TP4 de Algoritmos e Estruturas de Dados III

Lucas O. Silvestre[1]

[1]Sistemas de Informação – Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte, MG – Brazil

lsilvs@dcc.ufmg.br

1. Objetivo inicial

Este trabalho tem como objetivo levar o aluno a lidar com dois tópicos avançados de Algoritmos e Estruturas de Dados, problemas NP-Completo e programação concorrente. Espera-se que o aluno consiga modelar um problema de alto custo computacional e mostrar que não e possível implementar uma solução que consiga resolver entradas não triviais de maneira ótima. Ainda, o aluno deve implementar um algoritmo paralelo para resolver o problema em questão exercitando alguns conceitos básicos de programação concorrente.

A problema apresentado é descobrir qual o maior número possível de pessoas de uma dada rede social, tal que todas elas estejam conectadas (diretamente) entre si.

2. Modelagem e Solução Proposta

A solução implementada utiliza o problema da Clique Máxima para resolver o problema. O problema da Clique diz respeito a qualquer problema que tem por objetivo encontrar sub-grafos completos em um dado grafo. Como no nosso caso, o problema de encontrar grupos em que todos os elementos estão conectados entre si.

Sendo assim, vamos considerar uma rede social onde os vértices do gráfo representam pessoas e as arestas representam que são amigos entre si. Para encontrar o

maior sub-conjunto de pessoas em que todas se conhecem, iremos procurar (força bruta) em todos os sub-conjuntos possíveis e retornar o maior sub-conjunto encontrado. Esse é um processo muito demorado mesmo para redes sociais pequenas. Embora a pesquisa por força bruta possa ser melhorada através de algoritmos mais eficientes, todos estes algoritmos levam um tempo exponencial para resolver o problema. Dessa forma, como podemos comparar nosso problema com o problema da Clique (que é um problema NP-Completo), chegamos à comclusão de que nosso problema também é NP-Completo.

A modelagem desse trabalho foi dada em dois diferentes algoritmos. O primeiro deles resolve as entradas dadas de forma sequencial e pode levar muito tempo para resolvê-las (dependendo do tamanho das entradas). Já o outro algoritmo resolve o problema de forma paralela, tomando vantagem do processadores atuais que possuem mais de um núcleo e conseguem executar mais de uma tarefa ao mesmo tempo. A seguir iremos explicitar melhor cada solução proposta.

2.1. Algoritmo Sequencial

Como modelagem da solução sequencial, foi utilizado uma matriz de adjacência para armazenar os valores das entradas e um tipo abstrato de dados de lista encadeada para pesquisar o grafo a fim de encontrar a Clique Máxima. A função que busca a Clique Máxima percorre todo o grafo montando sub-grafos menores e verifica se o sub-grafo resultante é uma clique. Uma variável de controle armazena a clique máxima encontrada até o momento e é retornada quando não há mais sub-grafos maiores a serem montados.

2.2. Algoritmo Paralelo

O algoritmo da solução paralela utiliza as mesmas funções e estrutura de dados da solução sequencial, porém, implementamos a paralelização para agilizar a resolução

das diversas instâncias do arquivo de entrada. Como o arquivo de entrada contém multiplas entradas, o algoritmo processa mais de uma dessas entradas ao mesmo tempo (dependendo, claro, do número de núcleos que o processador que irá executar o código possui).

Para tal implementação, utilizamos a biblioteca pthread do C, que permite criar threads independentes e executar um pedaço do código em paralelo com outro. Por motivos de simplificação e falta de tempo hábil para tal trabalho, escolhi paralelizar o problema no número de entradas do arquivo. Sei que o ideal seria paralelicar a função que encontra a Clique Máxima, porém não tive tempo suficiente para isso.

A paralelização no número de entradas consiste em, dado o número de threads como parámetro da chamada do programa, criar múltiplas theads que resolvam várias entradas simultaneamente e, assim, reduza o tempo total de execução do programa.

3. Análise de Complexidade

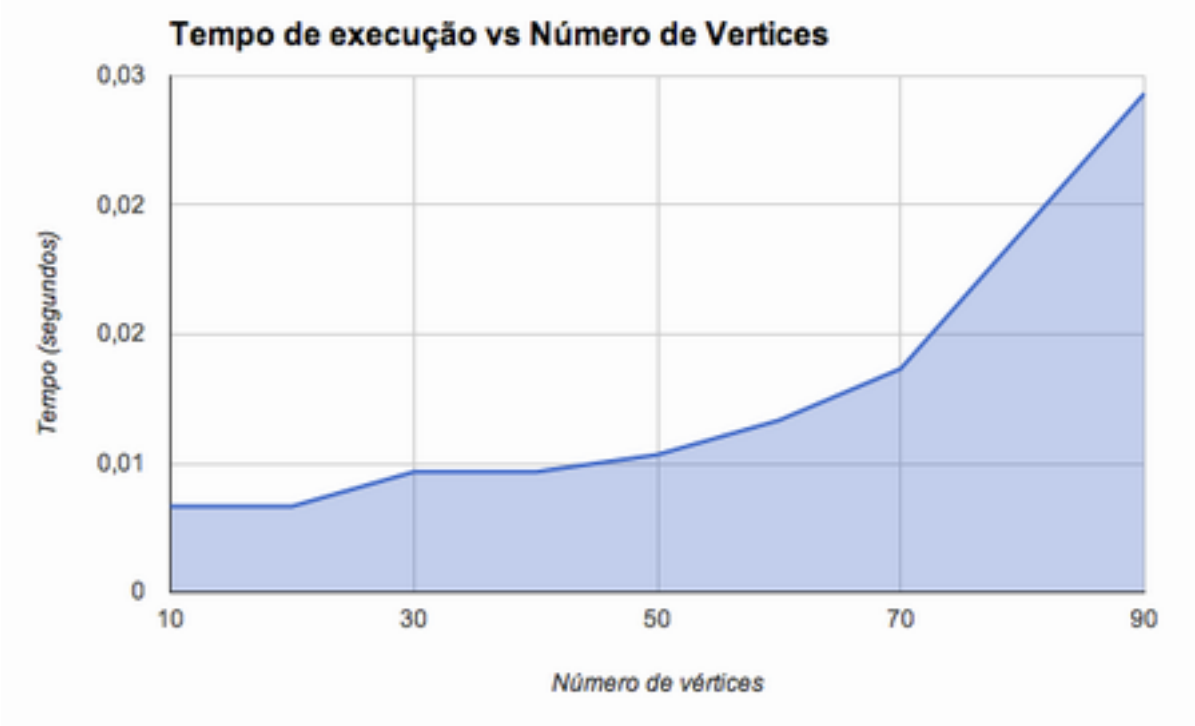
A análise de complexidade de tempo da solução proposta segue a complexidade do problema da Clique. Como ambos são NP-Completo, a complexidade é exponencial no número de vértices do grafo da rede social em questão.

Já a complexidade de espaço, por utilizarmos matriz de adjacência, é $O(n^2)$.

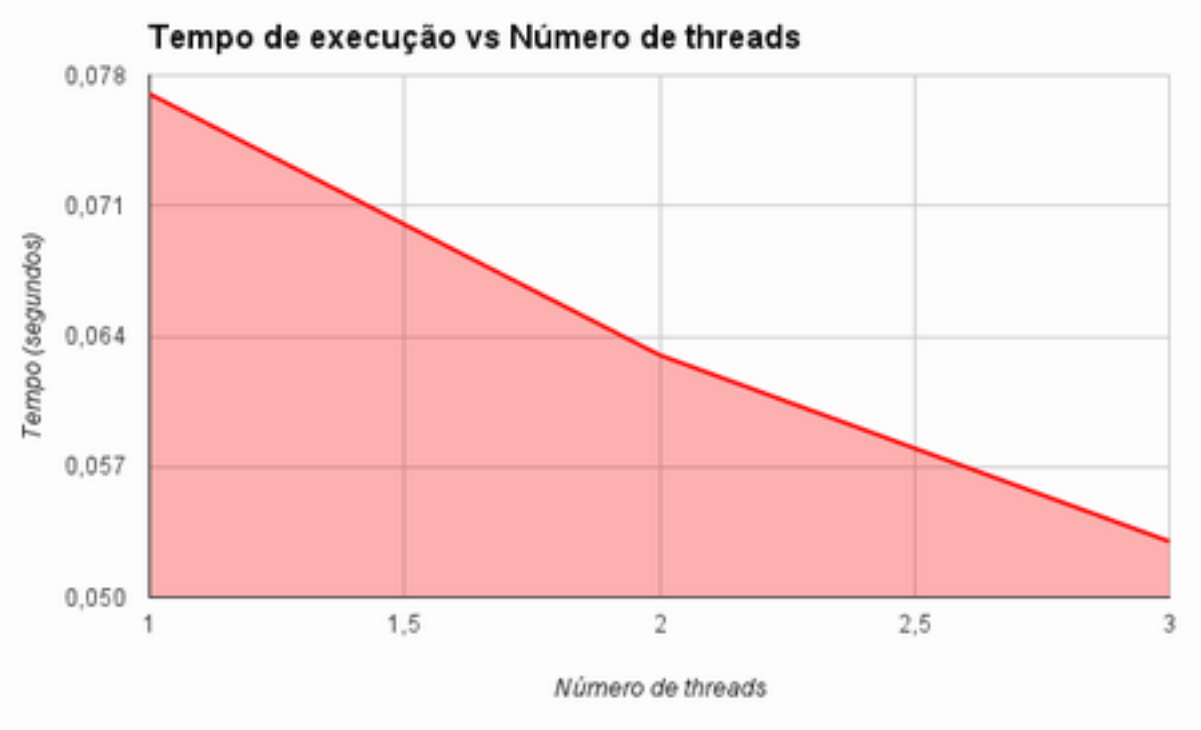
4. Experimentos e Análises

Como não foram disponibilizados arquivos de testes para este trabalho, foi criado alguns para rodar testes e analizá-los. Dois testes básicos foram rodados. O primeiro analisa o impacto do aumento de pessoas na rede social (ou seja, o número de vértices no grafo de entrada). O outro, analisa o impacto do aumento de threads para solução do problema.

4.1. Gráfico de Tempo vs Vértices



4.2. Gráfico de Tempo vs Threads



De acordo com os dados apresentados, percebemos que conforme aumentamos o número de vértices de entrada, mais ingreme é a curva de tempo de execução. Isso já era esperado, uma vez que o problema é exponencial.

Porém, para o gráfico que analisa o número de threads rodando simultaneamente, temos sim uma queda no tempo de execução, porém essa queda não é tão significativa quanto o esperado. Isso porque, pela implementação escolhida e justificada acima, há ainda no código uma grande fração serial que não foi paralelizada. Outra justificativa para o decrescimo sutil do tempo de execução, é o grande problema de desbalanceamento de carga que o algoritmo apresentado possui.

5. Especificação

Todo o desenvolvimento do código foi realizado utilizando uma máquina com as seguintes especificações:

Processador: Intel Core i5 1,7 GHz (2 núcleos e 4 threads)

Memoria: 4 GB 1333 MHz DDR3

Sistema Operacional: OS X 10.8.2

IDE: Sublime Test 2

GCC: i686-apple-darwin11-llvm-gcc-4.2 (GCC) 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00)

6. Conclusão

Como conclusão do trabalho apresentado, concluimos que o problema de se encontrar o maior grupo de pessoas onde todas elas se conheçam dentro de um rede social é um problema NP-Completo e se resume ao problema da Clique Máxima já conhecida pela literatura.

Por utilizar um algoritmo de força bruta, o programa possui complexidade exponencial e pode demorar anos para resolver entradas muito grandes. Essa demora pode ser reduzida fazendo uso do recurso de multi threads dos processadores atuais. Porém, por falta de tempo hábil para resolver o presente trabalho prático, a implementação paralela do código atual ainda matém uma grande fração serial do código, não sendo tão vantagojo quanto poderia ser se paralelizássemos a função que busca a Clique Máxima.

Por fim, ficou claro como utilizar a função pthread do C e seu poder de reduzir o tempo de execução de um programa, tamando vatagem dos processadores atuais que possuem multiplas threads simultaneas.