ORIENTAÇÕES PARA A AVALIAÇÃO PRÁTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

MATÉRIA: ENGENHARIA DE PROGRAMAS

Professor: Diego Frias
Semestre: 2021.2

1. Introdução:

A atividade prática tem como finalidades:

I. Consolidar os conhecimentos sobre:
   1. A dependência hierárquica 1:n nos níveis problema-algoritmo e algoritmo-código fonte
   2. A técnica analítica vista no curso para quantificação do número de operações (a,o,c) realizadas por um código fonte, em função do "tamanho" dos dados e dos valores dos "parâmetros" (configuração paramétrica do método).
   3. A técnica empírica (experimental) vista no curso para quantificação do tempo de execução de um código fonte, em função do "tamanho" dos dados e dos valores dos "parâmetros" (configuração paramétrica do método).

II. Desenvolver abilidades para:
   1. Obter a função vetorial "número de operações" do tamanho dos dados (n) e da configuração paramétrica do método (K – vetor de parâmetros distintos).
   2. Obter o "modelo analítico" do "custo (computacional)" do código fonte como a norma L1 (soma das componentes) da função vetorial do número de operações.
   3. Construir e validar setup experimental para a medição confiável (estatisticamente fundamentada) do tempo de execução de código fonte com determinados valores de n e K.
   4. Planejar um programa de experimentos com vistas a medir o tempo médio e o desvio padrão do tempo de execução do código fonte estudado com diferentes combinações de n x K, que chamamos estados de teste (E). O conjunto de todos os estados de teste é chamado de Domínio do Teste (D).
   5. Realizar a medição do tempo médio e do desvio padrão do código fonte em D, obtendo os "resultados experimentais".
   6. Visualizar os resultados experimentais:
      i. Tempos médios (+/- desvio padrão) vs n para diferentes K
      ii. Tempos médios (+/- desvio padrão) vs K para diferentes n
   7. Avaliar o modelo analítico (ponto 1) no domínio de teste D, obtendo os "resultados teóricos"
   8. Visualizar os resultados teóricos:
      i. custo computacional vs n para diferentes K
      ii. custo computacional vs K para diferentes n
   9. Comparar "qualitativamente" os resultados teóricos e experimentais (plots 6.i vs 8.i e plots 6.ii vs 8.ii)
   10. Comparar "quantitativamente" os resultados teóricos e experimentais plotando os custos vs tempos médios em D
   11. Obter a função linear de regressão tempo médio vs custo: tempo = a + b * custo
   12. Calcular o número de operações por segundo (FLOPS) do sistema computacional utilizado no estudo: FLOPS = 1/b, se e somente se 'a' (o intercepto) tende a zero, a<< 1 segundo.

2. Problema abordado:

Neste semestre o problema abordado é a classificação não supervisionada de imagens (segmentação).
Os algoritmos selecionados são:

I. K-means
II. Método de limiar de OTSU

Dataset: 20 imagens de rosto de diferentes resoluções + 1 imagem de calibração

OBSERVAÇÃO: É importante notar que:

- Nosso objetivo não é resolver o problema da melhor forma possível, o que é objetivo de outra

matéria (Inteligência Artificial). Aqui estamos interessados na caracterização do custo computacional de 1 código fonte que implementa 1 algoritmo de solução de 1 problema de IA.

- Contudo, como é necessário garantir que o código fonte resolve de forma correta o problema, é necessário "verificar" a corretude do código fonte. Como o problema é a segmentação de imagens, uma avaliação qualitativa da corretude do código consiste em plotar a imagem de calibração junto com as imagens segmentadas com 2 e 3 clusters.

3. Descrição geral dos métodos (algoritmos):

A maioria dos algoritmos de aprendizado de maquina possuem 3 fases consecutivas: (1) **Inicialização:** atribuição de valor às variáveis que irão mudar no processo iterativo,
(2) **Processo iterativo:** mudança de valor das variáveis do modelo, e
(3) **Finalização:** interrupção do processo iterativo quando determinada "condição" de parada é satisfeita. As "condições de parada" podem ser "positivas" quando sinalizam o sucesso da tarefa executada, ou "negativas" quando indicam falha do processo iterativo, por exemplo: número excesivo de iterações, divergência no tempo do critério de parada esperado a convergir para determinado valor, etc.

4. Análise Quantitativa do Custo Computacional:

Devido à estrutura dos algoritmos estudados, a fórmula geral do número de operações tem a forma

$$N(n,K,m) = N_{ini}(n,K) + m * N_{iter}(n,K) \qquad \text{(fórmula geral do método)}$$

onde:

- $N_{ini}(n,K)$ é o número de operações no bloco de inicialização, que geralmente depende do parâmetro "K" e pode também depender do tamanho dos dados "n"
- $N_{iter}(n,K)$ é o número de operações realizadas em 1 iteração completa, que geralmente depende do tamanho dos dados "n" e do parâmetro "K"
- "m" é o número de iterações

Os termos N_{init} e N_{iter} obedecem à fórmula geral estudada

$$N(n,K) = g1 + X * (gr + ex) \qquad \text{(fórmula geral dos blocos repetitivos)}$$

onde:

- g1 são as operação de gestão do laço que se realizam uma única vez,
- gr são as operações repetitivas na gestão do laço,
- ex são as operações que se executam dentro do laço, e
- X são as vezes que o laço se repete.

5. Descrição dos dados:

Cada instância de dado é uma imagem colorida (RGB) com L x A (largura x altura) pixels de cor c = [R,G,B], onde a intensidade de cada cor varia no intervalo 0:255.

Cada imagem é então considerada uma coleção de n=A*L pontos num espaço de 3 dimensões.

A distância a ser usada é a Euclideana (norma L2) por padrão, ou seja, dados 2 pontos $\boldsymbol{p}=[R_p,G_p,B_p]$ e $\boldsymbol{q}=[R_q,G_q,B_q]$ a distância (contraste) "entre as cores" deles vem dada por

$$d(\boldsymbol{p},\boldsymbol{q})=\text{sqrt}((R_p-R_q)^2+(G_p-G_q)^2+(B_p-B_q)^2)$$

Importante lembrar que os pontos $\boldsymbol{p}$ e $\boldsymbol{q}$, por se tratar de uma imagem 2D, são "localizados" com duas coordenadas, ou seja a "posição" do ponto $\boldsymbol{p}$ é um vetor bi-dimensional, que vamos denotar por $\boldsymbol{P_p}=[x_p,y_p]$, para $x_p=0,1,...,L-1$ e $y_p=0,1,...,A-1$ (IDEM para $\boldsymbol{P_q}$). Por isso a "distância geométrica" entre 2 pontos $\boldsymbol{P_p}$ e $\boldsymbol{P_q}$, vem dada por

$$D(\boldsymbol{P_p},\boldsymbol{P_q})=\text{sqrt}((x_p-x_q)^2+(y_p-y_q)^2)$$

No processo de segmentação usa-se apenas o contraste ou a distância $d(\boldsymbol{p},\boldsymbol{q})$ entre as cores. A distância geométrica $D(\boldsymbol{P_q},\boldsymbol{P_q})$ foi introduzida apenas para esclarecer o fato que o conceito de distância usado em segmentação não tem nada a ver com a distância entre 2 pixels na imagem.

Como a distância geométrica não tem utilidade neste tipo de problema (nos métodos desenvolviddos até agora) é comum transformar as imagens 2D lidas em vetores 1D usando uma função flatten(). Nós vamos "denotar" a posição dos pontos no vetor 1D obtido via flatten() de uma imagem

2D, por P_i = 0,1,...,n-1, onde n=A*L (definido anteriormente) e a cor do ponto na posição P_i por i=[R_i,G_i,B_i]. Então, a distância em cor entre 2 pontos i e j vem dada por

$$d(i,j)=\text{sqrt}((R_i-R_j)^2+(G_i-G_j)^2+(B_i-B_j)^2)$$

para i, j variando entre 0 e n-1.

5.1. Transformando imagens coloridas (RGB) em escala de cinza

O método OTSU trabalha com imagens preto-e-branco, nas quais os pixels tem um único valor associado com a "intensidade" da cor. Ou seja em vez da cor do pixels vir definida por 3 valores, é definida por um único valor entre 0 (preto) e 255 (branco), o que estabelece uma escala de cores cinza. Existem diversas formas de transformar imagens coloridas em escala de cinza respeitando o conceito perceptivo/sensorial de luminância (para mais detalhes consulte https://en.wikipedia.org/wiki/Grayscale). Aqui vamos usar os pesos [0.2989,0.5870,0.1140] para os canais R,G,B de forma que a cor C na escala de cinza de um pixel com valores RGB = [R,G,B] vem dado por:

$$C = \text{int}(0.2989*R+0.5870*G+0.1140*B) \qquad \text{(Eq. transformação)}$$

As equipes que estudarão o método OTSU precisam transformar as imagens RGB em imagens monocromáticas (escala de cinza) calculando a cor C (que varia no intervalo de 0 a 255) para cada pixel da imagem sendo segmentada. Este preprocessamento faz parte do método, pelo que as operações envolvidas precisam ser levadas em conta no estudo analítico do custo computacional da inicialização.

6. Metodologia:

A atividade prática consta das seguintes 9 fases de trabalho, sendo a última a de apresentação de resultados:

I. Construção de programa para a 'leitura' do dataset e 'processamento' do mesmo
1. Leitura de um número dado de imagens do dataset
2. Construção do código fonte de processamento com alguma das alternativas seguintes:
   i. Programar o algoritmo a partir de modelo analítico-matemático-descritivo em artigo/página de referência fornecido.
   ii. Montar código usando código-fonte de bibliotecas abertas (substituindo as chamadas a funções pelo código da função chamada).

II. Obtenção da função vetorial "número de operações" do tamanho dos dados (n) e da configuração paramétrica do método (K – vetor de parâmetros distintos) do código fonte de processamento (ponto b). Obtenção do "modelo analítico" do "custo (computacional)" do código fonte como a norma L1 (soma das componentes) da função vetorial do número de operações.

III. Construção e validação de setup experimental para a medição confiável (estatisticamente fundamentada) do tempo de execução de código fonte com determinados valores de n e K. A validação é feita executando 100 vezes o código fonte para uma única combinação de n e K[1], medindo o tempo de execução em cada rodada. Os tempos medidos nas 100 rodadas são usados para calcular o tempo médio 'Tm' e o desvio padrão dele 'dT', para calcular o índice de estabilidade 'Ixe' dado por Ixe = dT/Tm que deve ser menor que 0.15. Em caso contrário, será preciso usar um servidor virtual simples ou desligar serviços (internet, bluetooth, aplicativos de monitoração do computador), etc.

IV. Planejamento do programa de experimentos com vistas a medir o tempo médio e o desvio padrão do tempo de execução do código fonte estudado com diferentes combinações de n x K, que chamamos estados de teste (E). O conjunto de todos os estados de teste é chamado de Domínio do Teste (D). Definir o número de repetições.

V. Avaliação do modelo analítico (ponto 1) no domínio de teste D, obtendo os "resultados teóricos" (usando excel). Visualização dos resultados teóricos:
   i. custo computacional de inicialização "C_{ini}" vs n para diferentes K
   ii. custo computacional de inicialização "C_{ini}" vs K para diferentes n
   iii. custo computacional por iteração "C_{iter}" vs n para diferentes K
   iv. custo computacional por iteração "C_{iter}" vs K para diferentes n

VI. Medição do tempo médio e do desvio padrão do código fonte em D, obtendo os "resultados

1. No caso de processamento de imagens usamos uma imagem de teste com K=3.

experimentais" ("tempo de inicialização - T_{ini}" e "tempo por iteração - T_{iter}" em cada experimento). Visualização dos resultados médios experimentais:

i. Tempos médios (+/- desvio padrão) vs n para diferentes K
ii. Tempos médios (+/- desvio padrão) vs K para diferentes n
iii. Número de iterações (+/- desvio padrão) vs n para diferentes K

VII. Análise de Resultados:

1. Comparação "qualitativa" dos resultados teóricos e experimentais (plots 6.i vs 8.i e plots 6.ii vs 8.ii)
2. Comparação "quantitativa" dos resultados teóricos e experimentais plotando os custos vs tempos médios em D.

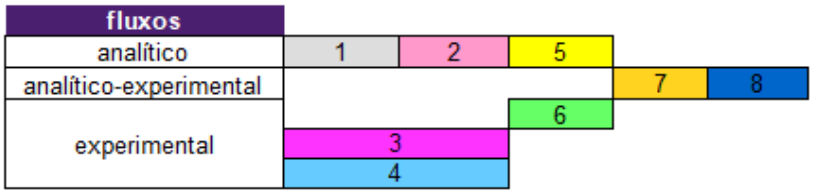
VIII. Obtenção da função linear de regressão tempo médio vs custo: tempo=a+b*custo. Calculo do número de operações por segundo (FLOPS) do sistema computacional utilizado no estudo: FLOPS = 1/b, se e somente se 'a' (o intercepto) tende a zero, a<< 1 segundo.

IX. Elaboração de relatório com os resultados de todos os passos acima. Gravação e envio de vídeo com a apresentação do relatório acima pela equipe (balanceando a participação dos membros)

7. Fluxo do projeto

Levando em conta a natureza (analítica ou experimental) das fases da atividade e a precedência obrigatória entre elas, se definem 2 fluxos paralelos que conduzem a um fluxo final analítico-experimental conforme mostrado na figura a seguir:

fase número	descrição	depois de
1	codificação	-
2	fórmula custo analítico	1
3	setup experimental	-
4	plano experimental	-
5	cálculo do custo	2
6	medição do tempo	2 & 3 & 4
7	análise de resultados	5 & 6
8	cálculo FLOPS	7



8. Resultados Esperados (no relatório e na apresentação) – Entregáveis:

1. Código fonte executando corretamente em ambiente computacional comprovadamente estável (fases 1 e 3)
2. Domínio de Teste configurado definindo malha parâmetrica (valores de n vs de K) (fase 4)
3. Fórmula do custo obtida (fase 2)
4. Custo computacional calculado (inicialização e por iteração) na malha paramétrica definida pelo Domínio de Teste (tabela) (fase 5)
5. Tempos de execução (inicialização e por iteração) medidos 20 vezes em cada ponto da malha paramétrica definida pelo Domínio de Teste (tabela) (fase 6 primeira parte)
6. Tempos de execução (inicialização e por iteração) médios e desvio padrão calculados em cada ponto da malha (tabela). (fase 6)
7. Gráficos da dependência do custo (inicialização e por iteração) e dos tempos médios de inicialização e por iteração em função de n e de K. (fase 7 primeira parte)
8. Texto com análise comparativa das dependências do custo e do tempo com n e K: Quão similares ?

Quais são as diferenças ? (fase 7)

9. Gráfico Tempo vs Custo com linha de regressão y=a+b*x, mostrando o intercepto “a” e o slope “b”. (pode ser em excel) (fase 8 primeira parte)
10. FLOPS calculado. (fase 8)

3. Avaliação:

Cada fase do projeto descrita na seção de Metodologia terá a seguinte pontuação:

fase número	descrição	nota MÁXIMA
1	codificação	1.00
2	fórmula custo analítico	2.00
3	setup experimental	0.50
4	plano experimental	0.50
5	cálculo do custo	2.00
6	medição do tempo	2.00
7	análise de resultados	1.50
8	cálculo FLOPS	0.50
	TOTAL	10.00

A Nota do trabalho prático acima terá peso 0.7 enquanto a nota do relatório escrito + apresentação (vídeo) terá peso 0.3 na nota final do crédito avaliado.

4. Prazo:

Datas de entregas parciais e final:

- 31-10-2021 – fases 1, 3 e 4 (codificação, setup e plano experimental)
- 07-11-2021 – fase 2 (fórmula do custo)
- 14-11-2021 – fases 5 e 6 (cálculo do custo e medições de tempo)
- 21-11-2021 – fases 7 e 8 (análise de resultados e cálculo dos FLOPS)
- 28-11-2021 – relatório e vídeo da apresentação (até as 23:59). Atrasso implica em redução da nota em 0.2 por dia.

BOM TRABALHO!

ANEXO – DESCRIÇÃO DOS MÉTODOS DE SEGMENTAÇÃO DE IMAGENS

1. K-MEANS

Assume-se que estamos processando imagens mapeadas em 1D após sua leitura no arquivo de entrada (este preprocessamento não se inclui no cálculo nem na medição)!

Parâmetros: K – número de cluster a serem formados

Dimensionador: n = A*L número de pontos na imagem de resolução AxL

- ***Inicialização (ingênua)***: Selecione aleatoriamente K pontos como centroides dos clusters: $\mathbf{C}_k$, k=1,2,...,**K** (sendo $\mathbf{C}_k$ a cor do centroide k-ésimo). O centroide k-ésimo está na posição **pk** , k=1,2,...,**K** do vetor de pontos $\mathbf{P}_i$, i =0,1,...,n-1.

```
% pseudo-código
init_time()
for k=1:K {
    % seleciona um ponto aleatório como centroide do cluster k-ésimo
     pk=int(n*rand()) % se a função rand() retorna um número entre 0 e 1
     Rc[k]=R[pk]
     Gc[k]=G[pk]
     Bc[k]=B[pk]
}
tempo de inicialização = get_time()
```

- ***Processo Iterativo e Finalização:*** Agrupe os pontos mais próximos a cada um dos K centroides, atribuindo um cluster (classe) a cada ponto na imagem. Monitore se a classe atribuida a algum ponto é diferente da classe atribuída anteriomente. Se não houve mudança de atribuição finalize as iterações, mas, em caso contrário, atualize a cor dos K centroides e inicie uma nova iteração.

```
% pseudo-código:
init_time()
class[0:n-1]=-1 % inicialização – todos os pontos são atribuído a uma classe
        inexistente (-1)
iter=0
repeat while true: { % gerencie iterações
     iter+=1
     change=0 % -> set flag of change to zero
     for i=0:n-1 { % varre lista de pontos
          min_dist=1e10
          for k=0:K-1 {  % calcula a distância do ponto i a cada cluster K
               d=sqrt((R[i]-Rc[k] )^2+(G[i]-Gc[k] )^2+(B[i]-Bc[k] )^2)
               if d<min_dist { % registra a menor distância
                    min_dist=d
                    if change==0 & class[i]<>k { % notifico que atribuição mudou
                         change=1
                    }
                    % atualiza a classe do ponto i
                    class[i]=k
               }
          } % fim do laço para achar o cluster mais próximo ao ponto
    } % fim lo laço pelos pontos da imagem
     if change==0 { % se não houve mudança de atribuição finaliza
          print('convergencia obtida')
             break % finaliza o repeat
    }
```

```
        else if change>0 { % se houve mudança atualize os centroides dos clusters
            for k=0:K-1 {
                  Rc[k]=0
                  Gc[k]=0
                Bc[k]=0
                  m[k]=0 % número do peontos no cluster k
                  for i=0:n-1 {
                   if class[i]==k {
                              Rc[k]+=R[i]
                              Gc[k]+=G[i]
                              Bc[k]+=B[i]
                              m[k]+=1
                    }
                 }
                    % divido entre m para achar a média
                 Rc[k]/=m[k]
                 Gc[k]/=m[k]
               Bc[k]/=m[k]
              }
        }
}  % fim das iterações
time=get_time()
tempo_por_iteração = time/iter
 % output:  tempo_por_iteração e iter
```

2. Método baseado em limiar de OTSU

Processando imagens mapeadas em 1D após sua leitura no arquivo de entrada. Considere a imagem 1D colorida dada pelos vetores R[i], G[i] e B[i], i=0,2,...,n-1, com as intensidades das 3 cores básicas, vermelho, verde e azul, em cada pixel da imagem.

Parâmetros: K – número de cluster a serem formados

Dimensionadores:

1. **n = A*L número de pontos na imagem de resolução AxL**
2. **M = 256 intensidade máxima de uma cor**

- ***Inicialização***: Consiste em:

1. Transformação da imagem colorida em preto-e-branco como descrito acima, atribuindo a cada pixel da imagem uma cor entre 0 e 255.

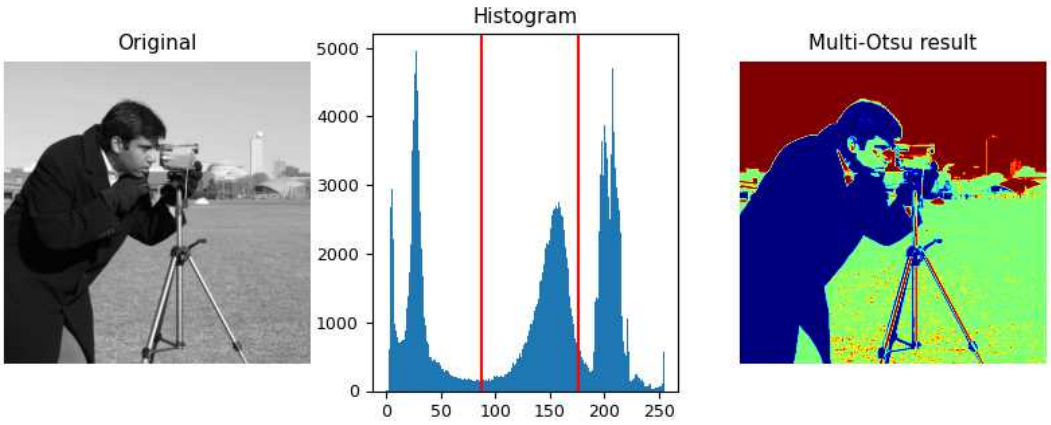
```
% pseudo-código imagem colorida para tons de cinza
init_time()
for i=0:n-1 { % varre lista de pontos
    C[i]=int(0.2989*R[i]+0.5870*G[i]+0.1140*B[i]) % os tres vetores R,G,B são condensados
                                                    no vetor C
}
```

2. Calcular o número de pixels de cada cor de 0 a 255 (histograma p)

```
% pseudo-código para cálculo do histograma
for j=0:M-1 { p[j]=0 } % inicializando com 0 todas as cores
for i=0:n-1 { % varrendo os pixels contando o numero de cada cor
   p[C[i]]+=1
}
for j=0:M-1 { p[j]=p[j]/n } % normalizando o histograma p
```

tempo de incialização = get_time() % medição do tempo de inicialização

- ***Processo iterativo***: Consiste em dividir as M cores do histograma em K regiões consecutivas, uma para cada classe. Isto implica na necessidade de achar K-1 fronteiras $f_0 < f_1 <...< f_{K-1}$ (cores) entre as K classes, usando como critério maximizar a variança interclasses (veja descrição do modelo matemático embaixo do código).
  Como temos M cores c = 0,1,...,M-1 e as fronteiras entre classes são ordenadas e não sobrepostas, se diferenciam a primeira e a última região por ter uma das fronteiras fixas: a cor 0 a região inicial e a cor M-1 (255) a última região. As outras regiões intermediárias têm as duas fronteiras variáveis. Ou seja, temos
  - 1 região inicial da cor 0 à f_0,
  - M-2 regiões internas da core f_k à f_{k+1}, para k=0,1,K-2 e
  - 1 região final da cor f_{K-1} à M-1.



O método avalia a qualidade de todas as divisões possíveis (posições das fronteiras entre as classes - regiões), para selecionar a de qualidade máxima. A medida de qualidade Q vem dada pela fórmula

$$Q = \sum_{k=1}^{K} P_k (m_k - m_G)^2,$$

onde Pk é a probabilidade do cluster C_k dada por

$$P_k = \sum_{i \in C_k} p_i$$

e m_k é a cor do centroide do cluster C_k, dada por

$$m_G = \sum_{k=0}^{K-1} P_k \cdot m_k$$

com m_G sendo a cor média dos pontos, dada por

% pseudo-código para o cálculo de Q

```
function Q = quality(f, K, M, p) {
      for k=1:K {
         m[k]=0 % cor inicial dos clusters
         P[k]=0 % probabilidade dos clusters
      }
      # primeiro cluster
      a=f[0]+1
      for c =0:a { % se não for Python a-1
          P[0]+=p[c]
          m[0]+=c*p[c]
      }
      # ultimo cluster
      b=f[K-2]+1
      for c=[b,M] { % se não for Python M-1
          P[K-1]+=p[c]
          m[K-1]+=c*p[c]
       }
      % clusters intermediarios
      if K>2 {
```

```
            d=K-1
            for k =1:d { % se não for Python d-1
                a=f[k-1]+1
                b=f[k]+1
                for c=a:b { % se não for Python b-1
                    P[k]+=p[c]
                    m[k]+=c*p[c]
                }
            }
        }
        mG=0
        for k=0:K { % se não for Python K-1
            mG+=P[k]*m[k]
        }
        Q=0
        for k=0:K {
            Q+=P[k]*(m[k]-mG)^2
        }
    } % end function
```

A base do método consiste em um algoritmo que percorre "todas" as divisões possíveis do histograma levando em conta a ordem necessária das fronteiras. Os métodos que exploram todas as alternativas possíveis:

1. São chamados de métodos de busca exhaustiva
2. Garantem encontrar a melhor solução entre todas as possíves (convergência global)
3. Sempre realizam o mesmo número de iterações, independentemente da imagem processada. Este número de iterações pode ser estimado a priori (analiticamente), conhecendo o número de cores (M) e o número de classes (K).

<u>% pseudo-código (não generalizável para qualquer K)</u>

```
init_time()
Qmax=0
iter=0
for k=0:K-1 { fMax[k]=0 } % python limits
if K==2 {
   for f[0]=0:M-K {  % python limits
      iter+=1
      Q=quality(f, K, M, p)
       if Q>Qmax {
         Qmax=Q
         for k=0:K-1 { fMax[k]=f[k] } % python limits
      }
   }
} elsif K==3 {
    for f[0]=0:M-K { % python limits
        for f[1]=f[0]+1:M-(K-1) { % python limits
            iter+=1
            Q=quality(f, K, M, p)
            if Q>Qmax {
               Qmax=Q
               for k=0:K-1 { fMax[k]=f[k] } % python limits
            }
        }
    }
} elsif K==4 {
    for f[0]=0:M-K { % python limits
        for f[1]=f[0]+1:M-(K-1) { % python limits
            for f[2]=f[1]+1:M-(K-2) { % python limits
```

```
                iter+=1
                Q=quality(f, K, M, p)
                if Q>Qmax {
                   Qmax=Q
                   for k=0:K-1 { fMax[k]=f[k] } % python limits
                }
             }
          }
       } % OLHO: estender para K>4 se for preciso
}
time=get_time()
 tempo_por_iteração = time/iter % medição do tempo por iteração
 % output:  tempo_por_iteração e iter
```

10.3.6 Multiple thresholds

Otsu's method extended to K classes, $C_1, C_2, \ldots, C_K$

Between-class variance:

$$\sigma_B^2 = \sum_{k=1}^{K} P_k(m_k - m_G)^2, \quad \textbf{where} \quad P_k = \sum_{i \in C_k} p_i \quad \textbf{and} \quad m_k = \sum_{i \in C_k} i p_i$$

The K classes are separated by $K-1$ thresholds whose values $k_1^*, k_2^*, \ldots k_{K-1}^*$ maximize

$$\sigma_B^2(k_1^*, k_2^*, \ldots, k_{K-1}^*) = \max_{0<k_1<k_2<\ldots<k_{K-1}<L-1} \sigma_B^2(k_1, k_2, \ldots, k_{K-1})$$

Otsu's method extended to three classes, C_1, C_2, C_3

Between-class variance:

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 + P_3(m_3 - m_G)^2$$

$$P_1 = \sum_{i=0}^{k_1} p_i, \quad P_2 = \sum_{i=k_1+1}^{k_2} p_i, \quad P_3 = \sum_{i=k_2+1}^{L-1} p_i$$

$$m_1 = \frac{1}{P_1}\sum_{i=0}^{k_1} ip_i, \quad m_2 = \frac{1}{P_2}\sum_{i=k_1+1}^{k_2} ip_i, \quad m_3 = \frac{1}{P_3}\sum_{i=k_2+1}^{L-1} ip_i$$

The following relationships also hold:

$$P_1 m_1 + P_2 m_2 + P_3 m_3 = m_G, \qquad P_1 + P_2 + P_3 = 1$$

The three classes are separated by two thresholds whose values k_1^* and k_2^* maximize

$$\sigma_B^2(k_1^*, k_2^*) = \max_{0<k_1<k_2<L-1} \sigma_B^2(k_1, k_2)$$

Algorithm

(1) **Let** $k_1 = 1$

(2) **Increment** k_2 **through all its values greater than** k_1 **and less than** $L-1$

(3) **Increment** k_1 **to its next value and increment** k_2 **through all its values greater than** k_1 **and less than** $L-1$

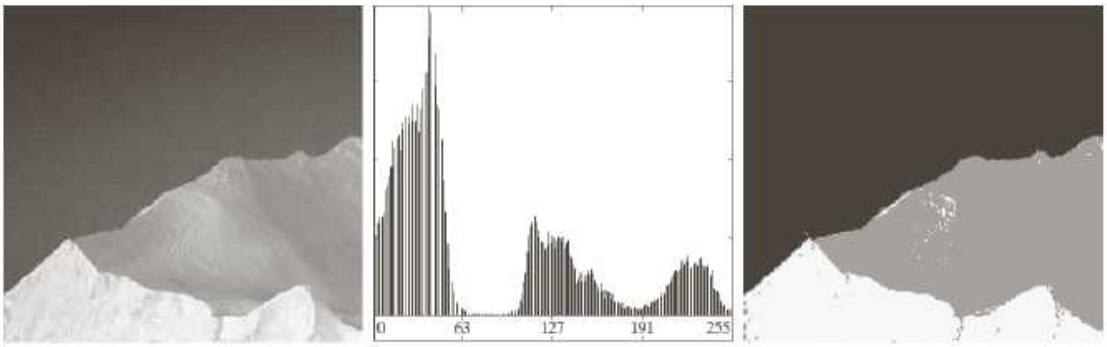
(4) **Repeat until** $k_1 = L-3$

This results in a 2-D array $\sigma_B^2(k_1, k_2)$, after which k_1^* and k_2^* that correspond to the maximum value in the array, are selected

Segmentation is as follows: $g(x,y) = \begin{cases} a, & \text{if } f(x,y) \leq k_1^* \\ b, & \text{if } k_1^* < f(x,y) \leq k_2^* \\ c, & \text{if } f(x,y) > k_2^* \end{cases}$

Separability measure: $\eta(k_1^*, k_2^*) = \dfrac{\sigma_B^2(k_1^*, k_2^*)}{\sigma_G^2}$

Example 10.19: Multiple global thresholding



a b c

FIGURE 10.45 (a) Image of iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds. (Original image courtesy of NOAA.)