Introdução à Otimização Combinatória Aplicada

Segundo Trabalho Prático
Entrega Sugerida: 27/06/2020

1 Introdução

No segundo Trabalho Prático (TP02) será solicitada a entrega de um arquivo **.zip** contendo um programa que solucione o problema apresentado na próxima seção, bem como, para cada instância, um arquivo **.sol** contendo a solução que seu programa encontrou para a mesma. Por conta de detalhes do corretor, no **.zip** também devem estar os arquivos **Makefile** e **verifier.py**, que são disponibilizados junto do trabalho. Além disso, os itens a seguir devem ser respeitados:

- O TP02 deve ser feito individualmente e plágio não será tolerado;
- O TP02 deve ser entregue no run codes (https://run.codes);
- Seu programa deve apresentar um código em uma das seguintes linguagens (C, C++, Java, Python 3) e deve conter um cabeçalho com informações do estudante, como: nome, curso (caso se aplique) e RA (ou RG, se for membro externo);
- Cada estudante deve se cadastrar no run codes (https://run.codes) informando Nome Completo, escolhendo "UFSCar - Universidade Federal de São Carlos" no campo Universidade e colocando seu RA (ou RG, se for membro externo) no campo Núm. Matrícula. Depois de cadastrado, basta logar no run codes e se matricular na disciplina "IOCA - Introdução à Otimização Combinatória Aplicada" usando o Código de Matrícula EZMQ.
- Caso utilize PL ou PLI, pode usar resolvedores como OR-Tools, CPLEX e Gurobi.

2 Localização de Instalações com Capacidades

Neste trabalho, seu objetivo é projetar um algoritmo para resolver um problema enfrentado por empresas de distribuição, que fazem uso de instalações de armazenamento para fornecer mercadorias para diversos clientes. A finalidade deste problema é determinar quais instalações serão as mais rentáveis para servir os clientes. A complexidade do mesmo vem do fato de que cada instalação possui custos e capacidades de armazenamento diferentes, bem como cada cliente possui uma demanda específica.

Podemos formular matematicamente o problema da seguinte forma: temos um conjunto de instalações que devemos escolher abrir ou não, e um conjunto de clientes que precisam ser atendidos. Cada instalação $f \in N$ possui um custo de abertura s_f e uma capacidade de armazenamento cap_f. Cada cliente $c \in M$ possui uma demanda d_c. As instalações e os clientes estão localizados no espaço euclidiano, de modo que cada $i \in N \cup M$ tem coordenadas $\langle x_i, y_i \rangle$. O custo para entregar mercadorias a um cliente específico c a partir de uma instalação f é a

distância euclidiana entre os dois locais, $dist(f, c) = \sqrt{(x_f - x_c)^2 + (y_f - y_c)^2}$. Por fim, todos os clientes devem ser atendidos por exatamente uma instalação, e cada instalação pode atender um número w qualquer de clientes ($0 \leq w \leq |M|$), desde que o somatório da demanda desses clientes não ultrapasse a capacidade da instalação. O objetivo é, então, definir quais instalações abrir, de forma que minimize os gastos de abertura e de transporte.

Entrada:

Cada instância contém um único caso de teste. A primeira linha contém dois inteiros: o número de instalações N e o número de clientes M. As próximas N linhas contêm as informações das instalações. Especificamente, cada linha tem 4 números, o primeiro indicando o custo de abertura s_f, o segundo correspondendo à capacidade cap_f, seguido da localização da instalação no espaço euclidiano, x_f, y_f. Por fim, as M linhas seguintes guardam as informações dos clientes. Cada linha possui três números, indicando sua demanda d_c e localização x_c, y_c.

Saída:

A saída de seu programa deve ser composta por duas linhas. A primeira contendo obj, o custo total da solução encontrada (i.e. o valor objetivo) como um número real com duas casas decimais. A próxima linha é uma lista de M valores, um para cada cliente c, sendo que o i-ésimo valor corresponde ao índice da instalação a que o cliente i está conectado.

Exemplo de entrada

No exemplo abaixo temos um cenário com 3 possíveis instalações e 4 clientes. Todas as instalações possuem custo de abertura e capacidade igual a 100, exceto $cap_2 = 500$. Os dois primeiros clientes possuem demanda igual a 50, enquanto os dois últimos, 75. Os pontos $\langle 1065.0, 1065.0\rangle$, $\langle 1062.0, 1062.0\rangle$, $\langle 0.0, 0.0\rangle$, $\langle 1397.0, 1397.0\rangle$, $\langle 1398.0, 1398.0\rangle$, $\langle 1399.0, 1399.0\rangle$, $\langle 586.0, 586.0\rangle$ são as localizações, no espaço euclidiano, das instalações 0, 1, 2 e dos clientes 0, 1, 2, 3, respectivamente.

```
3 4
100 100 1065.0 1065.0
100 100 1062.0 1062.0
100 500 0.0 0.0
50 1397.0 1397.0
50 1398.0 1398.0
75 1399.0 1399.0
75 586.0 586.0
```

Saída esperada para esse exemplo

Neste resultado estamos escolhendo conectar 0 e 1 com a instalação 1, 2 com a instalação 0, e 3 com a instalação 2, resultando no custo 2550.01.

```
2550.01
1 1 0 2
```

3 Instruções

Disponibilizamos o trabalho no arquivo **tp02.zip**. Este contém o arquivo **solver.py** que realiza a leitura de uma instância, executa um algoritmo básico para o problema, desconsiderando os conflitos, devolve uma solução no formato de saída descrito acima, e grava essa solução no arquivo <**nome_da_instancia**>**.sol** na mesma pasta da instância lida. Modifique a função **CFLNaive()** do **solver.py** para resolver o problema proposto. Sua implementação pode ser testada com o comando

```
python ./solver.py ./data/<nome_da_instancia>
```

Você pode escolher entre implementar sua solução diretamente em python, modificar a função para chamar uma aplicação externa, ou desenvolver um código na linguagem que desejar, que substitua o **solver.py**.

As instâncias que devem ser resolvidas também estão no **tp02.zip**, mais especificamente na pasta **./data/**. Também disponibilizamos algumas instâncias menores na pasta raiz, cujas soluções não serão cobradas, mas que podem ser úteis nos seus testes durante a implementação.

Sua submissão deve corresponder a um arquivo **.zip** que contém, além do seu programa, um arquivo <**nome_da_instancia**>**.sol** para cada instância. O conteúdo desses deve ser a solução encontrada por seu programa ao resolver a instância correspondente. No **.zip** também devem estar os arquivos **Makefile** e **verifier.py**, que estão disponíveis na pasta **./sols/** do **tp02.zip**. Em particular, vocês podem usar o programa **verifier.py**, que valida as soluções encontradas, para facilitar seus testes durante a implementação.

Cálculo da Nota

Soluções inviáveis (i.e., aquelas que não seguem o formato de saída ou violam as restrições do problema) irão receber 0 pontos. Soluções viáveis irão receber pelo menos 3.33 pontos. Soluções viáveis superando um primeiro padrão de qualidade irão receber 6.66 pontos, e soluções que superam um alto padrão de qualidade irão receber 10 pontos. No run codes existem testes distintos, com a mesma instância, para verificar esses diferentes critérios. Para ser aprovado, o estudante deve obter pelo menos 70% dos pontos do trabalho.