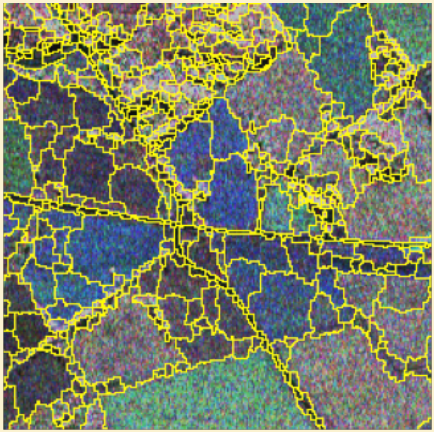
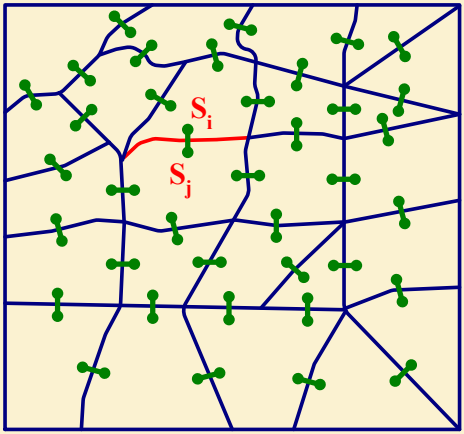
BeaulieuJM.ca/publi/Beau21sh

Segmentation hiérarchique avec Julia

Jean-Marie Beaulieu





Julia - un langage pour les scientifiques

Les types

pas des objets

Abstrait : 'abstract type T1'
'struct' : concret & terminal
Hiérarchie
    abstract type T2 <: T1
    struct T3 <: T2
Paramétré : struct T4{P}

abstrait
un seul

Les fonctions

function **nom**(a::T3, b::T2, c)

Une fonction '**nom**' est spécialisée par les types de ses arguments.

Sélection dynamique (dispatch)
➛ à l'exécution

Dynamique : compilé et/ou interprété
C++ Python
efficacité et souplesse

compilé à l'exécution
intégration avec le compilateur llvm

Générique : dynamique, obj. type, macro

fonction : compilé à l'exécution
définition ➛ arg. abstrait
appel ➛ arg. contrait

Mode interactif : fenêtre de travail
comme Matlab ou Mathematica

in : instruction
out : résultat

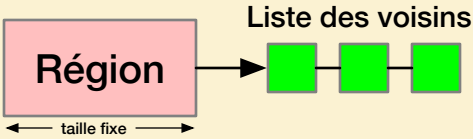
interface avec une foule de langages et de librairies
C/C++, Python, Java, Fortran, R, Linpack, ML, Plotly ...

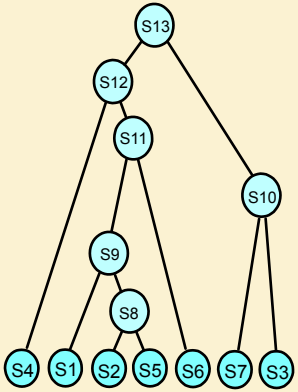
Segmentation hiérarchique

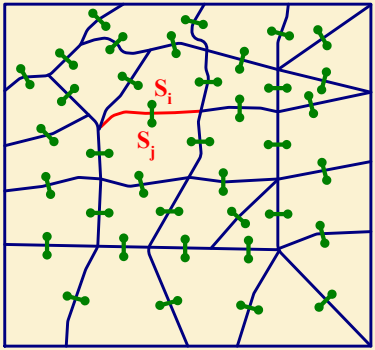
Produit un arbre des fusions

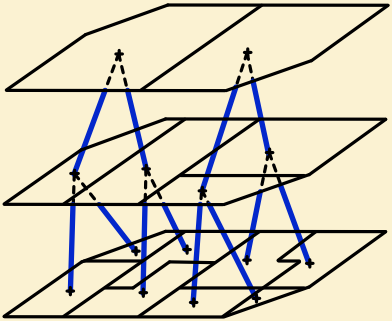
Optimisation séquentielle

Choisit le meilleur voisin

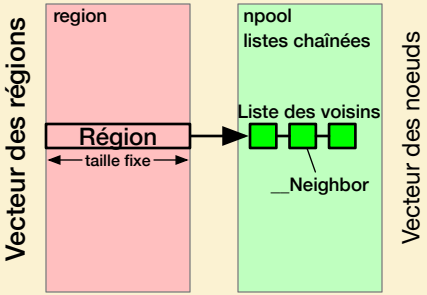


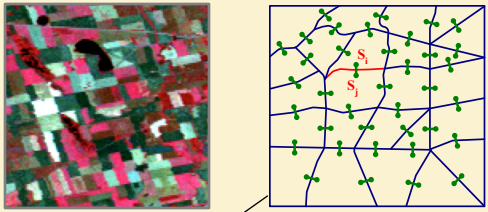






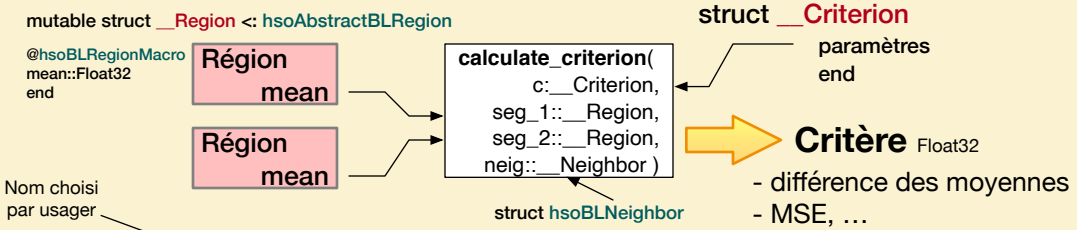
Architecture





Le Critère

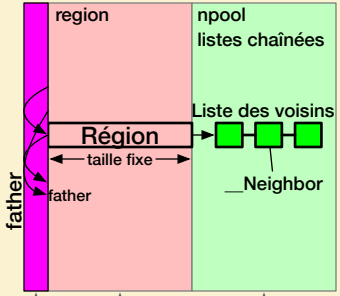
- permet de spécialiser le programme / les fonctions
- mesure la similarité entre deux régions



Les types __Criterion (C), __Region (R), __Neighbor (N) et le type d'image (P) spécialisent le programme

mutable struct hsoSegmentation{C,P,R,N}

Instanciation de hsoSegmentation



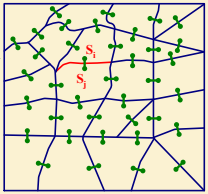
hs <— allocation des données
définition des types

__Criterion
type du pixel
__Region
__Neighbor

INPUT

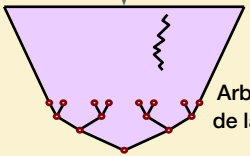
mutable struct **hsoSegmentation**{C,P,R,N}

IM ::Matrix{P}
label ::LabelImage
region ::Vector{R}
father ::Vector{hsoSegID}
npool ::azDataPool{N,hsoNeigID}
cri ::C
merger ::hsoMergerParam{R}
mparam::hsoMergeListParam
mlist ::Vector
ngsort ::BinaryHeap
crisort ::BinaryHeap{hsoCriNode,
hsoCriterionOrdering}

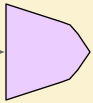


paramètres
__Criterion
contrôle
struct
struct
Segment Tree

OUTPUT
liste des fusions

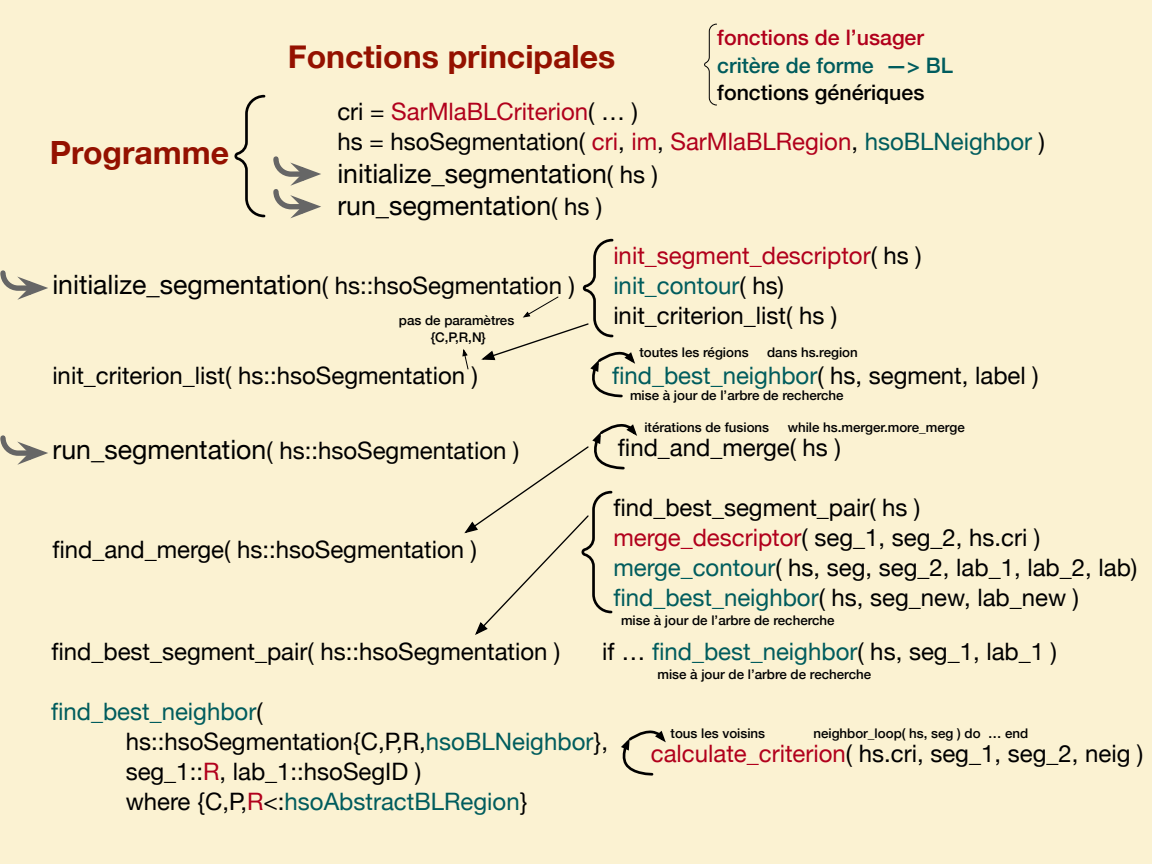


Arbre de recherche
de la meilleure paire



trie local des voisins

end


Fonctions principales
fonctions de l'usager
critère de forme —> BL
fonctions génériques
Programme
cri = SarMlaBLCriterion(…)
hs = hsoSegmentation(cri, im, SarMlaBLRegion, hsoBLNeighbor)
initialize_segmentation(hs)
run_segmentation(hs)
initialize_segmentation(hs::hsoSegmentation)
init_segment_descriptor(hs)
init_contour(hs)
init_criterion_list(hs)
pas de paramètres
{C,P,R,N}
init_criterion_list(hs::hsoSegmentation)
toutes les régions dans hs.region
find_best_neighbor(hs, segment, label)
mise à jour de l'arbre de recherche
run_segmentation(hs::hsoSegmentation)
itérations de fusions while hs.merger.more_merge
find_and_merge(hs)
find_and_merge(hs::hsoSegmentation)
find_best_segment_pair(hs)
merge_descriptor(seg_1, seg_2, hs.cri)
merge_contour(hs, seg, seg_2, lab_1, lab_2, lab)
find_best_neighbor(hs, seg_new, lab_new)
mise à jour de l'arbre de recherche
find_best_segment_pair(hs::hsoSegmentation)
if … find_best_neighbor(hs, seg_1, lab_1)
mise à jour de l'arbre de recherche
find_best_neighbor(
hs::hsoSegmentation{C,P,R,hsoBLNeighbor},
seg_1::R, lab_1::hsoSegID)
where {C,P,R<:hsoAbstractBLRegion}
tous les voisins neighbor_loop(hs, seg) do … end
calculate_criterion(hs.cri, seg_1, seg_2, neig)

Défini par l'usager

Critère de forme — BL

```
mutable struct SarMlaBLRegion      <: hsoAbstractBLRegion
    @hsoBLRegionMacro
    mean ::Float32
end

struct SarMlaBLCriterion
    nlook::Float32
    shape_dc::CriFactorDecay
end
```

```
macro hsoBLRegionMacro( )
  esc( quote
      r_best::hsoSegID
      r_nb::hsoNeigID
      r_neigh::hsoNeigID
      r_clng::hsoLineLength
      ssize::hsoCoord
      box::xyRect{hsoCoord}
  end )
end
```

liste des voisins

critère de forme

```
struct hsoBLNeighbor <:hsoAbstractNeighbor
    label::hsoSegID
    blng::hsoLineLength
end
```

```
function init_segment_descriptor(
        hs::hsoSegmentation{ SarMlaBLCriterion,P,
        SarMlaBLRegion,hsoBLNeighbour } ) where {P}
    mp  = hs.mparam
    seg = hs.region
    for lab=1:mp.nb_init_seg
        seg[lab] = SarMlaBLRegion()
    end
    for pix=1:mp.nb_pixel, lin=1:mp.nb_line
        label = hs.label[pix,lin]
        seg[label].mean += hs.IM[pix,lin]
        update_shape( seg[label], xyPoint{hsoCoord}(pix,lin) )
    end
end
```

```
function init_contour( … )
function merge_contour( … )
function find_best_neighbor( … )
```

Fonctions principales

```
function merge_descriptor( seg_1::SarMlaBLRegion,
        seg_2::SarMlaBLRegion, c::SarMlaBLCriterion)
    seg_1.mean += seg_2.mean
    merge_shape( seg_1, seg_2 )
end
```

```
function update_shape( seg::R, pos::xyAPoint{hsoCoord} )
                    where {R<:hsoAbstractBLRegion}
    seg.box = seg.ssize==0 ? xyRect(pos,pos) : xyBound(seg.box,pos)
    seg.ssize += 1
end
```

```
function calculate_criterion( c::SarMlaBLCriterion,
        seg_1::SarMlaBLRegion, seg_2::SarMlaBLRegion,
        neig::hsoBLNeighbour )
    n1,n2 = seg_1.ssize, seg_2.ssize
    nn = n1 + n2
    mean  = (seg_1.mean + seg_2.mean) / nn
    criter = nn*log(mean) - n1*log(seg_1.mean/n1) - n2*log(seg_2.mean/n2)
    sfac = shape_criterion( seg_1, seg_2, xyPoint(c.xside,c.yside), neig.blng )
    return criter * azweight( c.shape_dc, sfac, nn )
end
```

```
function merge_shape( seg1::R, seg2::R )
                    where {R<:hsoAbstractBLRegion}
    seg1.box   = xyBound(seg1.box,seg2.box)
    seg1.ssize += seg2.ssize
end
```

```
function shape_criterion( seg1::R, seg2::R, scale::xyAPoint,
        boundary::Real )  where {R<:hsoAbstractBLRegion}
    ....  end
```

Conclusion

Julia est un langage d'avenir.

Il permet une implémentation simple et efficace de la segmentation hiérarchique.