Trabalho 2 - Otimização

João Okimoto - GRR20186983

25/09/2020

1 Introdução ao Problema

Leopoldina tem n lugares para visitar, e entre cada lugar existe uma quantidade de tempo gasto para o deslocamento. Ela gostaria de visitar todos os lugares no menor tempo possível e retornar para casa, e além disso, é necessário que alguns lugares sejam visitados em ordem específica. Podemos formular o problema de forma mais formal e em notação mais precisa.

2 Modelagem e TSP

Representamos os lugares a serem visitados por um grafo completo $G = (V, E)$, no qual V é o conjunto de vértices (lugares) e E o conjunto de arestas (o caminho entre cada lugar). Podemos representar o tempo gasto para se deslocar de um lugar para o outro por uma função $C : E \to \mathbb{Z}^+$. O conjunto de lugares que precisam ser visitados em ordem é representado por $R \subset E$. O problema pode ser portanto descrito como:

Encontrar um circuito Hamiltoniano X, tal que a soma

$$Custo(X) = \sum_{e \in E(X)} C(e)$$

seja mínima, de forma que

$$R \subset X.$$

Este problema é uma variação do *Problema do Caixeiro Viajante* (*Traveling Salesman Problem* ou TSP), o leitor pode consultar a seção 34.5.4 de [1] para mais informações. A variação introduzida é justamente a necessidade de visitar uma certa quantidade de lugares em ordem.

2.1 Backtracking

Na seção 4.6.2 [2], o autor propõe utilizar *Backtracking* para resolver o problema do TSP original. Essa técnica procura resolver o problema por força bruta, exaustando a quantidade de possibilidades de soluções do problema. Podemos ilustrar isso utilizando uma árvore n-ária onde cada nodo é um lugar a ser visitado e uma lista $W = [x_0, x_1, \ldots, x_n - 1]$ que representa um caminho que inicia na raiz e termina em uma das folhas da árvore. Temos como exemplo a figura 1, no qual $W = [0, 2, 1, 3]$ é o trajeto destacado.

O algoritmo que utiliza a técnica de backtracking visita todos os nodos folha da árvore, e possui complexidade de tempo $O(n!)$.

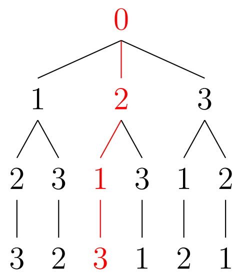

Figura 1: Árvore n-ária que representa os possíveis caminhos, com o trajeto $W = [0, 2, 1, 3]$ destacado.

2.2 Branch and Bound

Na seção 4.2 de [2], o autor introduz o conceito de *poda* (*pruning*) da árvore, no qual ao computar um conjunto C_l chamado de *conjunto de decisão* em cada passo l obtemos um conjunto de caminhos que a solução atual ainda pode seguir, o que representa os filhos de um nodo qualquer na árvore. A poda ocorre quando excluímos certos nodos de um trajeto, por exemplo, ao determinar que o nodo 1 deve ser visitado antes do nodo 3, o trajeto $W = [0, 3, 2, 1]$ é inviável, e portanto todas as soluções da forma $[0, 3, x_2, x_3]$ são descartadas.

Na seção 4.6, uma outra maneira de podar a árvore de estado é apresentada, utilizando o conceito de *funções limitantes* (*bounding functions*). Dado que $W = [x_0, x_1, \dots, x_l]$ é uma solução parcial do problema ($l \leq n$), denotamos como $P(W)$ o menor custo de todas as possíveis soluções que envolvam os descendentes de W. Então definimos que uma função B limitante inferior deve satisfazer

$$B(W) \leq P(W)$$

para qualquer solução parcial e viável que é descendente de W.

Funções limitadores podem ser usadas para podar a árvore de estados. Se no processo de percorrer todo o espaço de soluções, a atual solução parcial é $X = [x_1, x_2, \dots, x_l]$, e o menor custo encontrado é $OPTc$, então se $B(X) \geq OPTc$ pela definição tem-se que

$$P(X) \geq B(X) \geq OPTc$$

o que significa que qualquer descendente de X não pode gerar uma solução melhor que a atual, e portanto, eles são descartados. A ideia é que B seja uma função fácil de computar e que aproxime bem o menor custo de qualquer descendente da solução parcial, fazendo com que não seja necessário percorrer todo o espaço de soluções, economizando tempo.

Uma outra maneira de incorporar funções limitantes para resolver o TSP é utilizando a técnica de *Branch and Bound* descrita na seção 4.7 de [2]. Essa técnica utiliza as funções limitantes para determinar a ordem na qual as chamadas recursivas ocorrem a partir do conjunto de decisão C_l. A técnica de backtracking utiliza uma ordem qualquer para visitar os $n - l$ filhos de uma solução parcial $X = [x_1, x_2, \dots, x_l]$, no entanto, podemos calcular $B(\hat{X})$ para todos os filhos, e ordenar o conjunto de decisão de forma que cada caminho possua um valor de $B(\hat{X})$ crescente. Isso faz com que as chances de encontrar uma solução melhor que a atual seja maior ao seguir os primeiros elementos do conjunto, fazendo com que seja bem provável que os últimos elementos sejam podados.

2.3 Restrições do problema

O problema possui restrições na ordem em que os lugares devem ser visitados por leopoldina, fazendo com que o espaço de soluções do TSP seja modificado. Desta forma, é necessário que a factibilidade de cada nodo seja avaliada, o que é feito através do algoritmo 1, apresentado a seguir:

Algorithm 1 Factibilidade

function ISFEASIBLE($R, X, current$)
    $restricted \leftarrow false$
    for $y_i \in R$ **do**
        if $y_i = current$ **then**
            $restricted \leftarrow true$
            $dependency \leftarrow y_{i-1}$
    if $restricted = false$ **then**
        return $true$
    for $x_i \in X$ **do**
        if $dependency = x_i$ **then**
            return $true$
    return $false$

O algoritmo analisa uma solução parcial X que possui como candidato a próximo lugar a ser visitado *current*. Inicialmente o conjunto de restrições R é percorrido, e caso o candidato esteja incluso, a flag *restricted* é levantada. Esta flag indica que o elemento deve ser visitado após o elemento *dependency*, e portanto, para que a solução seja fáctivel, a solução parcial X é percorrida em busca do elemento que deveria ter sido incluso. Caso o elemento não tenha sido ainda visitado, a solução é descartada e a sub-árvore é podada.

3 Análise das Funções Limitantes

A implementação utiliza duas funções limitantes, a função **MinCostBound()** e **ReduceBound**, descritas na seção 4.6.2 de [2].

3.1 MinCostBound()

A função limitante **MinCostBound** é derivada do teorema 4.2 de [2]. Dado que x é um vértice qualquer de G, e W é um caminho de tamanho arbitrário, definimos

$$b(x, W) = \min\{C(x, y) : y \in W\},$$

como sendo o menor custo entre um nodo x e um nodo qualquer $y \in W$. O teorema afirma que se $\hat{X} = [x_0, x_1, \ldots, x_n]$ é o trajeto de custo mínimo que extende a solução parcial $[x_0, x_1, \ldots, x_l]$ para $l \leq n - 1$, então

$$Custo(\hat{X}) \geq \sum_{i=0}^{l-1} C(x_i, x_{i+1}) + b(x_{l-1}, Y) + \sum_{y \in Y} b(y, Y \cup \{x_0\})$$

no qual $Y = V \setminus \{x_0, x_1, \ldots, x_{l-1}\}$, a extensão da solução parcial X. O primeiro termo do lado direito da inequação corresponde a todo custo acumulado do trajeto já percorrido, dado pelo custo de percorrer a solução parcial X. O segundo termo correponde ao menor custo entre o atual lugar e o próximo. Por fim, o terceiro corresponde a todo custo mínimo necessário para terminar o circuito. A prova do teorema pode ser encontrada na mesma seção, mas a idéia central é a de que este é o menor custo possível que as soluções que incluem os descendentes de X possuem para "fechar" o circuito.

Pela definição de função limitante o autor define

$$MCB(X) = \begin{cases} \sum_{i=0}^{l-1} C(x_i, x_{i+1}) + b(x_{l-1}, Y) + \sum_{y \in Y} b(y, Y \cup \{x_0\}) & \text{se } l \leq n-1 \\ \sum_{i=0}^{l-1} C(x_i, x_{i+1}) + C(x_{n-1}, x_0) & \text{se } l = n \end{cases}$$

que pode ser computada em tempo $O(n^2)$.

3.2 ReduceBound()

A função **ReduceBound** utiliza operações de *redução de matriz* para definir um limite inferior sobre $Custo(x)$, e é definida pelo teorema 4.3 de [2]. A operação de redução envolve transformar uma matriz M qualquer, de forma que

- Todos os elemento são não-negativos.
- Toda linha de M possui pelo menos um elemento igual a 0.
- Toda coluna de M possui pelo menos um elemento igual a 0.

A operação de redução é apresentada no algoritmo 4.11 de [2], que também retorna um valor $Reduce(M)$. De acordo com o teorema, se $M_{i,j} = C(i,j)$, então qualquer circuito hamiltoniano de G possui custo de no mínimo $Reduce(M)$. Supondo que $X = [x_0, x_1, \ldots, x_{l-1}]$ é uma solução parcial do TSP, o autor define a função limitante

$$ReduceBound(X) = Reduce(M'(X)) + \sum_{i=0}^{l-2} M[i, i+1] \tag{1}$$

no qual $M'(X)$ é a matriz resultante das seguintes operações

- se $l < n$, $M[x_{l-1}, 0] = \infty$.
- delete as linhas $x_0, x_1, \ldots, x_{l-2}$ de M.
- delete as colunas $x_1, x_2, \ldots, x_{l-1}$ de M.

o que resulta em uma matriz $(n-l+1) \times (n-l+1)$. Estas operações são feita sobre a matriz de modo que ela represente o menor custo possível do trajeto que ainda precisa ser completado $[x_l, x_{l+1}, \ldots, x_{n-1}]$. A primeira operação garante que o ponto de partida não esteja incluso nas possibilidades do próximo nodo x_l, fazendo com o que o circuito não feche sem visitar todos os nodos. A segunda faz com que os custos de caminhos que saem dos nodos $x_0, x_1, \ldots, x_{l-2}$ não sejam considerados, pois estes já foram inclusos no custo da solução parcial. Por fim, a terceira garante que qualquer custo de caminhos que entram nos nodos $x_1, x_2, \ldots, x_{l-1}$ não seja considerado, pelo mesmo motivo. Esta matriz representa de certa forma o conjunto de decisões do algoritmo de backtracking, pois armazena todos os custos que ainda podem ser considerados. Em (1) o primeiro termo $Reduce(M'(X))$ representa justamente o limite inferior para o custo de soluções que são descendentes da solução parcial $X = [x_0, x_1, \ldots, x_{l-1}]$, enquanto o somatório representa o custo do caminho X.

4 Implementação

De forma a modelar e resolver o problema proposto, a linguagem de programação escolhida foi C++ devido a sua rapidez, conjunto de bibliotecas padrões e possibilidade de programação orientada à objetos. Foram utilizadas três bibliotecas padrões para a implementação do solver proposto, **vector** que permite o uso de vetores dinâmicos e **chrono** que permitiu calcular o tempo de execução do solver.

A implementação segue o algoritmo 4.23 descrito em [2] com uma pequena modificação para resolver o problema dado. Esta modificação inclui uma etapa de verificação de soluções viáveis utilizando o algoritmo 1. O solver portanto utiliza a técnica de branch and bound descrita na seção 2, juntamente com a opção de escolha de uma das duas funções descritas na seção 3.

O solver também utiliza uma estrutura de dados declarada no arquivo **graph.cpp**, que armazena a matriz de custos $n \times n$, além da matriz de restrições $2 \times m$.

Referências

[1] Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein. *Introduction to Algorithms*. MIT Press, 2009.

[2] Kreher Stinson. *Combinatorial Algorithms*. CRC Press, 1999.