Logística Urbana para a Entrega de Mercadorias

Desenho de Algoritmos - Grupo 113

Gustavo Costa - up202004187
Ricardo Cavalheiro - up202005103
Mª Leonor Beirão - up201806798

Descrição do Problema Geral

O problema geral trata-se de um problema de gestão e otimização na área de logística de entregas de mercadorias.

Para este trabalho, 3 cenários foram criados, com o objetivo de resolver de 3 sub-problemas:

-Cenário 1: otimização do número de estafetas,

-Cenário 2: otimização do lucro da empresa,

-Cenário 3: otimização das entregas expresso.

Descrição

Cenário 1 - otimização do número de estafetas

- O objetivo deste cenário é minimizar o número de estafetas utilizados na realização das entregas, maximizando o número de entregas realizadas.
- Este problema é uma variante do bin-packing problem apresentado nas aulas, que se distingue deste por ter restrições extra.
- As restrições extra revelaram-se um desafio bastante grande, deixando-nos com duas opções:
  - Encontrar um algoritmo para o bin-packing problem que funcione com uma dimensão extra para os pacotes (2D bin-packing) assim como *bins* de dimensões variáveis (variable sized bin-packing).
  - Realizar aproximações, possivelmente erróneas, comprometendo a otimalidade da solução, mas obtendo um algoritmo mais simples.
- O nosso grupo optou pela segunda opção, tentando encontrar, empiricamente, a melhor métrica, que permitisse uma solução aceitável para este problema.
- Ao longo da resolução do problema, várias métricas foram testadas, sendo a escolhida, a melhor entre elas.
- O pensamento inerente a algumas destas métricas baseia-se numa tentativa de pontuação das encomendas e carrinhas, baseada nos seus atributos.

```
void sortOrderByHighestWeight();
void sortOrderByHighestVolume();
void sortOrderByHighestDensity();
void sortOrderByHighestWeightVolume();
void sortOrderByHighestWeightAndVolume();
void sortVansByHighestWeight();
void sortVansByHighestVolume();
void sortVansByHighestDensity();
void sortVansByHighestWeightVolume();
void sortVansByHighestWeightAndVolume();
```

Formalização

- Input:
  - carrinhas de estafetas registados: ve (capacidade de volume) e we (capacidade de peso)
  - encomendas a entregar: vp (volume) e wp (peso)
- Output:
  - número total de carrinhas de estafetas utilizados: Te
- Variáveis de decisão:
  - nº de estafetas
- Funções-objetivo:
  - minimizar $\sum(N, i=1)Te$
- Restrições e Domínios de valores:
  - 0 <= ve e 0 <= we, ve e we reais
  - 0 <= vp e 0 <= wp, vp e wp reais
  - Te tem de ser um número natural
  - $\sum ve <= \sum vp$
  - $\sum we <= \sum wp$
- Objetivos:
  - minimizar o número de estafetas a trabalhar num dia
  - maximizar o número de encomendas entregues num dia

Algoritmos Relevantes

Cenário 1 - otimização do número de estafetas

```
void DeliveryService::sortOrderByHighestWeightByVolume() {
    std::sort(orders.begin(), orders.end(), [](Order *a, Order *b) { return ((*a).getWeight() * (*a).getVolume()) > ((*b).getWeight() * (*b).getVolume()); });
}
void DeliveryService::sortVansByHighestWeightByVolume() {
    std::sort(vans.begin(), vans.end(), [](Van *a, Van *b) { return ((*a).getMaxWeight() * (*a).getVolMax()) > ((*b).getMaxWeight() * (*b).getVolMax()); });
}
```

Como foi dito anteriormente, optamos por realizar aproximações, tentando arranjar uma métrica favorável à ordenação dos vetores que contêm as carrinhas e as encomendas.

Começámos por combinar as duas variáveis de restrição (volume e peso) numa só, ordenando os vetores por ordem decrescente. Várias combinações foram testadas e foi utilizadas a que obteve melhor resultado (volume e peso são multiplicados).

```
std::vector<Order *> used_orders;
std::vector<Van *> used_vans;
used_vans.push_back(vans[0]);
int next_van = 1;
```

Foi utilizado o algoritmo de resolução de problemas do tipo “bin-packing” usando a estratégia **first fit** aquando da distribuição das encomendas às carrinhas. Ou seja, depois de ordenados os vetores, foi atribuída cada encomenda à primeira carrinha onde coubesse.

Foram utilizados vetores para permitir a fácil ordenação e adição de objetos.

Análise de complexidade

Complexidade temporal:

- A ordenação dos vetores tem complexidade temporal O(N logN), em que N representa o número total de carrinhas ou de encomendas, e o algoritmo usado (first fit bin-packing) tem complexidade O(Et * Ct), em que Et representa o número de encomendas total e Ct o número de carrinhas usadas, logo, a complexidade temporal da solução encontrada é T(Et, Ct) = O(Et * Ct).

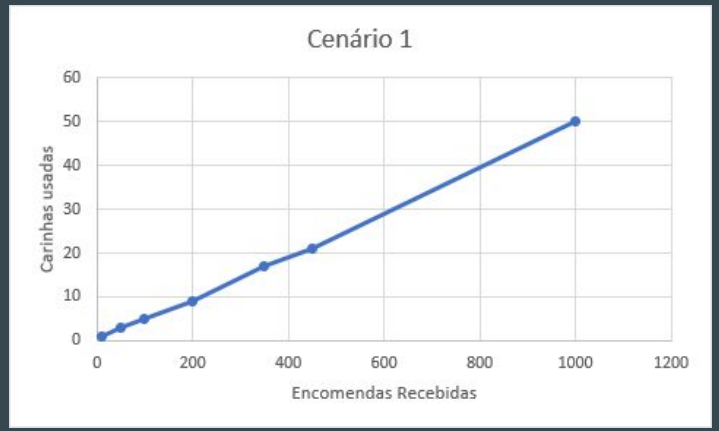
Complexidade espacial:

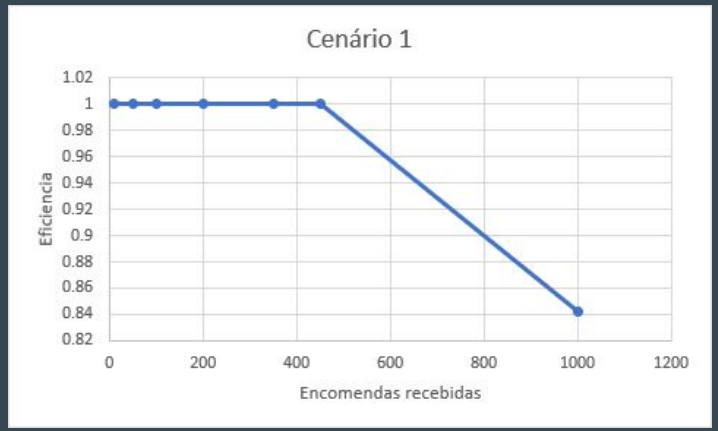
- Em termos espaciais, são utilizados 2 vetores temporários, um de encomendas entregues e outro de carrinhas usadas. No limite, todas as encomendas podem ser entregues, assim como todas as carrinhas podem ser usadas, logo, a complexidade espacial é S(Et, Ct) = O(Et + Ct), onde Ct representa o número máximo de carrinhas disponível e Et o número total de encomendas.

Resultados da Avaliação Empírica

Cenário 1 - otimização do número de estafetas

- Utilizando os ficheiros de dados de carrinhas e encomendas fornecidos como exemplo, conseguimos com que, num dia sejam, utilizadas 21 carrinhas (ou seja, empregar 21 estafetas), entregando todas as encomendas, o que se traduz numa eficiência de 100%.
- Este resultado foi o menor que conseguimos alcançar de entre os algoritmos de ordenação testados, e era consistentemente o melhor com outros dados.





A avaliação empírica revela uma relação, aparentemente, linear entre o número de encomendas e o número de carrinhas usadas. Como o número de carrinhas é fixo, a partir de um certo ponto, a eficiência diminui.

Descrição

- O objetivo deste cenário é maximizar o lucro obtido na realização das entregas de um dia.
- Este problema é uma variante do 0-1 knapsack problem apresentado nas aulas, que se distingue deste por ter restrições extra, como o volume das encomendas, e vários *knapsacks* de dimensões distintas.
- As restrições extra revelaram-se um desafio bastante grande, deixando-nos com duas opções
- Encontrar um algoritmo para o knapsack problem que funcione com uma dimensão extra para os pacotes assim como vários para vários e distintos *knapsacks* não é algo fácil.
- Sabemos que a solução ótima seria um algoritmo de dynamic programing sobre uma matriz tridimensional aplicando memorização.
- Infelizmente, não conseguimos uma versão deste algoritmo que nos satisfizesse, por isso, optamos por uma abordagem mais *naïve* do problema: ordenar as encomendas por recompensa e carrinhas por custo e simplesmente distribuir as encomendas pelas carrinhas.

```
int optimizeProfit(); // Dynamic programing
int optimizeProfit2(); // Naïve
int optimizeProfit3(); // Dynamic programing em matriz tridimensional com memoization
```

Formalização

Cenário 2 - otimização do lucro da empresa

- Input:
  - carrinhas de estafetas registados: ve (capacidade de volume), we (capacidade de peso) e cv (custo da carrinha)
  - encomendas a entregar: vp (volume) e wp (peso) e recompensa (rp)
- Output:
  - lucro da empresa: Te
- Variáveis de decisão:
  - nº de entregas
- Funções-objetivo:
  - maximizar $\sum(N, i=1)$ (rp - cv)
  - ou seja, maximizar Te
- Restrições e Domínios de valores:
  - 0 <= ve, 0 <= we e 0<=cv, ve, we e cv reais
  - 0 <= vp, 0 <= wp e 0<=rp, vp, wp e rp reais
  - Te >= 0, Te real
- Objetivos:
  - maximizar o lucro da empresa
  - maximizar o número de encomendas entregues num dia

Algoritmos relevantes

Cenário 2 - otimização do lucro da empresa

- O algoritmo de recursão que encontramos que solucionasse o problema é baseado em programação dinâmica e apesar de não termos conseguido obter um resultado aceitável, acreditamos que esta estratégia é a mais adequada para obter a melhor solução do problema em questão.
- Optamos também por utilizar uma abordagem semelhante aos restantes cenários em que aplicamos algoritmos simples de ordenação para as carrinhas e as encomendas. Apesar de considerarmos os resultados obtidos bastante mais satisfatórios (lucro: 220 000) no dataset fornecido, não consideramos que seja possível obter uma solução ótima na maior parte dos casos utilizando esta estratégia.

```
int DeliveryService::knapsack(int item, int weight, int volume,int W_max,int Vol_max, int n) {
    // See if we have calculated this item before
    if (dp[item][weight] == -1) {
        // Set initial value to -2 (invalid result)
        int max = -2;
        // Iterate though all items past current item
        for (int i = item; i < n; i++) {
            // Make sure we don't go over max weight
            if (volume + orders.at(i)->getVolume() <= Vol_max && weight + orders.at(i)->getWeight() <= W_max) {
                // Get the result of taking ith item
                int res = knapsack( item: i + 1,  weight: weight + orders.at(i)->getWeight(),
                                    volume: volume + orders.at(i)->getVolume(),W_max, Vol_max, n );
                // Make sure result is valid
                if (res != -2) {
                    // If the result is valid take the max
                    max = std::max(res + orders.at(i)->getReward(), max);
                }
            }
        }

        if (max == -2 && volume <= Vol_max)// No other items taken and over
            // Vol_min
            dp[item][weight] = 0;
        else // Eveything else
            dp[item][weight]= max;
    }
    // Return the value
    return dp[item][weight];
```

Análise de complexidade

Complexidade temporal:

- 1º Método: a complexidade temporal associada é T(v,o) = O(v*2^o)
- 2º Método: a complexidade temporal associada é T(v,o) = O(o log(o) + v log(v))

Complexidade espacial:

- 1º Método: a complexidade espacial associada é S(o) = O(o)
- 2º Método: a complexidade espacial associada é S(v,o) = O(1)

Nota: v - número de carrinhas da empresa;

o - número de encomendas recebidas;

Resultados da Avaliação Empírica

Cenário 2 - otimização do lucro da empresa

A avaliação empírica dos dados leva-nos a acreditar que, à medida que o número de encomendas recebidas aumenta, o lucro também deverá aumentar, mas a eficiência tem tendência a diminuir. É também de relevo que, para números reduzidos de encomendas, pode não ser possível lucrar de todo.

Descrição

- O objetivo deste cenário é minimizar o tempo médio das entregas expresso de um dado dia.
- Este problema é uma variante do *job scheduling* problem apresentado nas aulas, que se distingue deste por ser bastante mais flexível, dado que a entrega de cada encomenda não têm hora de início ou fim definido.
- Dos três cenários, este revelou-se o mais fácil, sendo a abordagem adotada um algoritmo *greedy*, na medida em que as encomendas mais rápidas de ser entregues têm prioridade superior às mais lentas.
- Assim sendo, para minimizar o tempo médio das entregas expresso, basta ordená-las pelo menor tempo de realização e realizá-las sequencialmente, desde o horário de abertura até o horário de fecho.
- Por simplicidade, assumimos que o estafeta responsável não poderia voltar para o armazém depois das 17h, mesmo tendo feito a entrega antes das 17h.

Formalização

- Input:
  - conjunto de pedidos P, com volume vp, peso wp e tempo estimado de entrega tp
- Output:
  - tempo médio de entrega
- Variáveis de decisão:
  - conjunto de entregas P
- Funções-objetivo:
  - minimizar $(\sum(N, i=1)\ tp) / (\sum(N, i=0)\ P)$
- Restrições e Domínios de valores:
  - $0 <= vp$ e $0 <= wp$, $0 <=$ tp, vp, wp e tp reais
  - número de pedidos tem de ser um número natural
  - $\sum(N, i=1)\ tp <= 28800$
- Objetivos:
  - minimizar o tempo médio previsto das entregas expresso a serem realizadas num dia

Algoritmos relevantes

Devido à simplicidade da implementação da nossa solução, que consiste apenas num algoritmo de ordenação e num algoritmo de *scheduling* que verifica se as *boundaries* são respeitadas, não há necessariamente um algoritmo relevante e essencial à resolução deste cenário.

Análise de complexidade

Complexidade temporal:

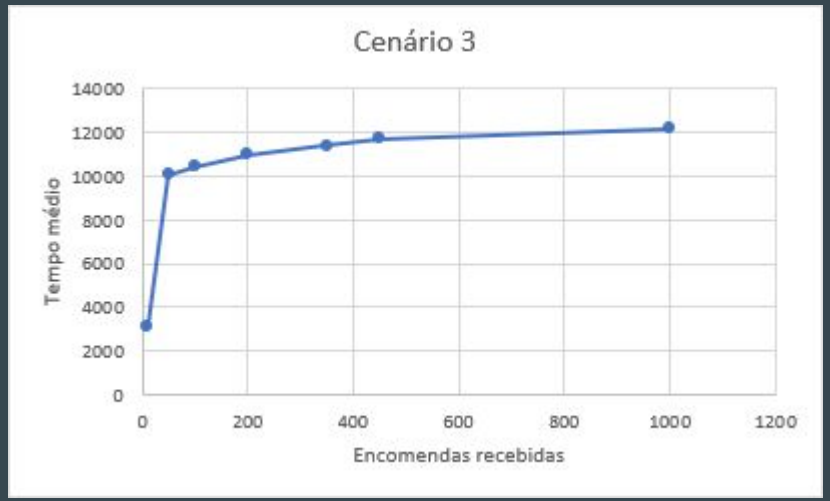
- A solução obtida pode ser dividida em 3 partes, das quais a que tiver maior complexidade temporal, definirá a complexidade temporal do algoritmo.
  1. Ordenação do vetor que contém as encomendas - $T(n) = O(n \log n)$
  2. Iteração pelo vetor das encomendas e adicioná-las à carrinha - $T(n) = O(n)$
  3. Limpeza do vetor das encomendas por entregar a modos de refletir a entrega das encomendas - $T(n) = O(n)$
- Conclui-se que a complexidade do algoritmo é da ordem de $T(n) = O(n \log n)$, em que n representa o número total de encomendas a entregar.

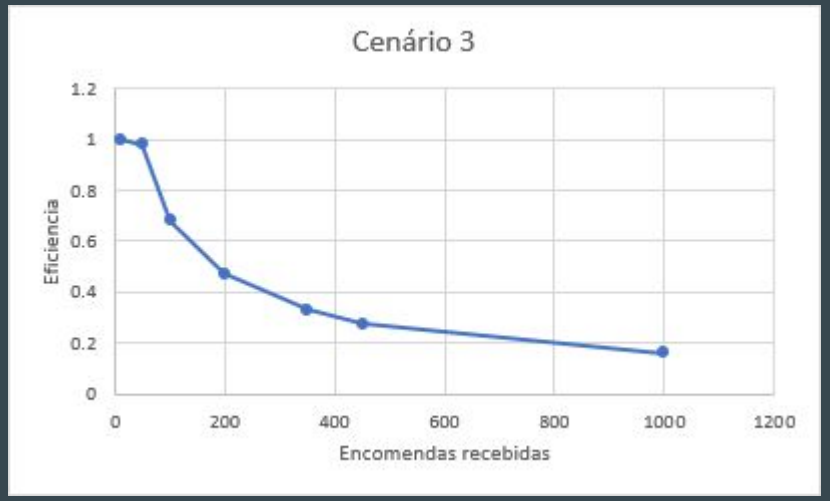
Complexidade espacial:

- Nenhum espaço adicional é utilizado, $S(n) = O(1)$

Resultados da Avaliação Empírica

Cenário 3 - otimização das entregas expresso





A avaliação empírica dos dados leva-nos a acreditar que, à medida que o número de encomendas recebidas aumenta, o tempo médio também deverá, embora, em intervalos cada vez mais pequenos. A eficiência diminui drasticamente.

Tarefas de valorização e observações

Das tarefas de valorização propostas foi implementada a que diz respeito ao cálculo da eficiência.

Foi ponderada a implementação da transferência dos pedidos não entregues para o dia seguinte, passando a priorizar os mais antigos, porém, da maneira como o nosso programa está estruturado, não haveria várias prioridades, pois todas as encomendas são recebidas ao mesmo tempo.

Relativamente às avaliações empíricas dos algoritmos, é de notar que estas foram feitas com subsets dos dados originais, logo, podem estar "viciadas" e não representar a realidade fielmente. Constata-se também o facto de o número de carrinhas se manter fixo, ao longo das experiências.

Dificuldades/Esforço

Dificuldades:

- As principais dificuldades deste trabalho foram a conceção dos algoritmos para a realização dos cenários e a decisão dos compromissos a fazer para tornar estes algoritmos viáveis.

Divisão do esforço:

- Gustavo Costa (up202004187) - 35%
- Ricardo Cavalheiro (up202005103) - 35%
- Mª Leonor Beirão (up201806798) - 30%