Processamento de Imagens usando Grafos

Prof. Alexandre Xavier Falcão

Segundo semestre de 2004

1 Árvore de componentes

Uma árvore de componentes é uma representação da imagem que descreve relações topológicas entre seus componentes conexos. Dada uma imagem cinza $\hat{I} = (D_I, I)$ e um limiar t, considere o conjunto X^t de todos os componentes conexos da imagem binária $\hat{B}^t = (D_I, B^t)$, onde $B^t(p) = 1$ se $I(p) \geq t$ e $B^t(p) = 0$ no caso contrário, para todo $p \in D_I$. Os conjuntos X^t, $t = 0, 1, 2, \ldots, I_{\max}$, formam uma **decomposição por limiarização** de $\hat{I}$, onde $I_{\max}$ é o maior brilho de pixel na imagem. Analisando níveis consecutivos desta decomposição, podemos perceber que alguns componentes do nível $t+1$ estão contidos em componentes do nível t. Uma árvore de componentes é, portanto, um grafo que armazena esta hierarquia entre componentes. Os nós da árvore são conjuntos de pixels de mesmo brilho (**mas não necessariamente conexos**). Cada nó tem um único pai, exceto o nó raiz, que corresponde ao conjunto de pixels com valor 0 (ou valor mínimo na imagem). Um componente conexo $X_k^t \subset X^t$ é formado pela união do nó k no nível t com os nós descendentes de k nos níveis subseqüentes. Esta árvore é conhecida como *max-tree*, pois as folhas são **máximos regionais** da imagem— isto é, componentes conexos de mesmo brilho e tamanho máximo (*flat zones*), cujo valor é estritamente maior que o das *flat zones* vizinhas. Similarmente, podemos construir a árvore *min-tree*, cujas folhas são **mínimos regionais**, calculando os componentes conexos X^t, $t = I_{\max}, I_{\max} - 1, \ldots, 0$, de $\hat{B}^t = (D_I, B^t)$, onde $B^t(p) = 1$ se $I(p) \leq t$ e $B^t(p) = 0$ no caso contrário, para todo $p \in D_I$.

Observe, no caso da *min-tree*, que um componente conexo no nível t é na verdade um componente κ-conexo na imagem $\hat{I}$ com relação à função de custo f_{peak} e $\kappa = t$. O equivalente pode ser observado para a *max-tree*, tomando-se a definição complementar de κ-conexidade.

1.1 Aplicações

As árvores de componentes, *min-tree* e *max-tree*, possibilitam diversos tipos de operadores de imagem, com aplicações em filtragem e segmentação. Os operadores de filtragem baseados nessas árvores são conhecidos como **filtros conexos**, pois pares de pixels que pertencem a uma mesma *flat zone* antes da filtragem, continuam pertencendo a uma mesma *flat zone* após filtragem. Exemplos são filtros de fechamento/abertura por reconstrução morfológica. Muito embora a reconstrução morfológica possa ser calculada de forma eficiente pela IFT, certos filtros

requerem variantes da IFT não tão intuitivos. Um exemplo é a filtragem por abertura baseada em área (*area opening*). Após calcular a *max-tree*, este filtro pode ser implementado removendo da árvore os nós de maior brilho com área menor ou igual a um dado limiar. Isto equivale a gerar uma imagem simplificada, onde os pixels que pertencem a esses nós têm associado o brilho do nó ascendente cuja a área é maior que o limiar, e os demais pixels permanecem com o brilho da imagem original. Se ao invés de área, levarmos em conta o volume dos domos da imagem, teremos um *volume opening*, e similarmente podemos implementar *area closing* e *volume closing*. Podemos também selecionar sementes e atribuir regras de poda para ramos da árvore a fim de gerar filtragens e segmentações.

1.2 Algoritmo

Para construir uma árvore de componentes, precisamos identificar os pixels que pertencem a cada nó X_k^t da árvore e a relação de parentesco entre esses nós. Outra informação importante é o mapa de componentes, que indica para cada pixel $p \in D_I$, qual nó ele pertence na árvore. Para construir uma *max-tree*, a idéia é fazer um percurso na imagem que parte das *flat zones* de maior brilho (folhas da árvore) para as de menor brilho (raiz), identificando o pai no nível $t' < t$ de uma *flat zone* do nível t apenas quando estamos visitando o pai. Note que a construção da relação de parentesco ocorre naturalmente durante o percurso, assim que um pixel da *flat zone* do nível t' tiver como vizinho um pixel da *flat zone* do nível t. Para representar cada *flat zone* por um único pixel, vamos adotar um percurso em profundidade nas *flat zones* de cada nível seguindo a prioridade $t = I_{\max}, I_{\max} - 1, \ldots, 0$. Isto elimina a necessidade de tratarmos a *flat zone* como conjuntos disjuntos que devem ser unidos durante o percurso.

Note também que um nó da árvore no nível t pode conter várias *flat zones*, desconexas neste nível, mas conexas em X^t. Ou seja, durante o percurso, duas $f$lat zones de mesmo nível t vão pertencer a um mesmo nó sempre que existirem *flat zones* de níveis maiores que t ligando elas. Esta informação é descoberta através da relação de parentesco. Isto é, se um pixel de uma *flat zone* x no nível t encontra um adjacente em uma *flat zone* y de nível maior que t, três situações podem ocorrer:

1. y ainda não tem pai, portanto x é o pai de y, ou
2. y tem pai e o brilho da raiz da subárvore que y pertence é maior que o brilho de x, portanto x é o pai desta raiz, ou
3. y tem pai e o brilho da raiz da subárvore que y pertence é igual ao brilho de x, portanto x e a raiz pertencem ao mesmo nó.

Neste sentido, como as *flat zones* de um mesmo nó são percorridas separadamente, vamos tratá-las como conjuntos disjuntos que devem ser unidos assim que descobrimos que elas pertencem a um mesmo nó. O algoritmo abaixo ilustra o processo de construir a relação de parentesco em um mapa P e os nós da max-tree em um mapa R. A árvore pode ser facilmente gerada a partir de P e R.

Algoritmo para construção de uma max-tree:

Entrada: Imagem $\hat{I} = (D_I, I)$ e adjacência A.
Saída: Imagens $\hat{P} = (D_I, P)$ de parentesco entre nós e $\hat{R} = (D_I, R)$ de representantes dos nós.
Auxiliares: Fila Q de prioridade com política LIFO e imagem $\hat{N} = (D_I, N)$ do número de elementos por nó.

1. Para todo pixel $p \in D_I$, faça $P(p) \leftarrow nil$, $R(p) \leftarrow p$, $N(p) \leftarrow 1$, e insira p em Q.
2. Enquanto $Q \neq \emptyset$, faça
3. Remova p de Q tal que $I(p)$ seja máxima.
4. $r_p \leftarrow Representante(\hat{R}, p)$.
5. Para todo pixel $q \in A(p)$, faça
6. Se $I(p) = I(q)$, então
7. Se $q \in Q$, então
8. Remova q de Q, $R(q) \leftarrow r_p$, e insira q em Q.
9. $N(r_p) \leftarrow N(r_p) + 1$.
10. Se não,
11. Se $I(p) < I(q)$, então
12. $r_q \leftarrow Representante(\hat{R}, q)$.
13. $r \leftarrow RaizAtual(\hat{P}, \hat{R}, r_q)$.
14. Se $r = nil$, então $P(r_q) \leftarrow r_p$.
15. Se não,
16. Se $I(r) = I(r_p)$, então
17. Se $r \neq r_p$, então $Junte(\hat{R}, \hat{N}, r, r_p)$.
18. Se não, $P(r) \leftarrow r_p$.
19. Para todo pixel $p \in D_I$ faça $R(p) \leftarrow Representante(\hat{R}, p)$.

Algoritmo para encontrar o representante com compressão:

$Representante(\hat{R}, p)$

1. Se $R(p) = p$, então
2. retorne p.
3. Caso contrário,
4. retorne $R(p) \leftarrow Representante(\hat{R}, R(p))$.

Algoritmo para unir componentes de mesmo nível:
$Junte(\hat{R}, \hat{N}, r, r_p)$

1. Se $N(r_p) \leq N(r)$, então $R(r_p) \leftarrow r$, $N(r) \leftarrow N(r) + N(r_p)$, e $r_p \leftarrow r$.
2. Se não, $R(r) \leftarrow r_p$ e $N(r_p) \leftarrow N(r_p) + N(r)$.

Observe que r_p deve ser passado por referência.

Algoritmo para encontra a raiz da subárvore atual que contém o componente:
$RaizAtual(\hat{P}, \hat{R}, r_q)$

1. $r_1 \leftarrow r_2 \leftarrow P(r_q)$
2. Enquanto $r_2 \neq nil$, faça
3. $r_1 \leftarrow r_2 \leftarrow Representante(\hat{R}, r_2)$.
4. $r_2 \leftarrow P(r_2)$.
5. Retorne r_1.

Note que o número de nós da *max-tree* é o número de representantes em $\hat{R}$ (i.e. pixels p tais que $R(p) = p$). A fila de prioridades com desempate LIFO garante que cada componente conexo de mesmo brilho seja inteiramente visitado de forma independente do outro. Portanto, a união entre eles só ocorrerá se eles se conectarem por componentes de brilho maior (menor) na max-tree (min-tree). Após criar a árvore, o mapa de representantes pode ser facilmente transformado em um mapa de componentes, onde cada pixel p tem $R(p)$ igual ao endereço do nó correspondente na *max-tree*. O mapa P guarda a relação de pai de cada nó no pixel representante do nó. Portanto, pode ser usado para encontrar a raiz da árvore, o pai de cada nó, e os filhos de cada nó. Além dessas informações, o nível de cinza de cada nó e o número de pixels do nó podem ser armazenados na árvore.

2 Exercício

Implemente a *max-tree* e a *min-tree*, os filtros discutidos acima, e mais alguns outros que você inventar.