UNIVERSIDADE FEDERAL DE LAVRAS

CLAUDINEI MOREIRA DA SILVA
FRANCISCONE LUIZ DE ALMEIDA JUNIOR
IGOR EMANUEL CARVALHO DA CRUZ
VICTOR HUGO DE ANDRADE LANDIN

Algoritmo de Dijkstra aplicado ao Problema do Caixeiro Viajante

Lavras - MG

2019

Introdução

Um problema bastante comum na área de transporte é o de encontrar rotas que diminuam algum determinado custo, como por exemplo a distância percorrida, o tempo gasto, o consumo de combustível, entre outras métricas. Esse problema pode ser utilizado em diversas empresas, principalmente as responsáveis por realizar entregas em diversas cidades, onde a sequência das cidades visitadas pode interferir consideravelmente no custo adquirido.

Como solução a este problema, o presente trabalho visa demonstrar uma aplicação do algoritmo de Dijkstra para encontrar soluções referentes ao Problema do Caixeiro Viajante, ao qual é aplicado no problema de se encontrar rotas adequadas para um caminhão ao qual realiza entregas em diversas cidades, de modo a reduzir a distância percorrida pelo mesmo.

Utilizando como modelo alguns grafos fictícios, onde os vértices representam as cidades e as arestas representam as distâncias entre estas cidades, aplicamos, à princìpio, um algoritmo sequencial para estimar uma solução à este problema e, posteriormente, aplicamos o mesmo algoritmo de forma paralelizada, realizando o estudo sobre as diferenças no desempenho à medida em que se aumenta o número de processos utilizados para a paralelização.

Referencial Teórico

1.1 Definição de Grafo:

Um grafo é um conjunto de vértices e um conjunto de arestas que ligam pares de vértices distintos.

Um grafo ***G(V,A)*** é definido pelo par de conjuntos *V* e *A*, onde:

V - conjunto não vazio: os **vértices** ou **nós** do grafo;

A - conjunto de pares ordenados *a*=(v,w), v e w ∈ V: as **arestas** do grafo.

1.2 Problema do Caminho Mínimo:

O Problema do Caminho Mínimo consiste em encontrar o caminho de menor custo entre dois vértices, sendo este caminho composto por uma só aresta ou passando por outros vértices. O custo referente ao problema pode representar distância, gasto monetário, entre outros.

1.3 Algoritmo de Dijkstra:

O algoritmo de Dijkstra foi proposto para se obter uma solução ao problema do Caminho Mínimo de uma origem para os demais vértices em um grafo com custos positivos nos arcos. No início rotula-se a origem como 0 e os vértices

restantes com um valor proporcionalmente grande, após isso cria-se duas listas, uma com os vértices já adicionados a solução, começando somente com a origem e a outra lista com os demais vértices. A partir do primeiro vértice na lista solução, encontra todos seus adjacentes e rotula-os com a distância do primeiro vértice somado com o rótulo do primeiro vértice. Após isto, seleciona o próximo vértice a ser adicionado na lista solução, usando como critério o vértice adjacente ao vértice anteriormente utilizado com a menor distância, repetindo o procedimento de rotular seus adjacentes somando a distância entre eles somado com seu rótulo. Este procedimento é repetido até que todos os vértices sejam adicionados à lista de vértices solução.

```
DIJKSTRA (n, Adj, f, r)      comentário: f ≥ 0
1   para u ← 1 até n faça
2       dist[u] ← ∞
3   dist[r] ← 0
4   Q ← CRIA-FILA-VAZIA ( )
5   para v crescendo de 1 até n faça
6       INSERE-NA-FILA (v, Q)
7   enquanto Q não está vazia faça
8       u ← EXTRAI-MIN (Q)
9       para cada v em Adj[u] faça
10          se dist[u]+f(uv) < dist[v]
11              então DIMINUI-CHAVE (v, Q, dist[u]+f(uv))
12  devolva dist[1..n]
```

2.1 Problema do Caixeiro Viajante:

O Problema do Caixeiro Viajante (PCV) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

Este problema consiste em determinar em um grafo G = (N,M) ponderado em arestas (direcionado ou não) um ciclo hamiltoniano de custo mínimo.

2.2 Ciclos Hamiltonianos:

O ciclo Hamiltoniano consiste em um ciclo que percorra o grafo visitando uma única vez cada vértice do grafo e retorne para o vértice inicial.

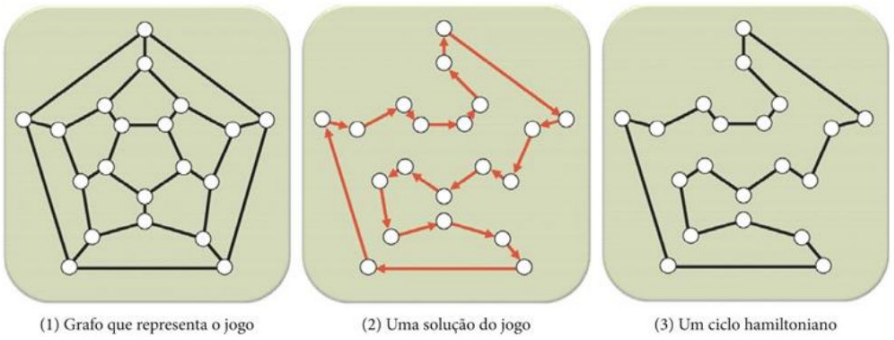

Figura: Fonte – Goldbarg & Goldbarg (2012).

2.3 Bibliotecas:

2.3.1 Mpi4py:

MPI, Message Passing Interface, é um sistema de transmissão de mensagens padronizado e portátil projetado para funcionar em uma grande variedade de computadores paralelos. O padrão define a sintaxe e a semântica das rotinas de bibliotecas e permite aos usuários escrever programas portáveis nas principais linguagens de programação científica (Python, Fortran, C ou C ++).

O mpi4py é a biblioteca que instancia todas as funcionalidades do mpi no python.

2.3.2 MathPlotLib:

O Matplotlib é uma biblioteca de plotagem 2D do Python.

2.3.3 Numpy

NumPy é um pacote para a linguagem Python que suporta arrays e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas.

2.3.4 Networkx

O NetworkX é uma biblioteca Python para estudar gráficos e redes. O NetworkX é um software livre lançado sob a licença BSD-new.

2.3.5 Time

Este módulo fornece várias funções relacionadas ao tempo. Para funcionalidade relacionada, consulte também os módulos datetime e calendar.

Métodos

Com o intuito de encontrar um ciclo hamiltoniano para apresentar uma solução ao Problema do Caixeiro Viajante, a implementação proposta encontra a solução para grafos que sejam hamiltonianos.

1. Data Set:

   O algoritmo recebe um grafo externo através de um arquivo, e o mesmo será extraído para ser manipulado pela classe Graph, explicada posteriormente. Deste

arquivo é recebido a quantidade de vértices do grafo e cada uma de suas arestas da forma: [*vertice vertice peso*], sendo uma aresta por linha do arquivo.

2. Classe Graph:
   A classe Graph é utilizada para instanciar um objeto que manipule o grafo utilizado, nela utilizamos uma matriz de adjacência para representar os vértices e suas arestas com respectivos pesos.
3. Utilização do algoritmo de Dijkstra:
   O algoritmo de Dijkstra é utilizado para gerar árvores de caminhos mínimos a partir de cada vértice, ou seja, cada árvore tem seu respectivo vértice como raiz, sendo o número de árvores igual ao número de vértices. No algoritmo temos a função *dijkstra* que recebe o grafo e o vértice raiz e é gerado a árvore de caminho mínimo referente a este vértice, retornando a mesma.
4. Paralelização:
   Na implementação do algoritmo foram utilizadas rotinas para paraleliza-lo de modo a aproveitar o multiprocessamento, com a finalidade de reduzir o tempo de execução do algoritmo. A biblioteca MPI do python, definida como mpi4py, foi utilizada para realizar essa paralelização do algoritmo, onde a mesma quando instanciada cria n processos, sendo n o valor de processos definido na execução pelo usuário, que executam todo o código, exceto áreas especificadas para um ou mais processos específicos, como por exemplo o processo 0 que faz a leitura do arquivo com o grafo, instancia o objeto grafo e o envia para cada processo, além de receber informações dos demais processos como o menor custo mínimo e o maior tempo de execução.

Resultados

1. Subgrafos
   Como já citado nos métodos a função dijkstra recebe um grafo e o vértice inicial, e com isso o algoritmo calcula a árvore de caminhos mínimos partindo do vértice raiz (vértice inicial). Por definição a árvore de caminhos mínimos é um subgrafo gerado a partir do grafo original. Considerando um grafo que seja hamiltoniano e com pesos diferentes para cada aresta a heurística desenvolvida irá encontrar um ciclo hamiltoniano em algum dos subgrafos gerados pelo algoritmo de *dijkstra*. A figura 1 mostra um exemplo de execução do algoritmo e os subgrafos gerados para cada vértice.

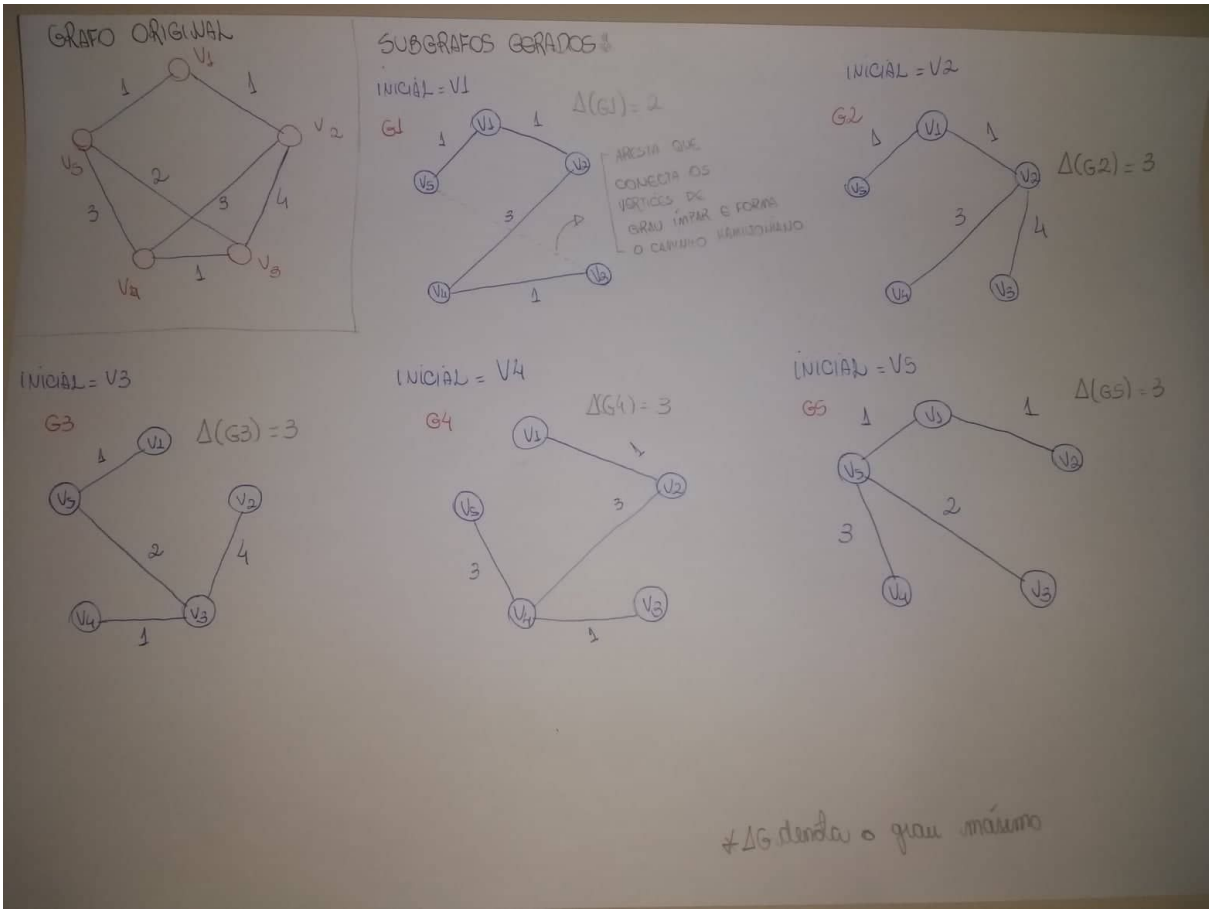

Figura 1: Grafo original e subgrafos gerados pelo algoritmo de dijkstra

2. Solução

   Para encontrar o caminho hamiltoniano que resolve o problema do caixeiro viajante o algoritmo desenvolvido procura, entre os subgrafos gerados, um grafo que possua grau máximo igual a 2 e apenas dois vértices de grau ímpar igual 1. Uma vez que os subgrafos que atendem a essa restrição são encontrados é feito um teste para verificar se existe uma aresta no grafo original que conecta dos dois vértices de grau ímpar, e se essa condição é satisfeita o grafo possui um ciclo hamiltoniano, logo esse grafo é hamiltoniano e resolve o problema do caixeiro viajante.

3. Estatísticas

   Neste trabalho, primeiramente foi implementada uma versão sequencial do algoritmo para fins de comparações. Para uma instância pequena (um grafo com 5 vértices e 16 arestas) o algoritmo sequencial gastou em média 0.000190 segundos para encontrar a solução. Em contrapartida, o algoritmo paralelo gastou em média 0.0006327, 0.001, 0.014 e 0.007 para a quantidade de 1, 2, 3 e 4 entidades de processamento, respectivamente, para essa mesma instância. Esses resultados mostram que para uma instância pequena dos dados o *speedup* em relação ao tempo de execução sequencial foi de aproximadamente 0.19 o que indica uma piora em relação ao tempo de execução sequencial.

   Além disso, o algoritmo foi testado em um grafo com aproximadamente 643 vértices, e nesse caso os resultados da implementação paralela foram bem melhores. Para o algoritmo sequencial o tempo médio de execução foi cerca de 1121 segundos (aproximadamente 20 minutos), enquanto que no algoritmo paralelo, os resultados

foram 623 segundos e 241 segundos para 2 e 8 entidades de processamento, respectivamente. Nessa instância o *speedup* total foi de 1.799, que evidencia a melhora do algoritmo para uma instância de dados volumosa em um algoritmo paralelo.

As métricas calculadas com os valores do speedup foram:

- Métrica da Eficiência $\mathfrak{E}(n,p)$:
  - Instância 1:
    - 2 processadores: 0.095
    - 3 processadores: 0.063
    - 4 processadores: 0.0475
  - Instância 2:
    - 2 processadores: 0.895
    - 3 processadores: 0.59
    - 4 processadores: 0.44

- Métrica de Karp-Flat:
  - Instância 1:
    - 2 processadores: 9.52
    - 3 processadores: 7.39
    - 4 processadores: 6.68
  - Instância 2:
    - 2 processadores: 0.11
    - 3 processadores: 0.33
    - 4 processadores: 0.40

Referências

Python Documentation - Disponível em: https://www.python.org/doc/.

Time - Python Library - Disponível em: https://docs.python.org/3/library/time.html

Overview of NetworkX for python: Biblioteca de visualização em grafos. Disponível em: https://networkx.github.io/documentation/latest/index.html.

Aplicação do algoritmo de Dijkstra para o problema de roteamento da frota de táxis partindo de um ponto fixo - Santos, Heverton e Junior, Alamir. Disponível em: http://bd.centro.iff.edu.br/bitstream/123456789/21/1/Artigo_Producao_Sistemas_Versao_Final.pdf.

KAISER, T., RYJÁČEK, Z., KRÁL, D., et al., "Hamilton cycles in prisms", Journal of Graph Theory, v. 56, pp. 249–269, 2007.

Grafos: conceitos, algoritmos e aplicações. Goldbarg, Marco e Goldbarg, Elizabeth. Rio de Janeiro, Elsevier, 2012.

Slides de Aula do Prof. Dr. Mayron César de Oliveira Moreira - Disponível em acervo pessoal.