Efficiency Cliff to Invert the Identity Map

H. Ben Ameur[1], F. Clément[2] & P. Weis[2]

1: IPEST and ENIT-LAMSIN, University of Tunis,
BP 37, Le Belvédère, 1002 Tunis, Tunisia
`hbenameur@yahoo.ca`
2: EPI Pomdapi, INRIA Paris - Rocquencourt,
Domaine de Voluceau, BP 105, F-78153 Le Chesnay cedex, France
`{Francois.Clement,Pierre.Weis}@inria.fr`

Abstract

1. Introduction

A mathematical approach for segmentation (well-foundedness), oracle mathematiquement fonde de la convergence : decroissance de J

2. Inverse problems and image segmentation

We denote by $\mathcal{F}_{a,b}$ the set of functions from $\mathbb{R}^a$ to $\mathbb{R}^b$, for any positive numbers a and b. Given some positive numbers l, m, p, and q, let F be a functional from $\mathcal{F}_{l,m}$ to $\mathcal{F}_{p,q}$.

Typically, for simulation applications, argument and result functions are scalar functions depending on a space variable ranging on a bounded support, thus $m = q = 1$, and $p \leq l \leq 3$. Moreover, functional F may be complicated to compute. For instance, it may involve the resolution of some partial differential equation depending on a distributed parameter, the argument of F. Thus, functional F is said to model the cause-to-effect relation between its argument and result functions.

Given such a functional F, we call *direct problem* associated to F the computation of $F(c)$ for some function c. We call *inverse problem* associated to F the estimation of a solution of the equation $F(c) = d$ for some function d in the range of F.

In practice, for numerical applications, input spaces of functions are discretized using finite meshes, and approximated functions are then represented by finite graphs. Moreover, F may be difficult to invert, or even may not be invertible, and data d may not be precisely in the range of F. Instead of exactly solving equation $F(c) = d$, we search for a function c that minimizes the Euclidean distance between d and $F(c)$. We call $J(c)$ the corresponding least-squares data misfit. Popular minimization algorithms involve the computation of the derivative of J with respect to c, and the routine resolution of direct problems.

This complex approach is well suited for the estimation of physical quantities that cannot be directly measured. For instance, this is the case for non-invasive control methods when there is no way to access inside the region of interest, as in seismic inversion for oil or gas prospection, and in most medical imaging techniques. For example, in elastography for tumor detection, unknown parameter c is Young's modulus distribution that measures the stiffness of an elastic material, data d are displacements induced by a mechanical compression measured at the surface of the body, and F is an elasticity model (e.g. see [3]). This is also the case when the measure acquisition device would not survive inside the region of interest, as in the inversion of spectroscopic measurements for estimating the temperature and the concentration of the exhaust gas of stato-reactors.

The refinement indicator approach proposed in [2] is designed to give an optimal answer to the least-squares formulation of the inverse problem: it iteratively builds a piecewise constant approximation of the unknown parameter function c that satisfies some regularity properties. Estimating such a piecewise constant function means finding a zonation, i.e. the geometry of zones, and the constant value taken in each zone. Zone boundaries pinpoint the spatial discontinuities of the parameter function c. Starting with a one-zone zonation c_0, i.e. a constant function, zones are iteratively splitted. At each iteration, we use refinement indicators to quantify at first order the effect of splitting zones on the least-squares misfit $J(c)$. Refinement indicators are inexpensive computations using the derivative of J with respect to c, which is already necessary to carry out the minimization. The idea is then to select the cutting providing the largest decrease of $J(c)$, and resume the iterative process. In other words, contrasts in the unknown function c having the largest effect on $J(c)$ are recovered first.

The segmentation of an image consists in identifying zones where the pixels share some property, such as similar colors. Let us consider the identity functional, $F = \mathrm{Id}$. Let data d be an image, i.e. a collection of pixels, represented as a discrete function from integer couples to Red/Green/Blue vectors, namely we have $l = p = 2$, and $m = q = 3$. Then, applying the refinement indicator approach to the associated inverse problem builds a finite sequence of segmentations of the given image, see [1]. Of course, the invert of the functional is perfectly known since it is the functional itself. The true interest of the method is the sequence of intermediary steps of the image reconstruction. In particular, the segmentation corresponding to a given number of zones, or to a given number of constrasts, is a valuable intermediate result. This is exactly what we are looking for when we have access to direct measurements of the physical quantity of interest, and want simpler representations of these measurements. The sequence of segmentations of an image starts with an homogeneous image with the mean color of d. Consecutive segmentations only differ by the splitting of a single zone. Segmented images are getting closer to the given image. The sequence ends with d itself where pixels with the very same color are ultimately grouped into zones, the data misfit and its derivative vanish, it is no more possible to split any zone. Such image segmentation approach inherits desirable properties from its mathematical origin: soundness, robustness, and flexibility (no gestalt inside). Furthermore, from the inverse problem point of view, the extreme simplicity of the functional to invert also brings very appealing properties. Computations of $F(c)$, and of the derivative $F'(c)$ are trivial. A closed-form formula gives the minimum of the least-squares data misfit $J(c)$ for a given geometry of zones. Last, but not least, the functional is diagonal: there is no interaction between the pixels, thus the evaluation of refinement indicators for the possible cuttings of a given zone does not depend on how other zones would be splitted.

In the simpler case of monochrome images, such as greyscale images, the crucial step of the determination of the best cutting in each zone happens to be simple: we just have to follow the sign of the derivative of J with respect to c inside each zone. When the derivative vanishes in some zone, there is no more possible best cutting in it. When the derivative vanishes in every zones, the minimum is reached.

For color images, functions d and c are intrinsically vector-valued. The derivative, or gradient, of J with respect to c becomes a vector, and it is no longer possible to define a sign for it. Of course, we can still define a “scalar best cutting” in each zone for each component. Multidimensional refinement indicators were introduced in [2], two versions and four variants of the segmentation algorithm using refinement indicators are proposed in [1].

When components of unknown function c are assumed to be dependent, we consider one zonation with a vector value, i.e. a color, in each zone. At each iteration, a “vector best cutting” is obtained by selecting the scalar best cutting inducing the largest decrease of the data misfit. This leads to the *vector* version of the segmentation algorithm.

When components are assumed to be independent, we consider a vector of scalar zonations, i.e. one zonation for each RGB channel. Then, at each iteration, each component has its scalar best cutting. This leads to three variants of the *multiscalar* version of the segmentation algorithm. With the *best_ component_ only* variant, only the scalar best cutting corresponding to the largest decrease of the data misfit is applied to its associated component. When each scalar best cutting is applied independently to its associated component, we call the variant *best_ component_for_ each.* Finally, for the *combine_ best_ components* variant, all scalar best cuttings are applied to all components. This last variant is less interesting since it may be impossible to build a segmentation with exactly a given number of zones (at each iteration a zone may be splitted into up to 8 subzones). It will not be considered any further.

Note that the vector version can be seen as a fourth variant of the multiscalar algorithm for which the selected scalar best cutting is applied to all components. Thus, it is called the *best_ component_for_ all* in the unified version of the algorithm presented in the next section.

3. Adaptive segmentation algorithm

3.1. Monochrome images

Let us consider the segmentation of monochrome images, such greyscale images. This means that the unknown function c is scalar valued, i.e. $m = q = 1$.

Scalar image functions c and d are represented by their graph on some finite domain of cardinal $i_{\max}$, $D = \{x_i, 0 \leq i < i_{\max}\}$.

$$c = \{(x_i, c_i), 0 \leq i < i_{\max}\} \qquad \text{and} \qquad d = \{(x_i, d_i), 0 \leq i < i_{\max}\}. \tag{1}$$

The least-squares data misfit is the discrete sum

$$J(c) = \frac{1}{2} \sum_{0 \leq i < i_{\max}} (d_i - c_i)^2. \tag{2}$$

The first derivative of the data misfit is given by

$$\forall i, 0 \leq i < i_{\max}, \quad \frac{\partial J}{\partial c_i} = c_i - d_i, \tag{3}$$

and the first order indicator associated with some split of some part $\mathbf{P}$ of domain D into parts $\mathbf{P}^+$ and $\mathbf{P}^-$ is given by

$$I_P = \left| \sum_{i \in \mathbf{P}^+} \frac{\partial J}{\partial c_i} - \sum_{i \in \mathbf{P}^-} \frac{\partial J}{\partial c_i} \right|. \tag{4}$$

The best split $C^\star_{\mathbf{P}}$, associated with the largest indicator, corresponds to group nonnegative value of the first derivative into $\mathbf{P}^+$, and negative ones into $\mathbf{P}^-$. The corresponding decrease of the data misfit is then given by the closed-form formula (see [1])

$$\Delta J_{\mathbf{P}} = \frac{\mathrm{Card}(\mathbf{P})}{\mathrm{Card}(\mathbf{P}^+)\,\mathrm{Card}(\mathbf{P}^-)} \frac{I^2_{\mathbf{P}}}{8}. \tag{5}$$

The geometry of zones is represented by a partition $\mathcal{P}$ of domain D. Scalar zonations, or scalar segmented images, are functions from parts of domain D to $\mathbb{R}$, represented by their graph on partitions,

$$s = \{(\mathbf{P}, s_{\mathbf{P}}), \mathbf{P} \in \mathcal{P}\}, \tag{6}$$

and the corresponding scalar image is given by

$$\forall x_i \in \mathbf{P}, \quad c_i = s_{\mathbf{P}}. \tag{7}$$

For a given partition $\mathcal{P}$, i.e. a given geometry of zones, the solution to the minimization of J with respect to s is given by

$$s = \tilde{\mathcal{M}}_{\mathcal{P}} d \tag{8}$$

where $\mathcal{M}_{\mathcal{P}}$ is the linear map $s \mapsto c$ defined by Equations (6) and (7), and $\tilde{\mathcal{M}}_{\mathcal{P}} = (\mathcal{M}_{\mathcal{P}}^T \mathcal{M}_{\mathcal{P}})^{-1} \mathcal{M}_{\mathcal{P}}^T$ is its least-squares pseudo-inverse. This amounts to associate each part in $\mathcal{P}$ with the mean value of the scalar data image on that part.

We summarize the algorithm for the scalar segmentation of some monochrome image d.

Initialization.

0. Define the initial partition $\mathcal{P}_0$ as the 1-part partition of domain D.

Iterations. For $n \geq 0$, do:

1. Compute the scalar segmented image associated with current partition $\mathcal{P}_n$, $s_n = \tilde{\mathcal{M}}_{\mathcal{P}_n} d$.
2. Compute data misfit $J_n = J(\mathcal{M}_{\mathcal{P}_n} s_n)$.
3. If $J_n = 0$, or the desired number of zones is attained, then the result is s_n.
4. For all parts $\mathbf{P}$ of partition $\mathcal{P}_n$, do:
   - 4a. Compute the best cutting $C_{\mathbf{P}}^\star$ and the associated refinement indicator $I_{\mathbf{P}}$.
   - 4b. Compute the decrease of the misfit function ΔJ associated with $C_{\mathbf{P}}^\star$.
5. Select part $\mathbf{P}^\star$ whose best cutting $C_{\mathbf{P}^\star}^\star$ is associated with the largest decrease ΔJ.
6. Define next partition $\mathcal{P}_{n+1}$ by splitting $\mathbf{P}^\star$ according to cutting $C_{\mathbf{P}^\star}^\star$.

3.2. Color images

Let us detail the multiscalar segmentation algorithm for color images. The unknown function c takes vector values of dimension 3, i.e. $m = q = 3$. Most quantities possess three components, corresponding to the three Red/Green/Blue channels of color images. Each component behaves as in the monochrome case, and is denoted with superscript ^{k} for $k \in \{R, G, B\}$.

Unknown color image function c and color data image d are represented by three graphs on the same domain D. The *total* least-squares data misfit is the sum of three "scalar" terms,

$$J(c) = \sum_{k \in \{R,G,B\}} J^k(c^k) \tag{9}$$

For a given multiscalar partition $\mathcal{P} = (\mathcal{P}^R, \mathcal{P}^G, \mathcal{P}^B)$, i.e. three "scalar" partitions, the linear map $\mathcal{M}_{\mathcal{P}}$ behind Equation (8) is a block-diagonal linear map with blocks $\mathcal{M}_{\mathcal{P}^R}$, $\mathcal{M}_{\mathcal{P}^G}$, and $\mathcal{M}_{\mathcal{P}^B}$.

We summarize the algorithm for the multiscalar segmentation of some color image d.

Initialization.

Scalar initializations. For all channel $k \in \{R, G, B\}$, do:

- 0a. Define the initial partition $\mathcal{P}_0^k$ as the 1-part partition of domain D.
- 0b. Compute the associated 1-part scalar segmented image $s_0^k = \tilde{\mathcal{M}}_{\mathcal{P}_0^k} d^k$.

0c. Compute the data misfit $J_0^k = J(\mathcal{M}_{\mathcal{P}_0}\, s_0^k)$.

Multiscalar initialization.

0d. Compute the total data misfit $J_0 = J_0^R + J_0^G + J_0^B$.

Iterations. For $n \geq 0$, do until $J_n = 0$, or the desired number of zones is attained:

Scalar iterations. For all channel $k = R, G, B$, if $J_n^k > 0$ do:

1. For all parts $\mathbf{P}$ of the current partition $\mathcal{P}_n^k$, do:
   - 1a. Determine the best cutting $C_{\mathbf{P}}^\star$ using refinement indicators.
   - 1b. Compute the decrease of the misfit function ΔJ associated with $C_{\mathbf{P}}^\star$:
     - *best_ component_for_ all*: also apply the cutting $C_{\mathbf{P}}^\star$ to other channels,
     - otherwise: keep other channels unchanged.
2. Select part $\mathbf{P}^\star$ whose best cutting $C_{\mathbf{P}^\star}^\star$ is associated with the largest decrease ΔJ.
3. Define next tentative partition $\mathcal{P}^{k\#}$ by splitting $\mathbf{P}^\star$ according to best cutting $C_{\mathbf{P}^\star}^\star$.

Update the multiscalar segmentation.

4. Determine the best channel $k^\star$ whose tentative partition $\mathcal{P}^{k^\star\#}$ is associated with the largest decrease ΔJ.
5. Define next multiscalar partition $\mathcal{P}_{n+1}$:
   - *best_ component_ only*: for all $k \in \{R, G, B\}$, $k \neq k^\star$, $\mathcal{P}_{n+1}^k = \mathcal{P}_n^k$, $\mathcal{P}_{n+1}^{k^\star} = \mathcal{P}^{k^\star\#}$,
   - *best_ component_for_ all*: for all $k \in \{R, G, B\}$, $\mathcal{P}_{n+1}^k = \mathcal{P}^{k^\star\#}$,
   - *best_ component_for_ each*: for all $k \in \{R, G, B\}$, $\mathcal{P}_{n+1}^k = \mathcal{P}^{k\#}$,
6. Define next multiscalar segmented image $s_{n+1} = \tilde{\mathcal{M}}_{\mathcal{P}_{n+1}}\, d$.

7a. For all channel $k = R, G, B$, do: $J_{n+1}^k = J(\mathcal{M}_{\mathcal{P}_{n+1}^k}\, s_{n+1}^k)$.

7b. Compute the next total data misfit $J_{n+1} = J_{n+1}^R + J_{n+1}^G + J_{n+1}^B$.

This algorithm is trivially generalizable to any dimension; for instance, we could handle the alpha channel in images by simply adding a fourth component to vector quantities.

For monochrome images d, all four variants of the multiscalar segmentation algorithm are isomorphic to the scalar segmentation algorithm.

4. First functional prototype

The vector variant of the algorithm was developed with a preliminary implementation based on imperative data structures: matrices (rectangular arrays of arrays) to represent images, lists of matrix cells (integer couples) to represent zones, and lists of zones to represent segmentations. Unfortunately, generalization to other variants proved to be so difficult that we decided to switch to purely functional data structures. Indeed, the adaptive segmentation algorithm is essentially based on the manipulation of dynamic data structures.

We implement the multiscalar segmentation algorithm presented in previous section within the functional paradigm. This will allow in the future for easier generalization when we will tackle the implementation of the general refinement indicators algorithm for the inversion of any given model F to estimate any (small) number of parameters. For instance, we use modules and functors for the abstraction of the dimension of the unknown parameter c.

As most numerical algorithms, the algorithm given in Section 3.2 is presented with an imperative flavor. As most iterative algorithm, it can be seen as the search for the fixpoint of some pure function, and can be rewritten as a pure function itself.

Let d be the image to segment, and D its domain. Let $n_{\max}$ be the desired number of zones for the segmentation of d. Let n be the current iteration number. let $\mathcal{P} = (\mathcal{P}^R, \mathcal{P}^G, \mathcal{P}^B)$ be the current multiscalar partition of D.

Define the function f to iterate as follows. Its argument is $(n, \mathcal{P}_n)$. Compute s_n and J_n following steps 1 and 2 of the algorithm. If $n \geq n_{\max} - 1$ or $J_n = 0$ (i.e. there is no more possible best cutting), then $f(n, \mathcal{P}_n) = (n, \mathcal{P}_n)$. Otherwise, compute $\mathcal{P}_{n+1}$ following steps 4 to 6. Then $f(n, \mathcal{P}_n) = (n+1, \mathcal{P}_{n+1})$.

Let $\mathcal{P}_0$ be the multiscalar partition given by step 0 of the algorithm. We compute $(n^\star, \mathcal{P}^\star)$ as the smallest fixpoint of f starting from $(0, \mathcal{P}_0)$. The result of the adaptive segmentation algorithm is the segmented image associated with $\mathcal{P}^\star$.

Let `initial_value` be the couple $(0, \mathcal{P}_0)$ where $\mathcal{P}_0$ is given by step 0 of the algorithm. Let `fixpoint` be the recursion that iterates a function over a given argument and outputs the fixpoint when it is reached. Let Then, given inputs d and $n_{\max}$, the functional version of the algorithm amounts to compute `fixpoint` $(f\ d\ n_{\max})$ (`init` d).

Components n and J

Type segmentation = scalar zonation codomain Always use multiscalar partitions, even for the vector variant for which all partitions are always equal (instead of using scalar codomain zonation). Abstraction of scalar, and codomain (-> functors)

Different types to represent images (or mappings): * image = color matrix * 'a gimage = graph = extentional representation * 'a fimage = function = intentional representation * 'a pimage = patchwork = structural representation

'a set = 'a list

'a patchwork = | Piece of 'a | Stitch of 'a patchwork_step 'a patchwork_step = patch : 'a patchwork; weave : 'a patchwork 'a step = | Patch of 'a patchwork | Weave of 'a patchwork 'a location = above : 'a step list; here : 'a patchwork; (= patchwork zipper)

('a, 'b) arrow = value : 'a; image : 'b; ('a, 'b) extentional_mapping = ('a, 'b) arrow set ('a, 'b) intentional_mapping = domain : 'a set; application : 'a -> 'b; ('a, 'b) structural_mapping = ('a, 'b) intentional_mapping patchwork

'a gimage = (point, 'a) extensional_mapping 'a fimage = (point, 'a) intentional_mapping 'a pimage = (point, 'a) structural_mapping = 'a fimage patchwork

The resulting program was presumably correct (no Coq proof of that), but was totally useless because of its inefficiency: complexity of the order of the square (is it correct?) of the number of pixels of the image to segment.

Reason = we repeatedly search for (find) all points in the structure when computing the explicit representation from the structural representation (why couldn’t we use map?).

The structural representation is built over the implicit representation. Why is it necessary to go back to the explicit representation (except for graphics and storage purposes)? Why is it not good to build the structural representation directly over the explicit representation?

5. Evolution of data structures

5.1. Imperative data structures with functional behavior

(iterator, map, fold)

We need direct access to items (pixels, zones) of the data structure. Why? Because to compute refinement indicators, we need to fold over all points of a given zone, that has no a priori structure.

Hash tables are not good since their functional behavior implies multiple expensive copies. AVL trees are still too slow (all finds = n log2 n).

Numbering of the elements of sets: direct access to the images of functions in vectors using the number of the element. Multi-index are too complicated: a lot of cases must be taken into account when computing the intersection, or the union of sets in the multi-d case. -> notion of universe with a linear numbering of all elements, independently of the underlying structure of sets (e.g., matrix slots for images).

’a uindex = private nat ’a uindices = (’a, ’a uindex) Hashtbl.t ’a elements = ’a array ’a universe = private indices : ’a uindices; mutable elements : ’a elements; mutable cardinal : nat; ’a index = ’a uindex uindex ’a nonempty_set = private universe : ’a universe; uindices : ’a index universe; ’a nonempty_subset = support : ’a nonempty_set; indices : ’a index array; nonempty_set : ’a nonempty_set; ’a dichotomy = ’a nonempty_subset * ’a nonempty_subset

’a list1_location = before : ’a list; here = ’a list1; (= list1 zipper)

’a images = | Constant of ’a | Images of ’a array (’a, ’b) graph = private domain : ’a nonempty_set; images : ’b images; (’a, ’b, ’c) block = map : (’a, ’b) graph; info : ’c (’a, ’b, ’c) structural_mapping = (’a, ’b, ’c) block list1 (’a, ’b, ’c) location = (’a, ’b, ’c) block list1_location

’a gimage = (point, ’a) graph ’a indicator = ’a ’a best_cutting = (’a indicator * point dichotomy) option ’a zone = (point, ’a, ’a best_cutting option) block = ’a gimage * ’a best_cutting option ’a pimage = (point, ’a, ’a best_cutting option) structural_mapping = ’a zone list1 = ’a gimage * ’a best_cutting option list1

5.2. Specialized hash tables

When the elements of sets follow some structure (e.g. matrix slots for pixels of an image), it is possible to define specialized hash tables that we always be perfect: buckets will always be singletons.

length = nat

’a value = ’a uindex (’sz, ’key) gen_numbering_hashtbl = length : length; size : ’sz; h : ’key -> ’key value; h_inv : ’key value -> ’key;

index_size = length ’a index_element = ’a uindex ’a index_hashtbl = private (index_size, ’a index_element) gen_numbering_hashtbl ’a index_universe = private tables : ’a index_hashtbl; cardinal : length;

matrix_size = length * length matrix_element = nat * nat matrix_hashtbl = private (matrix_size, matrix_element) gen_numbering_hashtbl matrix_universe = private tables :

matrix_hashtbl; cardinal : length; matrix_uindex = matrix_element uindex matrix_index = matrix_element uindex uindex matrix_nonempty_set = private universe : matrix_universe; uindices : matrix_element index_universe; matrix_nonempty_subset = support : matrix_nonempty_set; indices : matrix_index array; nonempty_set : matrix_nonempty_set; matrix_dichotomy = matrix_nonempty_subset * matrix_nonempty_subset

'a matrix_graph = private domain : matrix_nonempty_set; images : 'a images; ('a, 'b) matrix_block = ('a matrix_graph, 'b) arrow ('a, 'b) matrix_structural_mapping = ('a matrix_graph, 'b) delayed_graph

'a zone_cutting_candidate = indicator : 'a indicator; cutting : matrix_dichotomy; 'a best_zone_cutting = | No_more_zone_cutting | Best_zone_cutting of 'a zone_cutting_candidate

'a gimage = 'a matrix_graph 'a zone = ('a, 'a best_zone_cutting) matrix_block 'a pimage = ('a, 'a best_zone_cutting) matrix_structural_mapping

6. Structural equality versus physical equality

7. Acceleration of the algorithm

Careful analysis of the algorithm to prohibit any useless computation.

We compute in Table 7 the complexity of the scalar adaptive segmentation algorithm in terms of floating-point operations. Let N be the number of pixels of the monochrome image to be segmented. Let n be the iteration number. In the *naive* interpretation of the algorithm (second column), all parts are visited at each iteration. In the *memo* interpretation (third column), only the two most recent zones are visited, since computations in other zones are not modified by the most recent split. In this case, we denote by p_n^+ and p_n^- the number of pixels in the two zones resulting from the split at iteration n. Step 1a amounts to compute the gradient (1 flop per point, see Equation (3)), and then the first order indicator (1 flop per point, see Equation (4)). Step 2 implements Equation (5). Steps 3 and 4 implement Equation (8). Step 5 is for Equation (2). To estimate the total complexity for the n first iterations in the case of the memo interpretation, we assume that at each iteration, all zones are of equal size, i.e. $p_n^+ = p_n^- \simeq N/(n+1)$.

We always have $n \leq N$, and equality corresponds to singleton zones only, in the case where each pixel of the image has its own distinct color (this is limited to 16-megapixel images for 8-bit RGB colors). For a full reconstruction for which the data misfit vanishes, assuming that the number of iterations, i.e. the number of distinct colors in the image, is of the same order than the size of the image, the naive interpretations is of order n^2 whereas the memo interpretations is of order $n \ln n$.

	naive	memo
step 1a	$2N$	$2(p_{n-1}^+ + p_{n-1}^-)$
step 1b	$5(n+1)$	5
step 2	$n+1$	$Cn \ln n?$
steps 3,4	N	$p_n^+ + p_n^-$
step 5	$3N$	$3(p_n^+ + p_n^-)$
total	$6Nn + 5n^2$	$(12N + Cn) \ln n$

Table 1: Complexity of the scalar adaptive segmentation algorithm. For each step, we give the number of floating-point operations for the naive and memo interpretation.

For each channel, function `cut` implements steps 1a and 1b of the algorithm: it associates each zone with its best cutting and the corresponding exact indicator (decrease of the data misfit). Since the identity model is a diagonal linear operator, there is no influence from one pixel to the other, thus the evaluation of the best cutting in some zone is not modified by the split of another zone. Therefore, it is only necessary to evaluate the best cutting for the two new zones that were just created at the previous iteration. In other words, function `cut` must have some memo aspect.

The classical approach to make a function memo is to use an hash table. It is not efficient enough (find).

We need a memo function with direct access, i.e. memo for the poor, i.e. a graph.

Furthermore, we know exactly were we need to do the missing computation: for the two last appeared zones. -> stamped dynamic array (why?)

'a dynamic_array = mutable array : 'a array; default : 'a;

'a stamped = | Unset | Valid of 'a 'a stamped_darray = darray : 'a stamped dynamic_array; mutable next_unset : nat;

arrow_number = nat 'a image = | Not_yet_computed | Image of 'a ('a, 'b) arrow = value : 'a; image : 'b image; ('a, 'b) delayed_graph = private f : 'a -> 'b; graph : ('a, 'b) arrow stamped_darray; to_do : arrow_number list;

('a, 'b, 'c) block = (('a, 'b) graph, 'c) arrow ('a, 'b, 'c) structural_mapping = (('a, 'b) graph, 'c) delayed_graph

'a indicator = 'a 'a zone_cutting_candidate = indicator : 'a indicator; cutting : point dichotomy; 'a best_zone_cutting = | No_more_zone_cutting | Best_zone_cutting of 'a zone_cutting_candidate

'a zone = (point, 'a, 'a best_zone_cutting) block 'a pimage = (point, 'a, 'a best_zone_cutting) structural_mapping

8. General purpose library

8.1. Ulib

Unatural, Upositive_rational, Umatrix Umatrix_index, Umatrix_universe, Umatrix_nonempty_set, Umatrix_graph, Umatrix_structural_mapping Uhistory? Uordered_field, Uvector, Utuple

8.2. OCamlGen

8.3. Pph

9. Future work

Webdemo.

Smooth the gradient to produce zones with more regular boundaries.

Implement the more general form of the algorithm for the inversion of nonidentity forward map F.

Uhistory module (+ digest).

10. Conclusions

References

[1] H. Ben Ameur, G. Chavent, F. Clément, and P. Weis. Image segmentation with multidimensional refinement indicators. *Inverse Problems in Science and Engineering*, 19(5):577–597, 2011. Special Issue: Proceedings of the 5th Internat. Conf. on Inverse Problems: Modeling and Simulation, May 24th–29th, 2010, held in Antalya, Turkey.

[2] H. Ben Ameur, F. Clément, P. Weis, and G. Chavent. The multidimensional refinement indicators algorithm for optimal parameterization. *Journal of Inverse and Ill-Posed Problems*, 16:107–126, 2008.

[3] J. Fehrenbach, M. Masmoudi, R. Souchon, and P. Trompette. Detection of small inclusions by elastography. *Inverse Problems*, 22(3):1055–1069, 2006.