Processamento de Imagens usando Grafos

Prof. Alexandre Xavier Falcão

Aula 04

1 Variantes e questões de implementação

Nesta seção são apresentados alguns variantes do algoritmo geral e questões de implementação.

- Maximizando o mapa V.

  A maximização do mapa V para uma função como $f_{\min}$, por exemplo, requer a troca de $V(q) \neq +\infty$ por $V(q) \neq -\infty$ nas linhas 3 e 9, $V(q) > V(p)$ na linha 6 por $V(q) < V(p)$ e $tmp < V(q)$ por $tmp > V(q)$ na linha 8.

- Cálculo incremental de f.

  A linha 7 indica que algumas funções de conexidade podem requerer informações de todos os pixels em $P^*(p)$ para calcular $f(P^*(p))$. Na maioria dos casos, porém, o cálculo pode ser incremental. Por exemplo, podemos substituir a linha 7 por $tmp \leftarrow V(p) + w(p,q)$ no caso de f_{sum}.

- Propagação de rótulos das raízes.

  Aplicações como segmentação, por exemplo, buscam gerar uma imagem $\hat{L} = (D_I, L)$ onde cada objeto (incluindo o fundo) é representado por um rótulo (cor) distinto. Se fornecermos então uma função $\lambda(q)$ que associa o rótulo de $q \in S$ do objeto correspondente, podemos propagar este rótulo durante o algoritmo. Isto requer $L(q) \leftarrow \lambda(q)$ na linha 2 e $L(q) \leftarrow L(p)$ na linha 10.

- Fila de prioridade Q.

  A fila Q é a parte mais complicada do algoritmo, pois sua implementação eficiente é fundamental no tempo de execução do método. Uma implementação simples de Q poderia levar ao tempo $O(|D_I|^2)$, mas ela pode ser reduzida para $O(|A| + |D_I| \log |D_I|)$ com um *heap* binário.

  Para o caso especial, onde os incrementos $f(\pi_p \cdot \langle p, q \rangle) - f(\pi_p) \in [0..K]$ são inteiros, porém, aplica-se o variante de Dial para reduzir o tempo de execução para $O(|A| + |D_I|K)$. No entanto, esta redução e a redução logaritmica acima só são válidas em grafos esparsos (i.e., $|A| \ll |D_I|$).

A solução de Dial envolve *bucket sort*, usando um vetor circular com $K+1$ *buckets*, cada um apontando para uma lista duplamente ligada de nós com o mesmo valor em V durante a execução do algoritmo (Figura 1a). Nós q com valores $V(q)$ são inseridos na posição $V(q)\%(K+1)$ neste vetor. Isto garante que qualquer *bucket* $0 \leq i \leq K$ tenha apenas nós com valor $V(q)$ por vez, porque a inserção de um nó com valor $V(q)+(K+1)N$ (para inteiros positivos N) na posição i ocorre apenas quando todos os nós com valor $V(q)$ já foram removidos da posição i na linha 5.

Entretanto, a alocação dinâmica da lista duplamente ligada não é eficiente e uma solução mais rápida é usar um arranjo do tamanho de D_I para armazenar todas as possíveis listas (Figura 1b). A inserção, remoção e atualização da fila levam tempo $O(1)$. Encontrar o próximo nó a ser removido na linha 5 leva tempo $O(K)$. Portanto, o algoritmo executa em tempo $O(|A|+|D_I|K)$ e memória $O(|D_I|+K)$.

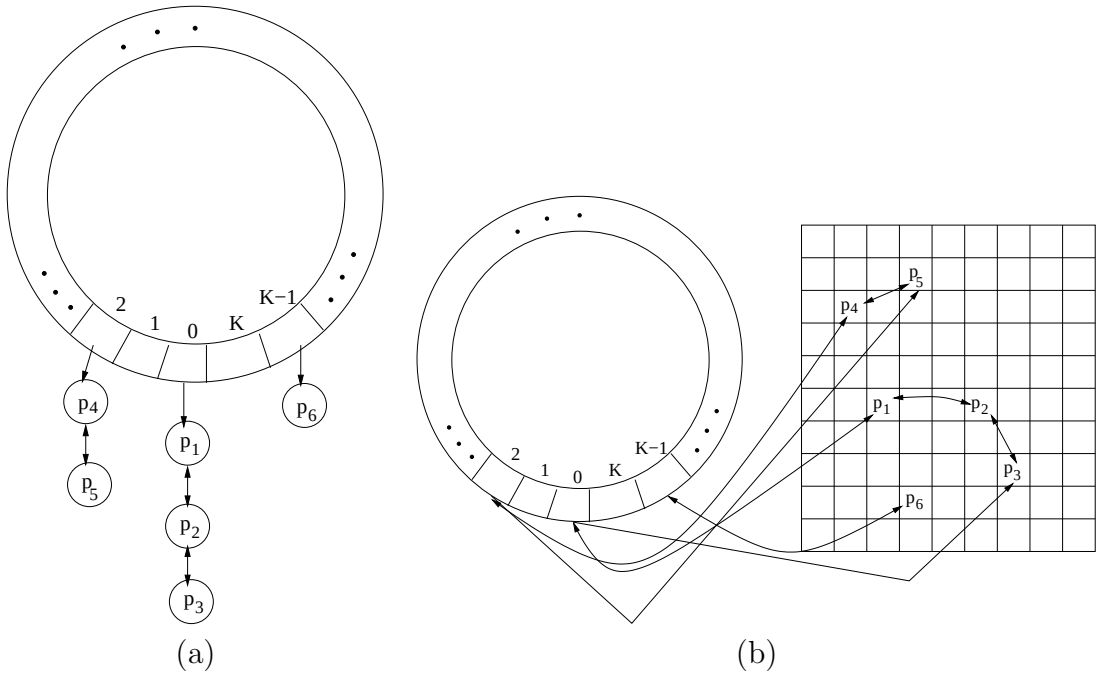

Figura 1: (a) Estrutura de Dial para Q. (b) Estrutura proposta para Q.

- Resolvendo empates na remoção do nó de valor mínimo em Q.

  Quando um nó p é encontrado por mais de um caminho ótimo, ele é normalmente atribuído ao primeiro caminho que o encontrou. Portanto, o algoritmo da IFT apresentado deve também resolver empates na remoção da linha 5, implementando a lista duplamente ligada como uma fila FIFO (*First-In-First-Out*). Neste caso, um componente conexo de mesmo valor de conexidade com várias raízes, tende a ser dividido igualmente entre elas (resultados indesejados, porém, podem ocorrer dependendo de A).

  Outra idéia seria atribuir este componente inteiramente a uma das raízes, por exemplo,

a última raiz que o encontrou. Isto requer um variante do algoritmo geral apresentado abaixo, onde empates para remoção dos nós de cada lista duplamente ligada em Q são resolvidos por LIFO (*Last-In-First-Out*).

Algoritmo da IFT com desempate LIFO:

Entrada: Imagem $\hat{I} = (D_I, I)$, adjacência A, e função f de custo de caminho.
Saída: Imagens $\hat{V} = (D_I, V)$ de conexidade, $\hat{P} = (D_I, P)$ de predecessores, e $\hat{R} = (D_I, R)$ de raízes.
Auxiliares: Fila Q de prioridade, *cor* do nó que indica quando nunca entrou na fila (branco), quando está na fila (cinza) e quando já saiu (preto), e variável *tmp*.

1. Para todo pixel $q \in D_I$ faça
2. $\quad V(q) \leftarrow f(\langle q \rangle)$, $P(q) \leftarrow nil$, $R(q) \leftarrow q$, $cor(q) \leftarrow cinza$ e insira q em Q.
3. Enquanto $Q \neq \emptyset$ faça
4. $\quad$ Remova um pixel p de Q cujo valor $V(p)$ é mínimo e faça $cor(p) \leftarrow preto$.
5. $\quad$ Para todo $q \in A(p)$, tal que $cor(q) = cinza$, faça
6. $\quad\quad tmp \leftarrow f(P^*(p) \cdot \langle p, q \rangle)$.
7. $\quad\quad$ Se $tmp \leq V(q)$ então
8. $\quad\quad\quad$ Remova q de Q, faça $V(q) \leftarrow tmp$, $P(q) \leftarrow p$, e $R(q) \leftarrow R(p)$, e insira q em Q.

A principal diferença entre este último algoritmo e o anterior está nas linhas 7 e 8, pois $P(q)$ e $R(q)$ devem ser atualizados, e q deve ser removido e reinserido em Q, mesmo quando tmp é igual a $V(q)$.

- Funções de conexidade degeneradas

  Algumas funções de conexidade não satisfazem as condições estabelecidas na aula anterior para gerar uma floresta de caminhos ótimos, mas geram uma **floresta espalhada** (mapa de predecessores acíclico) com algumas propriedades interessantes. Um exemplo é a função f_{euc}, que depende da adjacência A para garantir um caminho ótimo entre dois pixels usando o algoritmo com desempate FIFO. Esta função é usada no cálculo de uma transformada de distância Euclideana para o conjunto S. No caso 2D, por exemplo, uma adjacência-8 não garante um mapa Euclideano correto, porém garante que cada a árvode de caminhos enraizados em cada pixel de S é um componente conexo-8. Como veremos, esta propriedade é fundamental para obter esqueletos multi-escala conexos e com um único pixel de largura.

  Um outro exemplo interessante é a função f_{msf},

$$\begin{aligned} f_{msf}(\langle q \rangle) &= \begin{cases} 0 & \text{if } q \in S \\ +\infty & otherwise \end{cases} \\ f_{msf}(\pi_p \cdot \langle p, q \rangle) &= w(p, q) \end{aligned} \tag{1}$$

para pesos fixos de arco e algoritmo com desempate FIFO. Neste caso, a condição $V(q) > V(p)$ na linha 6 não indica que p não possa mudar os atributos de q. É necessário o arranjo *cor* usado no algoritmo com política de desempate LIFO. Este variante requer adicionar $cor(q) \leftarrow branco$ na linha 2 do algoritmo com desempate FIFO, $cor(q) \leftarrow cinza$ na linha 3, $cor(p) \leftarrow preto$ na linha 5, e a troca de $V(q) > V(p)$ por $cor(q) = cinza$ na linha 6. A saída deste variante do algoritmo com política FIFO é, portanto, uma **floresta de peso mínimo** com várias aplicações em análise de dados, onde as orientações dos arcos não fazem sentido neste caso.

A Figura 2 ilustra um exemplo que relaciona uma árvore de peso mínimo e florestas de caminhos ótimos para $f_{\max}$ com restrição de sementes e uma única semente em S. Em uma árvore de peso mínimo, se tomarmos qualquer nó $A \in S$, existe um único caminho conectando A e qualquer outro nó B. Este caminho é ótimo com relação à $_{\max}$ (Figuras 2a-b). Portanto, uma árvore espalhada mínima contém uma árvore de caminhos ótimos para $f_{\max}$ e raiz A. Se nós removermos o arco de maior peso do caminho de A a B, o resultado será uma floresta de peso mínimo que contém uma floresta de caminhos ótimos para $f_{\max}$ e raízes em $S = \{A, B\}$ (Figura 2c).

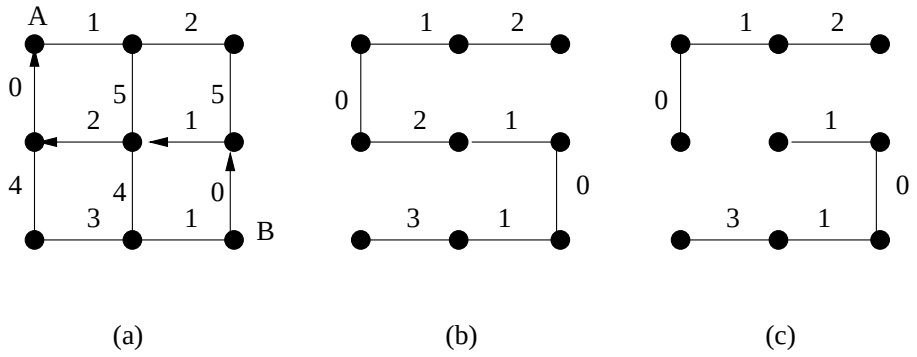

Figura 2: (a) Grafo mostrando caminho ótimo de A a B para $f_{\max}$ com $H(A) = 0$ e $H(q) = +\infty$ para $q \neq A$. (b) Árvore de peso mínimo, que contém caminhos ótimos para $f_{\max}$ entre qualquer par de nós. (c) Floresta de peso mínimo obtida da remoção do arco de maior peso no caminho ótimo de A a B, a qual contém a floresta de caminhos ótimos com raízes em $S = \{A, B\}$