Projeto de MC548: Problema da Rota com Coleta e Entrega

Prof. Flávio K. Miyazawa - 1s2021

Versão 1

1 Introdução

Nos tempos de covid, o problema de entrega de produtos de empresas para a casa dos consumidores ficou bastante comum e demanda por soluções com um bom custo. Um exemplo disso é o problema de entrega de comida. Os consumidores compram comida em restaurantes que devem ser entregues em suas casas. Isto gera um pedido i que deve ser coletado (pick) no restaurante p_i e entregue no destino (delivery) d_i. Os três projetos de implementação a serem tratados na disciplina são referente a este mesmo problema, e trata-se de uma versão simplificada da versão prática (não possui restrições de capacidade, nem janelas de tempo, multiplos entregadores/rotas, etc).

O projeto pode ser feito por um ou por dois participantes e consiste em uma implementação em C++ (compilável em linux/gcc por Makefile e linha de comando) de um programa para o problema da Rota com Coleta e Entrega. Caso o projeto seja feito apenas por um aluno, a nota final será multiplicada por 1.1. As entregas estão divididas em três partes (refente a uma implementação por branch and bound combinatório, por programação linear inteira, por meta heurística). A cada momento deverão ser submetidos, um arquivo .tgz contendo todos os fontes do programa e os diretórios dos pacotes utilizados para a compilação do mesmo. Além disso, cada entrega deve conter um relatório. Mais detalhes são dados abaixo.

2 Definição do problema

Seja $G = (V, A)$ um grafo completo direcionado com $2k + 2$ nós em V, que pode ser particionado em três conjuntos: $\{s, t\}$, P e D, onde $V = \{0\} \cup P \cup D$, $P = \{p_1, \ldots, p_k\}$ e $D = \{d_1, \ldots, d_k\}$. O conjunto de arcos A tem custos inteiros $c_a \in [0, M]$ para cada $a \in A$, onde M é um inteiro (haverá uma macro no esqueleto a ser disponibilizado em breve). O nó s é chamado de origem (source) e o nó t é chamado de destino (target). Nas implementações pode considerar $M = 1000$, mas deve ser um parâmetro para a rotina (possivelmente seu algoritmo não tire proveito desta informação).

Um *caminho de coleta e entrega (pick-delivery path)* C é um caminho direcionado hamiltoniano em G (que respeita as direções dos arcos) que começa no vértice s, percorre os demais vértices de V exatamente uma vez cada um e termina no vértice t. Além disso, considerando a sequência em C, cada vértice p_i deve aparecer antes do vértice d_i, para $i = 1, \ldots, k$ (i.e., deve pegar a comida para depois entregá-la). Note que uma solução ótima não necessariamente tem todos os nós de P aparecendo antes dos nós em D). Por simplicidade, vamos supor que o veículo que fará a rota não tem outras restrições, como a quantidade máxima de itens, ou peso ou tempo de trabalho, etc.

O custo de um ciclo de coleta e entrega C é dado pela soma dos custos de todos os arcos em C. O objetivo é encontrar uma solução de menor custo possível.

3 Datas e Prazo

O trabalho está dividido em 3 programas. Os prazos de entrega são:

Programa 1 15 de Junho.

Programa 2 15 de Junho (mesma do Programa 1).

Programa 3 04 de Julho.

O prazo de vários dias é justamente para que o grupo possa investir em uma boa implementação e obter bons resultados.

4 Sobre estratégias de resolução e tamanho do grupo

Em todos os programas, há algumas informações e parâmetros comuns a eles, como os dados de entrada (grafo, custos, pares de vértices, etc). Além disso, haverá os seguintes parâmetros de entrada:

`Tempo_Maximo` (Tempo máximo de execução): Tempo máximo em segundos que sua rotina poderá ficar executando. Mesmo nos programas exatos, sua rotina deverá obedecer o tempo máximo de execução e devolver as informações que puder disponibilizar (como melhores limitantes inferior (se tiver melhorado o limite inferior anterior) e superior (se tiver melhorado o limite superior anterior), melhor solução viável (se tiver melhorado a solução anterior) encontradas até o momento)

`Limitante_Inferior` (Limitante inferior para uma solução ótima): Valor que toda solução ótima tem que ter. É um valor que poderá ser atualizado, caso sua rotina encontre um limitante melhor. Caso não haja um valor inicial, este valor poderá ser zero, já que todos os custos nos arcos são não negativos).

`Limitante_Superior` (Limitante superior para uma solução ótima): Valor que indica que já se conhece uma solução viável com o valor dado. No caso de não se conhecer uma solução inicial, este valor será igual a INF (para representar infinito). Este parâmetro poderá ser atualizado com um valor melhor, caso sua rotina encontre uma solução de valor menor.

`Solução_Viável` (Solução viável gerada pela sua rotina): Caso o parâmetro anterior (`Limitante_Superior`) não seja INF, a rotina recebe uma solução com valor igual ao do parâmetro anterior. Ao termino da execução do algoritmo, caso sua rotina encontre uma solução melhor que a dada na entrada, você deverá devolver/atualizar a solução que encontrou neste parâmetro e atualizar o parâmetro (`Limitante_Superior`) com seu valor.

Obs. : Caso você encontre uma solução ótima (no caso dos algoritmos exatos), você deve atualizar os parâmetros `Limitante_Inferior`, `Limitante_Superior` para serem iguais ao valor da sua solução.

Programa 1 - Algoritmo exato : Neste programa, o grupo deve implementar um algoritmo exato, utilizando a abordagem *branch and bound*. Neste item, não é para utilizar programação linear inteira. Note que seu programa poderá se beneficiar de heurísticas, que é a parte do Programa 2 (veja o próximo item), para obter limitantes superiores e com isso ter um melhor desempenho ao podar ramos não promissores. Seu programa deve conter a implementação de alguma abordagem para podar ramos não promissores (o *bound* do *branch-and-bound*). A descrição de como é feita essa poda deve estar no relatório. Um exemplo de limitante inferior é a árvore geradora de peso mínimo, mas sugere-se que o grupo pesquise por limitantes melhores. Para esta parte, o grupo poderá utilizar até 10% do tempo total nesta parte, apenas para execução de heurísticas/metaheurísticas contidas no Programa 2 (o grupo tem a liberdade de usar

menos tempo ou mesmo não utilizar as heurísticas da parte do Programa 2). Note que a entrega desta parte é relativa a apenas a parte do Programa 2 e deve conter todas as heurísticas utilizadas pelo algoritmo exato.

Programa 2 - Heurísticas e/ou Metaheurística Esta parte pode ser feita por um conjunto de heurísticas ou por uma metaheurística.

Pode ser um conjunto de (no mínimo) duas heurísticas, que podem ser construtivas, gulosas, de melhoria, busca local, etc. Estas heurísticas devem ser na prática heurísticas razoavelmente rápidas, mas é desejável que obtenha soluções melhores caso a heurística permaneça executando por mais tempo. Note que a implementação de mais heurísticas com características diferentes pode levar a soluções melhores. Ao fazer um conjunto de heurísticas, é recomendado que o grupo procure combiná-las na obtenção de uma heurística que produza as soluções melhores. Por exemplo, uma heurística poderia simplesmente chamar (todas) as heurísticas simples ou de busca local e devolver a melhor solução. Outra alternativa seria fazer um Multi-Start Local Search, que itera até um limite de tempo e em cada iteração gera soluções iniciais das heurísticas construtivas e executa heurísticas de busca local. Heurísticas de melhoria/busca local poderão ser úteis na elaboração dos algoritmos exatos e meta heurísticas. Caberá ao grupo decidir como combinar as rotinas de maneira a dar os melhores resultados.

Outra alternativa é a implementação de apenas uma metaheurística (como GRASP, Busca Tabu, Algoritmo Genético ou outra a escolha do grupo). Uma sugestão é implementar um algoritmo genético utilizando o pacote BRKGA (disponível nas páginas do Maurício Resende em `http://mauricio.resende.info/src/index.html`, no item *A C++ application programming interface for biased random-key genetic algorithms* e com várias palestras sobre o pacote do *Biased random-key genetic algorithms* no link `http://mauricio.resende.info/cv/talks.html`), que pode dar bons resultados e já tem toda uma estrutura para facilitar a implementação de um algoritmo genético.

Programa 3 - Programa Linear Inteiro: Um algoritmo exato usando programação linear inteira. Neste programa, o grupo deverá utilizar o pacote de programação linear inteira para resolver o problema. Para isto, a rotina desta parte poderá utilizar até 10% do tempo (pode ser menos, ou mesmo não utilizar) para executar as heurísticas da parte do Programa 2 para obter limitantes superiores (todos os códigos das heurísticas anteriores poderão ser utilizados nesta parte).

Se a execução do algoritmo exato atingir o limite de tempo, ela deve retornar a melhor solução viável encontrada até o momento (se houver alguma). Também deve devolver um limitante inferior, e um limitante superior (pode ser o valor da melhor solução encontrada). Note que o limitante inferior pode ser usado para testar se a sua solução é ótima. Se você estiver usando um resolvedor de programação linear, existem formas de se obter esses limitantes diretamente.

Obs. Em qualquer dos programas acima, a rotina correspondente retorna um valor inteiro, igual a 1 se a rotina obteve um valor melhor que a solução de entrada ou 0 caso contrário. Todos os programas utilizarão a biblioteca LEMON - *Library for Efficient Modeling and Optimization in Networks*, disponível no link `https://lemon.cs.elte.hu/trac/lemon`. Na parte de programação linear inteira, será utilizado o pacote Gurobi, disponível para a área acadêmica.

Observações:

- O projeto pode ser feito por 1 ou 2 alunos. Caso um projeto seja feito de maneira individual, os programas serão corrigidos normalmente, porém a nota final do trabalho

correspondente será multiplicada por 1.1, para valorizar o esforço adicional (limitando cada uma das notas a 10).

- O algoritmo exato branch and bound que conseguir resolver de maneira exata as maiores entradas, dentro de um mesmo tempo testado pelo professor, terá a nota do correspondente trabalho multiplicada por 1.1. O segundo melhor trabalho neste sentido, terá a nota do correspondente trabalho multiplicada por 1.05. Todas as notas serão limitadas a 10.0. Situações de empate ou onde a comparação é mais complexa onde não há uma comparação clara, serão resolvidas pelo professor através de mais entradas, tempos diferenciados ou mesmo pelos melhores códigos, implementações e estruturas de dados.

- A heurística que conseguir obter as melhores soluções dentro de um mesmo tempo testado pelo professor, terá a nota do correspondente trabalho multiplicada por 1.1. O segundo melhor trabalho neste sentido, terá a nota do correspondente trabalho multiplicada por 1.05. Todas as notas serão limitadas a 10.0. Situações de empate ou onde a comparação é mais complexa onde não há uma comparação clara, serão resolvidas pelo professor através de mais entradas, tempos diferenciados ou mesmo pelos melhores códigos, implementações e estruturas de dados.

- O melhor algoritmo exato por programação linear inteira que conseguir obter as melhores soluções dentro de um mesmo tempo testado pelo professor, terá a nota do correspondente trabalho multiplicada por 1.2. O segundo melhor trabalho neste sentido, terá a nota do correspondente trabalho multiplicada por 1.1. Todas as notas serão limitadas a 10.0. Situações de empate ou onde a comparação é mais complexa onde não há uma comparação clara, serão resolvidas pelo professor através de mais entradas, tempos diferenciados ou mesmo pelos melhores códigos, implementações e estruturas de dados.

- Os relatórios devem contemplar a descrição dos algoritmos e heurísticas e tabelas com resultados computacionais sobre o desempenho dos programas.

- Os arquivos a serem entregues devem conter todos os códigos para poderem ser compilados e executados. Assim, caso você use alguma biblioteca particular, você deve incluir esta biblioteca na entrega do programa. A entrega deve ser por email e o nome do arquivo deve ter o formato: `programa_1_ra999999.tgz` se for individual ou `programa_1_ra999999_ra888888.tgz` se for em dupla. A extração dos programas deve gerar uma pasta com o nome `programa_1_ra999999` ou `programa_1_ra999999_ra888888`, respectivamente. Analogamente, para os programas 2 e 3.