TP Caml 5: Compression de niveaux de gris

lionel.rieg@ens-lyon.fr

9 décembre 2008 & 6 janvier 2009

1 Description théorique

1.1 Problème

On se donne une image en M niveaux de gris de $h \times l$ pixels représentée par une matrice d'entiers de taille $h \times l$ donnant le niveau de gris de chaque pixel. Notre but va être de réduire le nombre de niveaux de gris utilisés à k. Mais on veut le faire intelligemment de façon que la perte de qualité soit la plus faible possible. On va donc choisir soigneusement quels niveaux de gris garder.

Si on note $a_1, \ldots, a_k$ la palette des niveaux de gris conservés, on va remplacer le niveau de gris $v(p)$ de chaque pixel p par le a_i le plus proche, *i.e.* le niveau de gris a_i qui minimise l'erreur $|a_i - v(p)|$. On note $\varepsilon_{a_1,\ldots,a_k}(p)$ cette erreur d'approximation au pixel p.

Afin de limiter la perte en qualité, on va s'efforcer de trouver les k niveaux de gris qui minimisent $\varepsilon^t(a_1, \ldots, a_k)$ l'erreur totale sur l'image, définie comme

$$\varepsilon^t(a_1, \ldots, a_k) = \sum_{p \text{ pixel}} \varepsilon_{a_1,\ldots,a_k}(p)$$

1.2 La solution par programmation dynamique

Au vu de la définition de l'erreur totale, il est évident que seule la fréquence d'apparition de chaque niveau de gris est importante. On va donc utiliser la fréquence de chaque niveau de gris dans l'image (en fait, ce ne seront pas les fréquences mais plutôt les occurrences pour ne travailler qu'en entiers) plutôt que l'image elle-même. Ce sera facile à calculer : un simple parcours de l'image.

Pour résoudre ce problème, on va utiliser la programmation dynamique. Afin de réduire la dimension du problème (et par conséquent l'espace de solution à parcourir par programmation dynamique), on va supposer dans un premier temps que le dernier niveau de gris M de l'image originale est utilisé dans la palette à conserver. La formule de récurrence que l'on va utiliser donne donc l'erreur totale d'une répartition de k niveaux de gris dans une palette qui s'étale de 0 (noir) à $n < M$ (blanc) lorsque le dernier niveau de gris choisi est le dernier disponible, *i.e.* lorsque $a_k = n$. On ignore les pixels de niveaux de gris supérieurs à n (*i.e.* les nuances claires).

► **Question 1** Soit a_j et a_{j+1} deux niveaux de gris consécutifs dans la palette retenue. Calculer l'erreur $\delta(a_j, a_{j+1})$ commise en approximant par a_j ou a_{j+1} les pixels dont la couleur originale est comprise entre a_j et a_{j+1}.

► **Question 2** En notant $\text{Err}_k(n)$ l'erreur minimale de compression avec k niveaux de gris $a_1, \ldots, a_k$ compris entre 1 et n (avec $a_k = n$) sur les pixels de couleurs originales comprises également entre 1 et n, déterminer la relation de récurrence qui lie $\text{Err}_k(n)$ et les $\text{Err}_{k-1}(j)$. Quels sont les cas d'initialisation ?

2 Implémentation

► **Question 3** Écrire une fonction

```
interval_cost : int vect → int → int → int
```

qui calcule l'erreur engendrée par deux niveaux de gris consécutifs de la palette retenue, selon la formule calculée à la question 1. Ainsi, interval_cost freq a_i a_j calculera l'erreur commise en approchant par a_i ou a_j les pixels de couleurs comprises entre a_i et a_j sur une image dont le tableau de fréquence est donné par freq.

► **Question 4** Écrire l'algorithme de calcul de l'erreur totale minimale lorsque le dernier niveau de gris de la palette retenue est le dernier niveau de gris original (*i.e.* $a_k = n$) en utilisant la relation de récurrence de la question 2. Il sera de type

```
optimal_loss : int vect → int → int vect vect
```

► **Question 5** On veut obtenir une palette qui réalise le minimum de perte pour faire la compression. Modifier donc l'algorithme précédent pour garder trace de la case à la ligne précédente qui réalise l'optimum en la case courante. Son nouveau type sera

```
mem_optimal_loss : int vect → int → (int * int option) vect vect
```

► **Question 6** Terminer l'algorithme de calcul des niveaux de gris optimaux en omettant l'hypothèse selon laquelle le dernier niveau de gris conservé est le dernier disponible. Il renverra la liste des niveaux de gris d'une palette optimale (elle peut ne pas être unique).

3 Utilisation de l'algorithme

À présent que l'on possède l'algorithme de meilleur échantillonnage, on va s'en servir pour modifier de vraies images.

Pour commencer, il nous faut lire une image en niveaux de gris. Pour rester simple, on va utiliser le format PGM (Portable Grey Map), dont les spécifications (légèrement modifiées) sont :

1. sur la première ligne, le *nombre magique* « P2 » qui permet d'identifier le type du fichier (en plus de son extension)
2. sur la deuxième ligne et séparées par une espace [1], les largeur l et hauteur h de l'image
3. sur la troisième ligne, la valeur maximum M des niveaux de gris (0 = noir et M = blanc)
4. à partir de la quatrième ligne, h lignes de l nombres séparés par des espaces et compris entre 0 et M qui dénotent le niveau de gris de chaque pixel de l'image correspondant à la position dans cette matrice.

Pour manipuler les fichiers, on va se servir des types in_channel et out_channel de Caml qui autorisent respectivement à lire et écrire des fichiers mais pas les deux à la fois. Noter que lire ou écrire « consomme » le fichier, *i.e.* on ne peut relire ou récrire. Pour cela, on utilisera les fonctions suivantes de la bibliothèque standard :

- open_in : string →in_channel qui ouvre un fichier en lecture à partir de son nom
- input_line : in_channel →string qui permet de récupérer la ligne suivante d'un fichier ouvert en lecture
- close_in: in_channel →unit qui permet de fermer un fichier ouvert en lecture à la fin de son utilisation
- open_out : string →out_channel, output_string : out_channel →string →unit et close_out : out_channel →unit qui sont les pendants pour l'écriture

► **Question 7** Écrire une fonction

```
find_word : string → int → int * int
```

qui renvoie la position de début et la longueur du premier mot dans la chaîne de caractère donnée en premier argument à partir de la position donnée en second argument. Les espaces avant ce mot doivent être ignorées. Par exemple, si find_space s i vaut (k, l), cela signifie que le premier mot de s depuis le caractère s.[i] commence à l'indice k et comporte l caractères.

► **Question 8** Écrire une fonction get_header qui récupère l'en-tête d'une image PGM, *i.e.* qui prend en argument un fichier PGM ouvert à la seconde ligne et renvoie les dimensions de l'image et la valeur maximale des niveaux de gris (qui correspond au blanc).

► **Question 9** Écrire une fonction

```
frequency_array : in_channel → int → int → int → int vect
```

qui lit une image et génère le tableau de fréquence des niveaux de gris de l'image. Ses arguments sont :

- le fichier de l'image ouvert en lecture après l'en-tête (*i.e.* au début de la matrice)
- les dimensions de l'image (largeur puis hauteur)
- le niveau de gris M d'un pixel blanc

► **Question 10** Écrire une fonction qui, en appliquant l'algorithme défini à la partie précédente, transforme un tableau de fréquence en un tableau qui associe à chaque niveau de gris original celui de la palette qui le remplace.

► **Question 11** Écrire une fonction

```
compress : string → int → string → unit
```

qui compresse une image (à partir de son nom de fichier) en niveaux de gris en une image (en donnant le nom du fichier de sortie) avec moins de niveaux de gris. On n'oubliera pas de vérifier que le nombre magique est correct et d'écrire l'en-tête dans le fichier de sortie pour que le fichier compressé soit lisible.

1. En typographie, *espace* est un mot féminin.

4 Solutions

▸ **Question 1** Les pixels dont la couleur originale est comprise entre a_j et a_{j+1} vont être compressés soit en a_j, soit en a_{j+1} selon lequel des deux est le plus proche. Une façon d'écrire le résultat est alors $\delta(a_j, a_{j+1}) = \sum_{a_j \leqslant v(p) \leqslant a_{j+1}} \min(|v(p)-a_j|, |v(p)-a_{j+1}|)\cdot\text{freq}(p)$. On peut néanmoins trouver une forme plus explicite car la frontière entre les niveaux de gris qui seront associés à a_j et ceux associés à a_{j+1} se situe en la valeur moyenne $m = \left\lfloor \frac{a_j+a_{j+1}}{2} \right\rfloor$ (ne pas oublier de faire attention aux cas pair et impair), donc l'erreur commise pour ces pixels est $\delta(a_j, a_{j+1}) = \sum_{a_j \leqslant v \leqslant m} (v - a_j) \cdot \text{freq}(v) + \sum_{m < v(p) \leqslant a_{j+1}} (a_{j+1} - v) \cdot \text{freq}(v)$.

▸ **Question 2** En supprimant le dernier niveau de gris dans une palette optimale $a_1, \ldots, a_k$ à k niveaux de gris, on obtient une palette de compression optimale à $k-1$ niveaux de gris jusqu'à a_{k-1}. La preuve de cette propriété se fait très facilement par l'absurde. La relation de récurrence est donc :

$$\text{Err}_k(n) = \min_{k-1 \leqslant j \leqslant n-1} \Big(\text{Err}_{k-1}(j) + \delta(j, n) \Big)$$

▸ **Question 3** La sommation est faite différemment de la formule de la question 1 : par distance croissante depuis les extrémités. En détail, en supposant $a_{j+1} - a_j$ impair pour simplifier et en notant $m = \left\lfloor \frac{a_j+a_{j+1}}{2} \right\rfloor$, $\hat{m} = \frac{a_{j+1}-a_j-1}{2}$:

$$\begin{aligned}
\delta(a_j, a_{j+1}) &= \sum_{a_j \leqslant v \leqslant m} (v - a_j) \cdot \text{freq}(v) + \sum_{m < v(p) \leqslant a_{j+1}} (a_{j+1} - v) \cdot \text{freq}(v) \\
&= \sum_{v=a_j}^{m} (v - a_j) \cdot \text{freq}(v) + \sum_{v=m+1}^{a_{j+1}} (a_{j+1} - v) \cdot \text{freq}(v) \\
&= \sum_{i=0}^{\hat{m}} i \cdot \text{freq}(a_j + i) + \sum_{i=0}^{\hat{m}} i \cdot \text{freq}(a_{j+1} - i) \\
&= \sum_{i=0}^{\hat{m}} i \cdot \Big(\text{freq}(a_j + i) + \text{freq}(a_{j+1} - i) \Big)
\end{aligned}$$

Lorsque $a_{j+1} - a_j$ est pair, la seule différence est que la case médiane n'est pas comptée dans la dernière expression.

```
let interval_cost freq a_i a_j =
  let med = (a_j - a_i - 1) / 2 in
  let res = ref (if (a_j - a_i) mod 2 = 0 (* case centrale non comptée *)
                 then freq.(a_i + med + 1) * (med + 1) else 0) in
    for k = 1 to med do
      res := !res + k * (freq.(a_i + k) + freq.(a_j - k))
    done;
    !res ;;
```

▸ **Question 4** Comme l'initialisation de la première ligne est non triviale, il vaut mieux la séparer de la récurrence dans une fonction dédiée. Les valeurs de la première ligne $\text{Err}_1(n)$ sont les erreurs lorsque seul le dernier niveau de gris est conservé. Elles valent donc $\text{Err}_1(n) = \sum_{i=1}^{n} (n-i) \cdot \text{freq}(i)$, qu'on peut calculer plus astucieusement en remarquant que $\varepsilon_{\{n+1\}}(p) = 1 + \varepsilon_{\{n\}}(p)$ si $v(p) \leqslant n$ d'où la relation de récurrence suivante : $\text{Err}_1(n+1) = \text{Err}_1(n) + \sum_{i=0}^{n+1} \text{freq}(i)$. La somme des freq.(i) se trouve dans partial_sum.

```
let optimal_loss_init freq greys =
  let nb_colors = vect_length freq in
  let optima = make_matrix greys nb_colors 0 in
  let first_row = optima.(0) in
  let partial_sum = ref 0 in
    for n = 1 to nb_colors - 1 do
      partial_sum := !partial_sum + freq.(n - 1);
      first_row.(n) <- first_row.(n - 1) + !partial_sum
    done;
    optima ;;
```

```
let optimal_loss freq greys =
  let optima = optimal_loss_init freq greys in
    for k = 1 to greys - 1 do
      for n = k + 1 to (vect_length freq) - 1 do
        optima.(k).(n) <- optima.(k - 1).(n - 1);
        for j = k to n - 1 do
          optima.(k).(n) <-
            min optima.(k).(n) (optima.(k - 1).(j) + interval_cost freq j n)
        done
      done
    done;
    optima;;
```

▶ **Question 5** Il s'agit simplement de remplacer les valeurs de la matrices par des couples et de les mettre à jour.

```
let fst_min (a, b) (a', b') = if a <= a' then (a, b) else (a', b');;
```

À partir d'ici, on conserve la même organisation qu'à la question précédente, seules les affectations sont modifiées.

```
let mem_optimal_loss_init freq greys =
  let nb_colors = vect_length freq in
  let optima = make_matrix greys nb_colors (0, None) in
  let first_row = optima.(0) in
  let partial_sum = ref 0 in
    for n = 1 to nb_colors - 1 do
      partial_sum := !partial_sum + freq.(n - 1);
      first_row.(n) <- (fst first_row.(n - 1) + !partial_sum, None)
    done;
    optima;;

let mem_optimal_loss freq greys =
  let optima = mem_optimal_loss_init freq greys in
    for k = 1 to greys - 1 do
      for n = k to (vect_length freq) - 1 do
        optima.(k).(n) <- (fst optima.(k - 1).(n - 1), Some (n - 1));
        for j = k - 1 to n - 2 do
          optima.(k).(n) <-
            fst_min
              optima.(k).(n)
              (fst optima.(k - 1).(j) + interval_cost freq j n, Some j)
        done
      done
    done;
    optima;;
```

▶ **Question 6** Comme d'habitude, il y a plusieurs tâches à effectuer donc on les sépare en plusieurs fonctions :

– la fonction de calcul de l'erreur entre a_n et n :

```
let last_cost freq a_n =
  let n = vect_length freq - 1 in
  let sum = ref 0 in
    for i = a_n + 1 to n do
      sum := !sum + (i - a_n) * freq.(i);
    done;
    !sum;;
```

– la fonction de détermination de position du dernier niveau de gris de la palette retenue :

```
let find_last optima freq =
  let greys = vect_length optima in
  let last_row = optima.(greys - 1) in
  let nb_colors = vect_length last_row in
  let best = ref (fst last_row.(nb_colors - 1), nb_colors - 1) in
    for i = nb_colors - 2 downto greys - 1 do
      best :=
        fst_min !best (fst last_row.(i) + last_cost freq i, i)
    done;
    snd !best ;;
```

– la fonction de calcul d'une palette optimale :

```
let optimal_colors freq greys =
  let rec rebuild matrix acc i j =
    match snd matrix.(i).(j) with
      | None → j :: acc
      | Some next → rebuild matrix (j :: acc) (i-1) next
  in
  let optima = mem_optimal_loss freq greys in
    rebuild optima [] (greys - 1) (find_last optima freq );;
```

► **Question 7**

```
let find_word s start =
  let current = ref start in
  let s_length = string_length s in
    while !current < s_length && s.[!current] = ' ' do incr current done;
    let beginning = !current in
      while !current < s_length && s.[!current] <> ' ' do incr current done;
      beginning, !current - beginning ;;
```

Noter qu'on peut choisir d'élargir la notion d'« espace » (par exemple aux tabulations ou aux retours chariots) en utilisant une fonction auxiliaire is_space qui remplacerait le test d'égalité.

► **Question 8**

```
let get_header file =
  let dimensions = input_line file in
  let w1_beg, w1_length = find_word dimensions 0 in
  let w = int_of_string (sub_string dimensions w1_beg w1_length) in
  let w2_beg, w2_length = find_word dimensions (w1_beg + w1_length) in
  let h = int_of_string (sub_string dimensions w2_beg w2_length) in
  let white = int_of_string (input_line file ) in
    w, h, white ;;
```

► **Question 9** Comme le corps de la boucle est suffisamment complexe pour contenir des erreurs, il vaut mieux le séparer pour pouvoir le tester plus facilement. La première fonction s'occupe de gérer une ligne et la seconde d'itérer la première.

```
let get_line freq line width =
  let index = ref 0 in
    for j = 1 to width do
      let w_beg, w_length = find_word line !index in
      let grey = int_of_string (sub_string line w_beg w_length) in
        freq.(grey) <- freq.(grey) + 1;
        index := w_beg + w_length
    done;;
```

```
let frequency_array file width height white =
  let frequency = make_vect (white + 1) 0 in
    for i = 1 to height do
      get_line frequency (input_line file) width
    done;
    frequency ;;
```

▶ **Question 10** Tout d'abord, trois fonctions qui réalisent des sous-tâches : remplir des parties précises du tableau.

– avant la première valeur :

```
let fill_beg assoc a_1 =
  for i = 0 to a_1 do
    assoc.(i) <- a_1
  done;;
```

– entre deux valeurs consécutives :

```
let fill_middle assoc a_i a_j =
  let med = (a_i + a_j) / 2 in
    for i = a_i to med do
      assoc.(i) <- a_i
    done;
    for i = med + 1 to a_j do
      assoc.(i) <- a_j
    done;;
```

– après la dernière :

```
let fill_end assoc a_n =
  for i = a_n to (vect_length assoc) - 1 do
    assoc.(i) <- a_n
  done;;
```

Enfin la fonction demandée :

```
let matching_colors white colors =
  let color_vect = make_vect (white + 1) 0 in
  let rec list_it2 = function
    | [] -> failwith "matching_colors: no sample given"
    | [a_n] -> fill_end color_vect a_n
    | a_i :: (a_j :: _ as q) -> fill_middle color_vect a_i a_j; list_it2 q
  in
    fill_beg color_vect (hd colors);
    list_it2 colors;
    color_vect ;;
```

▶ **Question 11** De nouveau, plusieurs tâches = plusieurs fonctions :

– la conversion du texte de l'ancienne couleur en celui de la nouvelle

```
let new_color str index color_matching =
  let w_beg, w_length = find_word str !index in
  let old_color = int_of_string (sub_string str w_beg w_length) in
    index := w_beg + w_length;
    string_of_int (color_matching.(old_color));;
```

– l'écriture de la nouvelle matrice

```
let write_image file_in height width color_matching file_out =
  let line = ref "" in
  let index = ref 0 in
  for i = 1 to height do
    line := input_line file_in ;
    index := 0;
    for j = 1 to width - 1 do
      output_string file_out ((new_color ! line index color_matching) ^ " ")
    done;
    output_string file_out ((new_color ! line index color_matching) ^ "\n");
    flush file_out (* forcer l'écriture pour ne pas saturer le buffer *)
  done;;
```

– les tests de validité de l'image (nombre magique et quantité de niveaux de gris demandé)

```
let check_magic_number file_in =
  if input_line file_in <> "P2" (* vérifier le nombre magique du fichier *)
  then
    begin (* mauvais type de fichier *)
      close_in file_in ; (* à ne pas oublier !!! *)
      failwith "Wrong magic number! Compression aborted."
    end;;

let is_compression file_in white greys =
  if white < greys - 1 (* fait-on vraiment une compression ? *)
  then
    begin (* pas d'expansion *)
      close_in file_in ; (* à ne pas oublier !!! *)
      failwith "More grey levels asked than in the original image! \
                Compression aborted."
    end;;
```

– la compression de l'image avec la gestion de l'entête

```
let compress filename_in greys filename_out =
  let file_in = open_in filename_in in
    check_magic_number file_in;
    let width, height, white = get_header file_in in
      is_compression file_in white greys; (* fin des tests *)
      print_endline ("Compressing file " ^ filename_in ^ " of size "
                     ^ string_of_int width ^ "x" ^ string_of_int height
                     ^ " into " ^ string_of_int greys ^ " grey levels");
      let freq = frequency_array file_in width height white in
        close_in file_in ;
        let new_colors = matching_colors white (optimal_colors freq greys) in
        let file_in = open_in filename_in in
        let file_out = open_out filename_out in
          output_string file_out (input_line file_in ^ "\n"); (* P2 *)
          output_string file_out (input_line file_in ^ "\n"); (* w h *)
          output_string file_out (input_line file_in ^ "\n"); (* white*)
          write_image file_in height width new_colors file_out ;
          close_out file_out ;;
```

– en bonus : un test pour compresser toto, depuis l'image "toto.pgm" en 256 niveaux de gris

```
let test greys =
  let filename_in = "toto.pgm" in
  let filename_out = "toto-" ^ string_of_int greys ^ ".pgm" in
    compress filename_in greys filename_out ;;
```