafetas baseado no seu custo benefício, isto é, dando prioridade a estafetas de maior capacidade e menor custo (com recurso à função sort). Esta parte do algoritmo é gananciosa, uma vez que seleciona os estafetas individualmente e os processa um por um. O processamento de cada estafeta é realizado pela função processTruck(), que baseado nas encomendas disponíveis encontra o lucro máximo para o estafeta atual.

Cenário II - otimização do lucro da empresa

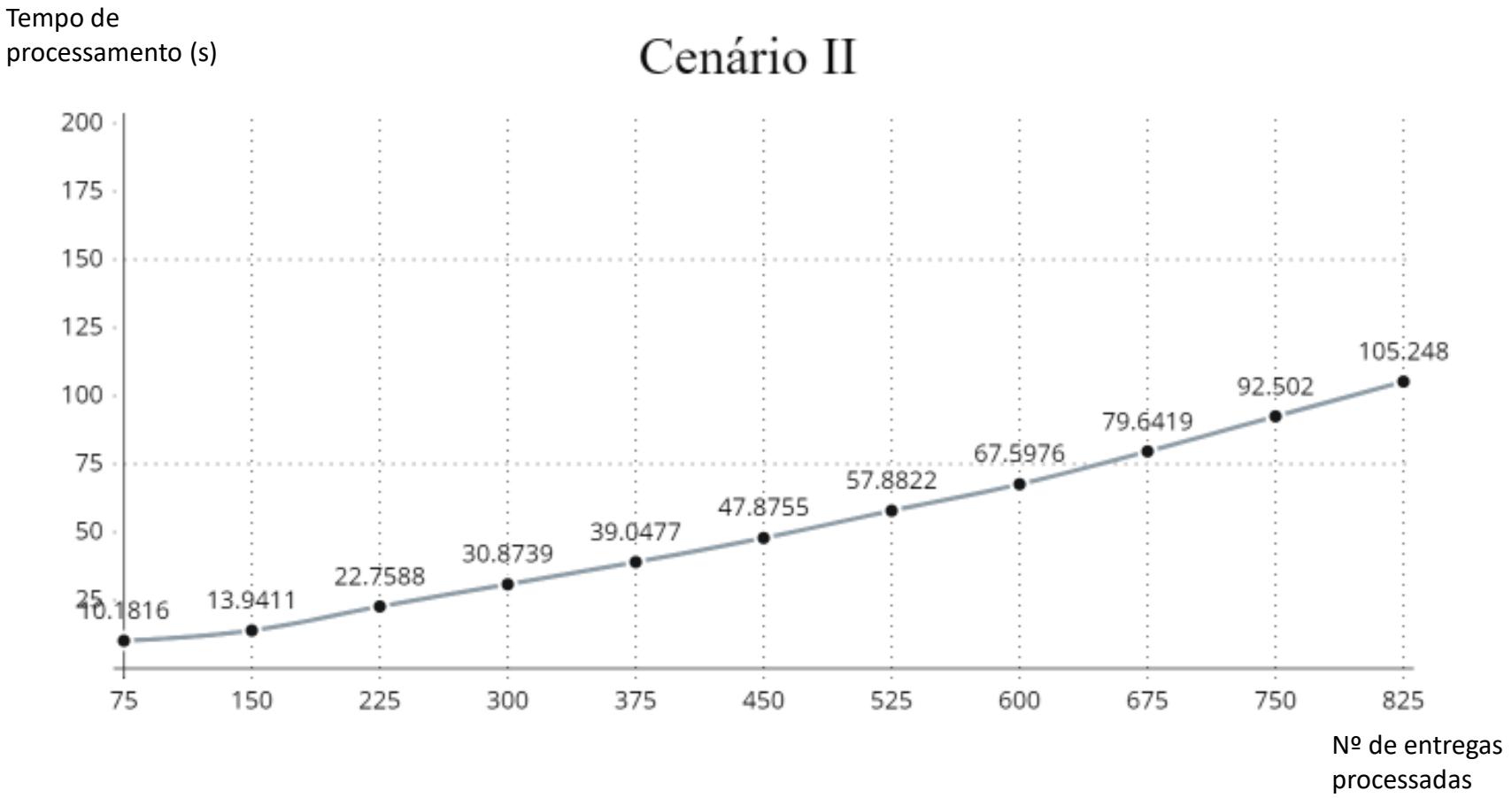
- Análise de complexidade

```
int Controller::processTruck(Truck &truck){
    if (truck.getVolMax() <= 0 || truck.getWeightMax()<=0) {
        return 0;
    }
    int n = orderDB.size();
    vector<vector<vector<int>>>dp(n, value: vector<vector<int>>( n: truck.getWeightMax() + 1, value: vector<int>( n: truck.getVolMax()+1)));
    for (int i = 0; i < n; i++) {
        dp[i][0][0] = 0;
    }
    for (int c = 0; c <= truck.getWeightMax(); c++) {
        for(int j= 0; j<= truck.getVolMax();j++){
            if (orderDB[0].getWeight() <= c && orderDB[0].getVol() <= j) {
                dp[0][c][j] = orderDB[0].getReward();
            }
        }
    }
    for (int i = 1; i < n; i++) {
        for (int c = 1; c <= truck.getWeightMax() ; c++) {
            for(int j = 1; j <= truck.getVolMax(); j++){
                int profit1 = 0, profit2 = 0;
                if (orderDB[i].getWeight() <= c && orderDB[i].getVol()<=j) {
                    profit1 = orderDB[i].getReward() + dp[i - 1][c - orderDB[i].getWeight()][j - orderDB[i].getVol()];
                }
                profit2 = dp[i - 1][c][j];
                dp[i][c][j] = max(profit1, profit2);
            }
        }
    }
    int totalProfit = dp[orderDB.size() - 1][truck.getWeightMax()][truck.getVolMax()];
    processSelectedOrders( &: dp, &: truck);
    return totalProfit;
}
```

Esta função, tem como base uma abordagem Bottom Up, para a qual foi criada uma tabela de 3 dimensões onde guardamos o maior lucro para um determinado volume e peso.
O algoritmo, então, subdivide os problemas até às suas menores partes, solucionando-as e guardando os seus resultados na tabela. Por fim é chamada a função processSelectedOrders(), que percorre a tabela e instancia o melhor resultado para aquela estafeta.
Sendo assim, o algoritmo terá uma complexidade temporal de Nlog(N) * P * V * E, onde N é o número de encomendas, P é o peso máximo de um estafeta, V o volume máximo de um estafeta e E o número de estafetas.

Cenário II - otimização do lucro da empresa

- Resultados da avaliação empírica



Cenário III - otimização das entregas expresso

- Formalização do problema

Objetivo principal : Máximar o número de pedidos de entregas expresso e minimizar o tempo médio dessas entregas num dia.

Contexto : As entregas são realizadas uma de cada vez por uma única viatura, independentemente do seu volume ou peso, durante o horário comercial, das 9:00 às 17:00.

Variáveis de decisão :

- Conjunto de pedidos entregues : P
  - tempo estimado de cada pedido : t_p (s)

Restrições :

- O somatório do tempo estimado de todos os pedidos entregues tem de ser menor ou igual a 28800(s) (segundos entre as 9:00h e as 17:00h).
- O número de pedidos de entregas expresso realizadas toma valores inteiros não negativos.

Cenário III - otimização das entregas expresso

- Descrição de algoritmos relevantes

De modo a encontrar uma solução que respondesse ao problema do cenário 3, foi implementado um algoritmo ganancioso. Isto é, um algoritmo que tenta realizar uma escolha ótima local em todo e cada estágio da solução.

Este tipo de algoritmo tem como bases :

1. - Escolher o melhor objeto que se pode obter no exato momento, sem considerar as consequências futuras para o resultado final;
2. - Por se ter escolhido um ótimo local a cada passo, espera-se encontrar um resultado ótimo global.

No contexto do cenário 3 :

Na tentativa de máximar o número de pedidos de entregas expresso e minimizar o tempo médio dessas entregas num dia, a cada iteração deve ser sempre escolhida o pedido de tempo de entrega mais curto.

Cenário III - otimização das entregas expresso

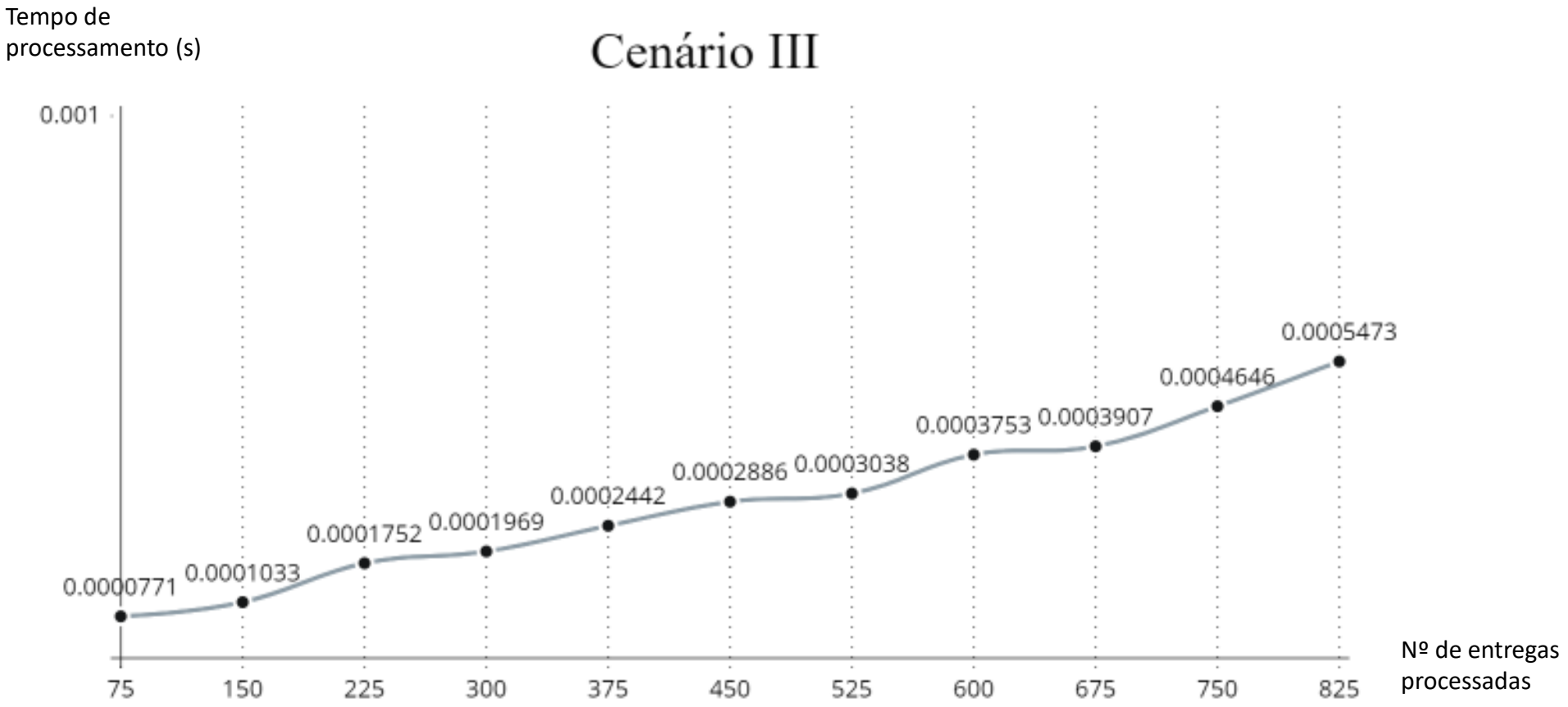
- Análise de complexidade

Para que a cada iteração do ciclo fosse escolhido o pedido com o período mais curto, foi utilizada a função template sort da biblioteca algorithm. Consequentemente, o algoritmo desenvolvido passará a ter a complexidade do sort, sendo esta, O (N*log2(N)) em que N é igual ao tamanho do vetor aux (vector que contém todos os pedidos registados no sistema).

```
vector<Order> Controller::scenery3(){
    vector<Order> res;
    vector<Order> aux = orderDB;
    sort(aux.begin(),aux.end(), comp: [](const Order &a, const Order &b) {
        return a.getDuration() < b.getDuration();});
    int total=0;
    int time=28800;
    for(int i=0; i<aux.size(); i++){
        if(total + aux[i].getDuration() >time){break;}
        total+=aux[i].getDuration();
        res.push_back(aux[i]);
    }
    return res;
}
```

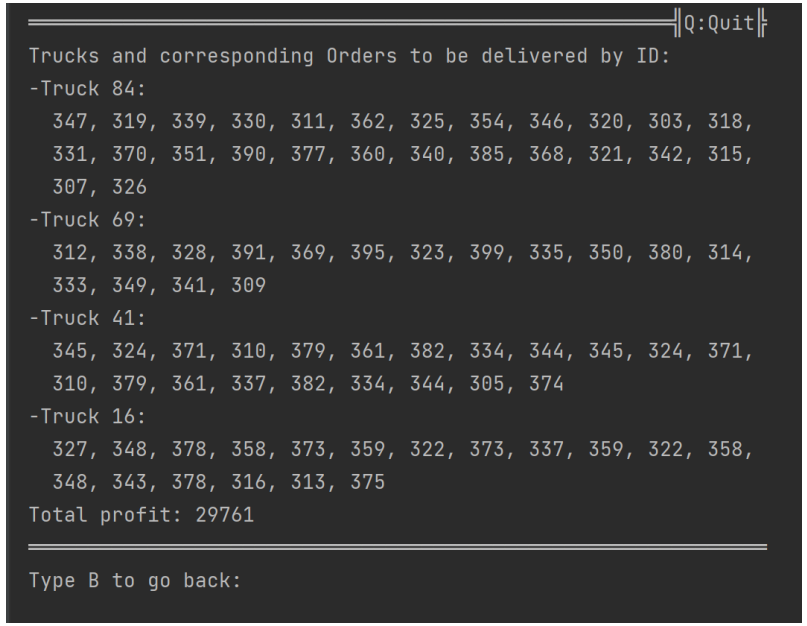
Cenário III - otimização das entregas expresso

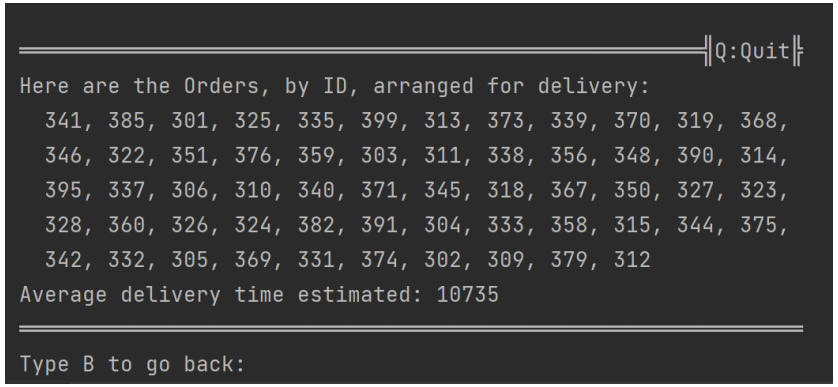
- Resultados da avaliação empírica



Funcionalidades extra

Lucro da empresa : Sendo o objetivo do cenário II aumentar o lucro da empresa, foi acrescentado no código a funcionalidade que permite ao utilizador conhecer o lucro da empresa, tendo em conta os estafetas e os pedidos de entrega que foram registados no sistema.





Tempo médio : Sendo o objetivo do cenário III minimizar o tempo médio das entregas, foi acrescentado no código a funcionalidade que permite ao utilizador conhecer o tempo médio de uma entrega, tendo em conta os estafetas e os pedidos de entrega que foram registados no sistema.

Destaque de algoritmo

- Cenário II :

```
int Controller::processTruck(Truck &truck){
    if (truck.getVolMax() <= 0 || truck.getWeightMax()<=0) {
        return 0;
    }
    int n = orderDB.size();
    vector<vector<vector<int>>>dp(n, value: vector<vector<int>>( n: truck.getWeightMax() + 1, value: vector<int>( n: truck.getVolMax()+1)));
    for (int i = 0; i < n; i++) {
        dp[i][0][0] = 0;
    }
    for (int c = 0; c <= truck.getWeightMax(); c++) {
        for(int j= 0; j<= truck.getVolMax();j++){
            if (orderDB[0].getWeight() <= c && orderDB[0].getVol() <= j) {
                dp[0][c][j] = orderDB[0].getReward();
            }
        }
    }
    for (int i = 1; i < n; i++) {
        for (int c = 1; c <= truck.getWeightMax() ; c++) {
            for(int j = 1; j <= truck.getVolMax(); j++){
                int profit1 = 0, profit2 = 0;
                if (orderDB[i].getWeight() <= c && orderDB[i].getVol()<=j) {
                    profit1 = orderDB[i].getReward() + dp[i - 1][c - orderDB[i].getWeight()][j - orderDB[i].getVol()];
                }
                profit2 = dp[i - 1][c][j];
                dp[i][c][j] = max(profit1, profit2);
            }
        }
    }
    int totalProfit = dp[orderDB.size() - 1][truck.getWeightMax()][truck.getVolMax()];
    processSelectedOrders( &: dp, &: truck);
    return totalProfit;
}
```

Principais dificuldades

Neste trabalho, a maior dificuldade foi a decisão de qual algoritmo seria mais eficiente para cada cenário.

Divisão de tarefas

Adam Nogueira : Interface e algoritmos;
Ana Sofia Costa : Leitura e Escrita em ficheiros, CRUD e apresentação;
José Artur Assunção : Algoritmos e lógica do sistema.