T1 - Logística Urbana para Entrega de Mercadorias

L.EIC016 - Desenho de Algoritmos

Grupo 106
Ana Silva (up202004380), Tomás Carmo(up202007590) e Tomé Cunha(201904710)

Problema

Este projeto procura abordar o problema de otimização de uma empresa de entregas de mercadoria.

Consiste em criar e implementar uma plataforma de gestão da empresa de logística urbana, com o objetivo de tornar a sua operação o mais eficiente possível.

De forma a :

- Otimizar o número de estafetas a ser utilizados de modo a entregar o máximo número de pedidos num dia.
- Otimizar o lucro da empresa, selecionando os estafetas mais lucrativos. Otimizar as entregas expresso de modo a minimizar o tempo médio de

Cenário 1 - Formalização

Dados de Entrada:

E: conjunto de estafetas com capacidade de volume *ve*, peso we

P: conjunto de encomendas com volume vp, peso *wp*, recompensa *rp*

Variáveis de decisão:

ne: número de estafetas

e: estafeta usado

I: conjunto finito de items

Objetivo:

$$\min \sum_{j=0}^{ne} e_j$$

Restrições:

$$\sum_{i \in I}^{np} vp(i)p_{ij} \leq ve_j, \forall j \in \{0, .., ne\}$$

$$\sum_{i \in I}^{np} wp(i)p_{ij} \leq we_j, \forall j \in \{0, .., ne\}$$

$$e_j \in \{0, 1\}, \forall j \in \{0, .., ne\}$$

$$p_{ij} \in \{0, 1\}, \forall i \in I j \in \{0, .., ne\}$$

Cenário 1 - Descrição de algoritmos relevantes

No cenário 1, escolhemos uma variação algoritmo de bin packing, first-fit decreasing:

- ➔ organiza os vetores de estafetas e encomendas por ordem decrescente;
- ➔ percorre todas as encomendas e por cada encomenda percorre os estafetas com carga, adicionando-a ao primeiro que caiba;
- ➔ se a encomenda não couber em qualquer estafeta já com carga usamos um novo estafeta;

```
int firstFit(vector<Truck> &trucks, vector<Package> &packages){
    int counter = 0;
    int tNum = trucks.size();
    // Try to fit the first bin, if full jump to next
    for(int i = 0; i < packages.size(); i++) {
        int j;
        for (j = 0; j < counter; j++) {
            if (trucks.at(j).getWeightCap() >= packages.at(i).getWeight() &&
                trucks.at(j).getVolumeCap() >= packages.at(i).getVolume()) {
                trucks.at(j).addPackage(packages.at(i));
                break;
            }
        }
        if(j == counter && tNum > 0) {
            trucks.at(counter).addPackage(packages.at(i));
            tNum--;
            counter++;
        }
    }
```

```
int firstFitDecreasing(vector<Truck> &trucks, vector<Package> &packages) {
    // First Fit Decreasing Algorithm
    // Sort the input sequences placing large items first
    sort(packages.begin(), packages.end(), comparePackages);
    sort(trucks.begin(), trucks.end(), compareTrucks);

    return firstFit(trucks,packages);
}
```

Cenário 1 - Análise de Complexidade

Seja “n” o número de estafetas e “m” o número de encomendas:

Algoritmo first-fit-decreasing:

- ➔ Análise temporal: O(n*log(n)+m*log(m)+m*n^2) = O(m*n^2)
- ➔ Análise espacial: O(n+m)

Cenário 1 - Resultados empíricos

Com a nossa implementação do algoritmo first-fit-decreasing, o mínimo número de estafetas possível foi 22. Achamos que é o número mais baixo que se consegue obter.

```
C:\Users\tomas\OneDrive\Documentos\GitHub\Proj-DA\cmake-build-debug\Proj_DA.exe
CENARIO 1:
 - Estafetas Utilizados: 22

Process finished with exit code 0
```

Cenário 2 - Formalização

Dados de entrada:

E: conjunto de estafetas com capacidade de volume ve , peso we , e custo ce ;

P: conjunto de encomendas com volume vp , peso wp , e recompensa rp ;

Variáveis de decisão:

np: número de encomendas

ne: número de estafetas

Objetivo:

maximizar $rp_i p_{ij} - ce_j e_j, \forall i \in \{0, .., np\} \forall j \in \{0, .., ne\}$

Restrições:

$$\sum_{i \in I}^{np} vp(i) p_{ij} \leq ve_j, \forall j \in \{0, .., ne\}$$

$$\sum_{i \in I}^{np} wp(i) p_{ij} \leq we_j, \forall j \in \{0, .., ne\}$$

Cenário 2 - Descrição de algoritmos relevantes

No cenário 2, escolhemos novamente o algoritmo first-fit decreasing:

- ➔ organizamos os vetores de estafetas e encomendas por ordem decrescente de recompensa e custo respetivamente;
- ➔ percorre as encomendas e por cada encomenda percorre os estafetas com carga, adicionando-a ao primeiro que caiba e somando o lucro;
- ➔ se a encomenda não couber em qualquer estafeta já com carga usamos um novo estafeta, subtraindo o custo do mesmo;

```
int maximizeProfit(vector<Package> &packages,  vector<Truck> &trucks){
    sort(packages.begin(), packages.end(), compareReward);
    sort(trucks.begin(), trucks.end(), compareCost);

    int counter = 0; int profit = 0;
    int tNum = trucks.size();
    for(int i = 0; i < packages.size(); i++) {
        int j;
        for (j = 0; j < counter; j++) {
            if (trucks.at(j).getWeightCap() >= packages.at(i).getWeight() &&
                trucks.at(j).getVolumeCap() >= packages.at(i).getVolume()) {
                trucks.at(j).addPackage(packages.at(i));

                profit += packages.at(i).getReward();
                break;
            }
        }
        if(j == counter && tNum > 0) {
            trucks.at(counter).addPackage(packages.at(i));
            profit += packages.at(i).getReward(); profit -= trucks.at(i).getCost();
            tNum--;
            counter++;
        }
    }

    return profit;
}
```

Cenário 2 - Análise das complexidades

Seja “n” o número de estafetas e “m” o número de encomendas:

Algoritmo adaptado de first-fit-decreasing:

- ➔ Análise temporal: O(nlog(n)+mlog(m)+m*n^2) = O(m*n^2)
- ➔ Análise espacial: O(n+m)

Cenário 2 - Resultados empíricos

O nosso algoritmo, adaptado do cenário 1 obteve-nos um lucro de 16136, não tendo muitas comparações com outros algoritmos devido à falta de tempo uma vez que tivemos mais dificuldade na implementação do mesmo.

```
C:\Users\tomas\OneDrive\Documentos\GitHub\Proj-DA\cmake-build-debug\Proj_DA.exe

CENARIO 2:
 - Lucro: 16136 euros

Process finished with exit code 0
```

Cenário 3 - Formalização

Dados de entrada:

P: conjunto de pedidos expresso com volume vp , peso wp , e tempo estimado de entrega tp ;

Variáveis de decisão:

T: total de horas por dia

k: número de entregas somadas

Objetivo:

$$\min \ (\sum_{i=0}^{np} tp_i)/k$$

Restrições:

$$\sum_{i=0}^{np} tp_i \leq T$$

Cenário 3 - Descrição de algoritmos relevantes

No cenário 3 usamos um algoritmo greedy chamado job scheduling que faz o seguinte, adaptado para o nosso problema :

- ➔ organiza as encomendas por duração por ordem crescente.
- ➔ percorre o vetor de encomendas somando todas as durações possíveis sem que ultrapasse o horário de trabalho.

```
int maximizeExpress(vector<Package> &packages){
    sort(packages.begin(), packages.end(), compareDuration);

    int timeleft = 28800;
    int counter = 0;

    for (auto p : packages) {
        timeleft -= p.getDuration();
        counter += 1;


        if (timeleft <= 0) {
            timeleft += p.getDuration();
            counter -= 1;
        }
    }

    //cout << "Encomendas: " << counter << endl;
    return 28800-timeleft;
}
```

Cenário 3 - Análise das complexidades

Seja “n” o número de encomendas:

Algoritmo Job-Scheduling:

- ➔ Análise temporal: O(nlog(n)+n) = O(nlog(n))
- ➔ Análise espacial: O(n)

Cenário 3 - Resultados empíricos

Como resultado, neste cenário, obtivemos que o tempo médio de entrega de uma entrega expresso é 161 segundos.

Este valor foi o mínimo valor que nós, como grupo conseguimos obter.

```
C:\Users\tomas\OneDrive\Documentos\GitHub\Proj-DA\cmake-build-debug\Proj_DA.exe
CENARIO 3:
 - Tempo Medio de Entrega Num Dia: 161 segundos

Process finished with exit code 0
```

Solução algorítmica a destacar

A solução algorítmica que decidimos destacar neste projeto é a do cenário 1, pois é um algoritmo simples mas muito eficaz que nos deu o resultado pretendido.

Definitivamente poupou-nos tempo na implementação comparado com os outros cenários.

Principais dificuldade e esforço de cada elemento do grupo

Como grupo, encontramos a maior dificuldade na implementação do cenário 2:

- Foram analisadas várias formas de implementação, mas encontramos um impasse no que toca à passagem dos dados para função

Quanto ao esforço de cada elemento do grupo o trabalho foi dividido de forma igual e todos cumpriram com a sua obrigação.