Jean-Marie Beaulieu

BeaulieuJM.ca

[Beau21sh]

Segmentation hiérarchique avec Julia

Author: **Beaulieu** Jean-Marie

Conference: Congrès 2021 de L'Association Québécoise de Télédétection

2-3 juin, 2021

URL: https://event.fourwaves.com/fr/aqt2021/pages

Segmentation hiérarchique avec Julia,
Beaulieu Jean-Marie,
Congrès 2021 de L'Association Québécoise de Télédétection, 2-3 juin, 2021.
[Bibtex]

DOWNLOAD the long abstract
DOWNLOAD the slide PDF file

L'Association Québécoise de Télédétection
Congrès virtuel 2021
https://laqt.ca

BeaulieuJM.ca/publi/Beau21sh

L'AQT
L'ASSOCIATION QUÉBÉCOISE DE TÉLÉDÉTECTION

Segmentation hiérarchique avec Julia

Jean-Marie Beaulieu
courriel: jean-marie.beaulieu@ift.ulaval.ca, site web: BeaulieuJM.ca

Une implémentation efficace de la segmentation hiérarchique d'image de télédétection a été développée initialement en Fortran, puis en C et en C++. Julia est un langage récent très prometteur. Il recouvre et combine les caractéristiques de plusieurs langages et logiciels populaires: Python, C, C++, R, Matlab … Dernièrement, nous avons réécrit la segmentation hiérarchique par optimisation séquentielle dans le langage Julia. Nous avons obtenu un code plus compact et simple à gérer.

La définition de la tâche de segmentation tourne autour de 4 structures et un ensemble de fonctions. Une région est décrite par une structure et elle référence une liste de voisins. La structure 'voisin' contient le numéro de la région voisine. La structure 'critère' contient les paramètres nécessaires dans le calcul de la similarité entre deux régions. Une structure globale 'segmentation' contient toute l'information et les données pour le processus de segmentation. Lors de son instanciation, tous les types de données nécessaires pour la segmentation sont définis. Ensuite, les fonctions 'initialisation' et 'exécution' font la tâche. La structure contient, entre autres, un vecteur de 'région' et un vecteur de nœuds de 'voisin' pour des listes chaînées de voisins.

Dans Julia, il n'y a pas de méthodes rattachées à un objet. Les fonctions ou méthodes sont choisies dynamiquement en fonction du type des arguments. Les structures sont terminales et peuvent avoir un parent de type abstrait. On obtient des fonctions plus ou moins génériques en utilisant les types abstraits comme arguments. Un usager qui veut définir un nouveau critère de segmentation doit définir un type 'région' et 'critère'. On a déjà défini un type abstrait 'région avec forme' et les fonctions associées pour utiliser la forme des régions dans le calcul de la mesure de similarité entre régions. L'usager doit définir une structure 'région' et utiliser une macro pour y inclure l'information de forme. Il doit y ajouter l'information spécifique à son type d'image et à son critère, par exemple, la moyenne de la région. La moyenne peut être un scalaire ou un vecteur selon le type d'image. Il doit ensuite définir les fonctions 'init_les_régions', 'fusion' pour deux régions et 'calcul_critère' entre deux régions. Les fonctions 'init_listes_voisins' et 'fusion_voisins' pour deux régions sont déjà définies pour le type abstrait 'région avec forme'.

La segmentation d'une image passe par l'exécution d'une cascade de fonctions. Chacune des fonctions est relativement simple. L'initialisation appelle 3 fonctions: 'init_les_régions', 'init_listes_voisins' et 'init_critères'. Cette dernière contient une boucle sur les régions et des appels à 'trouve_meilleur_voisin'. Celle-ci est une boucle sur les voisins de la région pour calculer le critère et retenir le meilleur. 'init_critères' place les meilleurs critères dans un arbre de recherche du minimum.

La fonction 'exécution' est une simple boucle qui appelle 'trouve_et_fusionne' la meilleure paire de régions. Celle-ci appelle 'trouve_meilleure_paire', puis 'fusion' et 'fusion_voisins' avec les deux régions pour obtenir une nouvelle région. On trouve son meilleur voisin et met à jour l'arbre de recherche du critère minimum. 'trouve_meilleure_paire', retire le minimum de l'arbre de recherche et vérifie si l'une des deux régions a déjà été fusionnée. Si oui, on retire la région ou met à jour le meilleur voisin de l'autre et retourne au début de la fonction.

— Instanciation de 'segmentation'
— 'initialisation' — 'init_les_régions'
— 'init_listes_voisins'
— 'init_critères' — 'trouve_meilleur_voisin' — 'calcul_critère'
— ajout dans l'arbre de recherche
— 'exécution' — 'trouve_et_fusionne'
— 'trouve_meilleure_paire'
— retire le minimum de l'arbre de recherche
— si déjà fusionné - retire ou met à jour
— 'fusion' régions et 'fusion_voisins'
— 'trouve_meilleur_voisin' — 'calcul_critère'
— ajout dans l'arbre de recherche

Nous avons présenté les éléments principaux d'une implémentation de la segmentation hiérarchique. Il y a d'autres structures, types et fonctions utilisés pour le contrôle des fusions et la sauvegarde de l'arbre des fusions qui est le résultat du programme. Plusieurs fonctions de support, souvent d'une ou deux lignes, permettent de simplifier l'écriture. Par exemple, des fonctions permettent d'itérer sur une liste des voisins avec une simple instruction 'for'.

Dans notre implémentation de la segmentation hiérarchique, nous supposons que le nombre moyen de voisins pour chaque région demeure faible, par exemple, entre 4 et 8. Tous les critères ne rencontrent pas nécessairement cette contrainte. À cette fin, nous avons ajouté des critères de forme pour la segmentation d'image SAR avec un canal. Julia est un mélange de langage dynamique et compilé. Le temps nous dira si notre implémentation est aussi rapide que celle en C++, ce qui semble possible.