Análise e Síntese de Algoritmos

Relatório do 2º projeto - 2017-18, 2º semestre

Grupo 08 (Taguspark) - João Freitas (87671), Pedro Soares (87693)

Introdução

O objetivo deste projeto consiste em obter uma segmentação dos pixéis que constituem uma imagem dada como input, classificando-os como sendo de primeiro plano ou de cenário. A segmentação determinada deverá ser a que minimiza o peso total. Este problema pode ser modelado através de uma rede de fluxo e determinando o corte mínimo para essa rede obter-se-á a solução para o problema.

Descrição da solução

A modelação escolhida para este problema foi a seguinte:

- Por cada pixel na imagem, adicionou-se um vértice na rede.
- Por cada relação de vizinhança entre pixéis **u** e **v**, foram adicionados dois arcos na rede entre os dois vértices correspondentes, com sentido de u para **v** e de **v** para **u**, com capacidade correspondente ao peso de cada relação de vizinhança.
- O vértice fonte **s** tem um arco para cada pixel com capacidade correspondente ao peso de primeiro plano do pixel a que se liga.
- Cada pixel da rede possui um arco para o vértice de destino **t** com capacidade correspondente ao peso de cenário do pixel que se liga a **t**.

No início do programa, são lidas as dimensões da imagem em que se irá trabalhar, **m** e **n**, sendo guardadas numa variável global **dimTotal**(m*n+2) para saber quantos pixéis irão ser colocados na representação do grafo. Em seguida é alocada uma lista de adjacências com **dimTotal** pixéis usando a função **initAll()**. Em seguida chamam-se as funções **getLp()** e **getCp()** que recolhem os pesos de primeiro plano e de cenário de cada pixel, respetivamente, e a função **initializeGraph (int m, int n)** que adiciona os arcos, à representação do grafo, que saem de **s** e se ligam a todos os pixéis e todos os arcos que saem dos pixéis e se ligam a **t**, usando a função **InsertEdge(int u, int v,int capacity)** que adiciona uma adjacência ao pixel correspondente, representada por uma estrutura chamada **Edge**. Nesta estrutura são guardadas informações sobre o fluxo e a capacidade do arco, o número do pixel de origem e de destino, e contém também um ponteiro para o pixel adjacente seguinte.

Posteriormente, são chamadas as funções **getHorizontalConnections (int m, int n)** e **getVerticalConnections (int m, int n)** que leem os valores dos pesos das relações de vizinhança entre os pixéis **u** e **v** e atribuem esse valor ao arco de **u** para **v** e de **v** para **u** recorrendo à função **InsertEdge**.

Neste momento ficam finalizados os passos de inicialização, e é aplicado o algoritmo de **Edmonds-Karp** sobre a representação do grafo a partir do vértice **s**.

No início do algoritmo é alocado um vetor **visited** de tamanho **dimTotal** tal que ao ser visitado um pixel pela BFS na rede, será colocado um 1 na posição de índice igual ao número do pixel, caso contrário estará um 0. É também criada uma fila queue pela função createQueue de tamanho dimTotal. Em seguida é inciado um ciclo while que se executará enquanto existir um caminho de aumento se **s** para **t** (**while(BFS()==1)**). Sempre que se encontra um caminho de aumento, é efetuado um backtrace a partir do destino seguindo os predecessores de cada pixel de modo a atualizar a variável **pathFlow** par saber a quantidade de fluxo que pode ser enviado através do caminho encontrado. Ao saber o valor de **pathFlow**, são atualizadas as capacidades residuais dos arcos que fazem parte do caminho e incrementada a variável **max_flow** que representa o fluxo máximo da rede.

Por fim, é impresso o valor de **max_flow**, e como todos os vértices que foram visitados correspondem aos vértices de cenário e os restantes de primeiro plano, é percorrido o vetor **visited**, e se a posição correspondente a cada pixel contiver um 0 nesse caso é impresso um **P** no terminal e um **C** caso contrário.

Otimizações efetuadas:

1. Na função **initializeGraph** a cada duas edges inseridas (**s** para **u** e **u** para **t**) foi colocado o valor mínimo entre o peso de cenário e de primeiro plano de cada pixel numa variável (**value**) de modo a poder somar esse valor ao fluxo máximo e poder subtraí-lo à capacidade residual dos arcos correspondentes. Se a capacidade dos arcos, depois desta atualização ficar igual a 0 então esse arco não é inserido no grafo, reduzindo assim os arcos a analisar na BFS. Esta otimização deve-se ao facto de ser sempre garantida a existência de um caminho de tamanho dois que se liga da fonte a cada pixel e desse pixel ao destino.

2. Nas funções **getHorizontalConnections** e **getVerticalConnections** sempre que o valor da relação de vizinhança for igual a 0 então os dois arcos correspondentes não são adicionados ao grafo.

3. À medida que se descobre o caminho de aumento na BFS é guardado um ponteiro na estrutura dos pixéis que dá acesso ao arco do antecessor que liga ao pixel. Deste modo é possível aceder às capacidades residuais de cada arco em tempo constante.

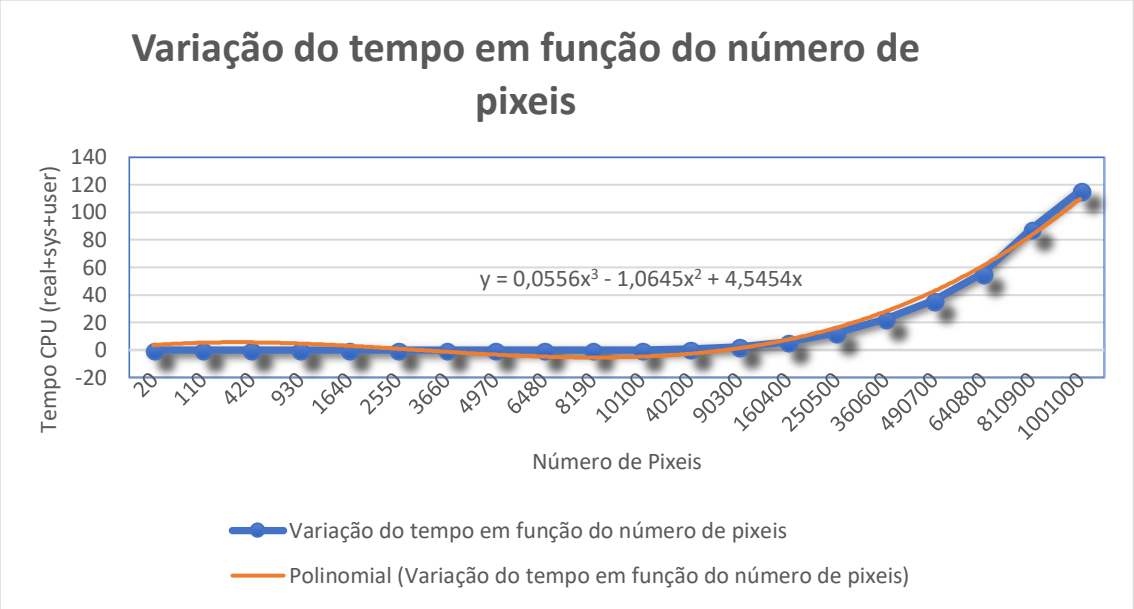
Análise teórica

Foi escolhida como estrutura de dados para a implementação deste problema um grafo representado como listas de adjacências dado que o grafo

que modela o problema é esparso. Sendo **V** o número de vértices e **E** o número de arcos num grafo, a complexidade associada à inicialização de um grafo sem ligações é de **O (V)** e à posterior inserção de todos os arcos é de **O (E)**.

No algoritmo de **Edmonds-Karp** encontrar um caminho de aumento de s para t tem complexidade O(E) e número de aumentos de fluxo é **O(VE)**. Deste modo a complexidade total do algoritmo é **O(VE^2)**. Devido à otimização 1, o número de arcos deverá ser **5*V** logo a complexidade total do programa será **O(25V^3).**

Análise experimental



O gráfico acima revela a variação do tempo em função do número de pixeis. Como esperado, de acordo com a análise teórica, a complexidade total do programa é aproximadamente, **O(V^3)**, como se pode observar pela expressão da linha de tendência (laranja). O gráfico apresentado acima representa, para este caso em particular, uma função do tipo cúbico, tal como esperado, dado que **E=5V**.

Referências bibliográficas

Introduction to Algorithms, Third Edition: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein September 2009 ISBN-10: 0-262-53305-7; ISBN-13: 978-0-262-53305-8