An improved seeded region growing algorithm

Andrew Mehnert *, Paul Jackway

Cooperative Research Centre for Sensor Signal and Information Processing, Department of Electrical and Computer Engineering, The University of Queensland, Brisbane, Qld 4072, Australia

Received 20 January 1997; revised 20 August 1997

Abstract

Recently Adams and Bischof (1994) proposed a novel region growing algorithm for segmenting intensity images. The inputs to the algorithm are the intensity image and a set of seeds — individual points or connected components — that identify the individual regions to be segmented. The algorithm grows these seed regions until all of the image pixels have been assimilated. Unfortunately the algorithm is inherently dependent on the order of pixel processing. This means, for example, that raster order processing and anti-raster order processing do not, in general, lead to the same tessellation. In this paper we propose an improved seeded region growing algorithm that retains the advantages of the Adams and Bischof algorithm — fast execution, robust segmentation, and no tuning parameters — but is pixel order independent. © 1997 Elsevier Science B.V.

Keywords: Priority queue; Seeded region growing; Segmentation; Watershed segmentation

1. Introduction

One of the early tasks in image analysis is to segment an image into its constituent parts. There are four basic approaches to image segmentation (Zhu and Yuille, 1996):
1. global optimisation,
2. local filtering (edge detectors),
3. snakes and balloons, and
4. region growing and merging.

Recently Adams and Bischof (1994) proposed a novel region growing algorithm called seeded region growing (SRG). The algorithm is fast, robust, and parameter free. It takes an intensity image and a set of seeds — individual points or connected components — as inputs. The seeds play the same role as the markers used in watershed segmentation (Meyer and Beucher, 1990). They mark each of the objects (regions) to be segmented. The SRG algorithm operates on the premise that the pixels within a region are similar. The algorithm grows the seed regions in an iterative fashion. At each iteration all those pixels that border the growing regions are examined. The pixel that is most similar to a region that it borders is appended to that region. Unfortunately the SRG algorithm is inherently dependent on the order of processing of the image pixels. One implication of this is that raster order processing and anti-raster order processing do not, in general, lead to the same tessellation. This order dependency is particularly evident when the regions are small and of very similar intensity. Order dependency is clearly an
undesirable property, especially when the images to be segmented have no obvious orientation. In this paper we propose an improved seeded region growing (ISRG) algorithm that is pixel order independent. Although the ISRG algorithm is more complex than the SRG algorithm, it retains all of its advantages – fast execution, robust segmentation, and no tuning parameters – whilst at the same time eliminating the disadvantage of pixel order dependence.

The remainder of this paper is organised as follows. In Section 2 the operation of the SRG algorithm is carefully examined and its inherent pixel order dependencies are highlighted. The shortcomings of the implementation proposed by Adams and Bischof are also detailed. In Section 3 we propose several improvements to the SRG algorithm that redress its order dependencies. In Section 4 we then describe an implementation of the ISRG algorithm based on a priority queue and several last-in-first-out (LIFO) queues. Finally, in Section 5 we provide a short discussion and conclusion.

2. The Adams and Bischof seeded region growing algorithm

2.1. Description

The seeded region growing approach to image segmentation is to segment an image into regions with respect to a set of n seed regions (Adams and Bischof, 1994). Each seed region is a connected component comprising one or more points and is represented by a set Ai, where i = 1, 2, ..., n. Let T be the set of all unallocated pixels that border at least one of the Ai, i.e.

\[ T = \{ x \in \bigcup_{i=1}^{n} A_i : N(x) \cap \bigcup_{j=1}^{n} A_j \neq \emptyset \} \]  

where \( N(x) \) represents the set of immediate neighbours – 6 for the hexagonal grid and either 4 or 8 for the square grid – of the pixel x. A single step of the algorithm involves examining the neighbours of each \( x \in T \) in turn. If \( N(x) \) intersects a region \( A_j \) then a measure, \( \delta(x) \), of the difference (similarity) between x and the intersected region is calculated. In the simplest case \( \delta(x) \) is defined:

\[ \delta(x) = |g(x) - \text{mean}_{y \in A_j} \{ g(y) \} |, \]  

where g(x) is the intensity (grey value) of the pixel x. If \( N(x) \) intersects more than one region then \( A_j \) is taken to be that region for which \( \delta(x) \) is a minimum (alternatively, the pixel x can be flagged as a boundary pixel for display purposes). In this way a \( \delta \) value is determined for each \( x \in T \). Finally, the pixel \( z \in T \) that satisfies

\[ \delta(z) = \min_{x \in T} \{ \delta(x) \} \]  

is appended to the region corresponding to \( \delta(z) \). The new state of the regions \( \{ A_j \} \) then constitute the input to the next iteration. This process continues until all of the image pixels have been assimilated.

2.2. Inherent order dependencies

The SRG algorithm has two inherent pixel order dependencies. The first manifests itself whenever, during an iteration, several \( x \in T \) determine the same, minimum, \( \delta \) value. Eq. (3) then offers several possible choices for z. The particular z chosen influences the running mean of the region that it is assigned to. This in turn influences the \( \delta \) values calculated for the \( x \in T \) in the next iteration, and ultimately affects the final segmentation. This problem is illustrated in Fig. 1. The second order dependency manifests itself whenever the chosen z has the same \( \delta \) value for several regions that it borders. Once again resolution of the deadlock ultimately influences the final segmentation (this would be the situation if the centre pixel in Fig. 1(d) was the first of the five “3”s, with a \( \delta \) value of 2, to be processed).

2.3. Implementation order dependencies

In implementing the SRG algorithm Adams and Bischof utilise a data structure called the sequentially sorted list (SSL). In their implementation the SSL is a linked list of pixel addresses, ordered with respect to \( \delta \). A pixel can be arbitrarily inserted into the list in the position prescribed by its \( \delta \) value. However,
only the pixel with the smallest \( \delta \) value can be removed from the SSL. Effectively, the SSL stores the points of the set \( T \) ordered according to \( \delta \). Adams and Bischof (1994) noted that their implementation does not update previous entries in the SSL to reflect new differences from a region whose mean has been updated. They stated that “this leads to negligible difference in the results, but greatly enhanced speed” (Adams and Bischof, 1994) (p. 643). As a consequence, in addition to the pixel order dependencies induced by the SRG algorithm, the SRG implementation is subject to two other pixel order dependencies. The first order dependency manifests itself during the initial process of adding the neighbours of the seed regions to the SSL. In particular, if a pixel borders two or more seed regions it is given a \( \delta \) value based on its similarity to that seed region which happens to be first in terms of the order of processing of the image pixels. Once inserted into the SSL the pixel position is never updated. The second order dependency manifests itself whenever the neighbours of a newly labelled pixel are added to the SSL. The order in which the neighbours are scanned can affect the \( \delta \) value assigned to each and hence their ordering within the SSL.

3. Improved seeded region growing algorithm

The first order dependency in the SRG algorithm is eliminated if all of the pixels \( x \in T \) that have the same minimum \( \delta \) value are processed in parallel. This means that no pixel can be labelled, and therefore no region means can be updated, until all other pixels at that priority have been examined. Thus for the situation depicted in Fig. 1(d) the “3”s (with a \( \delta \) value of 2) must be assigned labels independently of one another. Only once all the labels have been determined are the region means updated. If a pixel cannot be labelled because it is equally similar to two or more adjacent regions (same \( \delta \) values) – the situation that gives rise to the second pixel order dependency of the SRG algorithm – then it is marked as tied and takes no further part in the region

![Fig. 1. Order dependency of the SRG algorithm. (a) Grey value test image with four seeds marked (the initial \( \{ A_i \} \)). (b) Each \( x \in T \) is shown with its \( \delta \) value as a superscript numeral. (c) Result after 9 iterations. (d) Result after 13 iterations. (e) Final result assuming that the “3”s in (d) are scanned in raster order. (f) Final result assuming that the “3”s in (d) are scanned in anti-raster order.](image)
Growing process. After all the pixels have been labelled, the tied pixels are independently re-examined to see whether or not the ties can be resolved. Any remaining ties can of course be resolved by imposing additional assignment criteria if required; e.g., assigning the tied pixel to the largest region, and failing this, assigning it to the region with the larger mean, and so on. However, this constitutes a post-processing step and is not part of the improved seeded region growing algorithm. The behaviour of the ISRG algorithm is illustrated in Fig. 2. Parallel processing ensures that all pixels of the same priority are processed on an equal basis. Consequently the "3"s in Fig. 1(d) are shared equally between the top-left region and the bottom-right region.

4. Implementation

Our implementation of the ISRG algorithm utilises an ascending priority queue (PQ), and several LIFO queues. In contrast to the more familiar LIFO queue, the elements of an ascending priority queue are ordered from smallest to largest. A new element can be inserted arbitrarily but only the smallest element can be removed (Tenenbaum and Augenstein, 1986). In our implementation the elements of the PQ are LIFO queues. Each LIFO queue contains pixels at a specific priority; i.e., with the same $\delta$ value. When a new pixel is inserted into the PQ it is added to the LIFO queue corresponding to the pixel’s $\delta$ value. Instead of removing individual pixels from the PQ, the entire LIFO queue corresponding to the smallest $\delta$ (highest priority) is removed. This permits the processing of all the pixels at the highest priority at the same time. As each pixel is removed from the highest priority LIFO queue its label is determined and inserted into a LIFO queue of labels (LQ), and the pixel is inserted into a LIFO holding queue (HQ). A pixel’s label is determined by examining its neighbours. If those neighbours that possess a region label all have the same label then the pixel is deemed also to have this label. If, however, the pixel is surrounded by neighbours with several different region labels then the pixel is deemed to have the label of the neighbour that determines the minimum $\delta$ value. In the event of a tie the pixel is marked as tied and inserted into the priority queue with an infinite (in reality just a very large) $\delta$ value. This guarantees that a second attempt at resolving ties is made after all other pixels have been labelled. Any pixels that are still tied remain unclassified (the ties can be resolved as a post-processing step if required). Once all of the pixels in the highest priority LIFO queue have been processed, they are labelled. This is done by successively removing a label and a pixel from the LQ and HQ respectively, and assigning the label to the pixel. Thus all the pixels at the highest priority are labelled independently. To overcome scan order and neighbourhood order dependencies, previous entries in the PQ must be updated whenever region means are updated. Our implementation achieves the required functionality in the following way. As each pixel is labelled, any of its neighbours that are either unlabelled or marked as being in the PQ are inserted – once only – into a LIFO neighbours holding queue (NHQ). After labelling has completed each pixel is successively removed from the NHQ and inserted into the PQ. Thus a single pixel can be inserted into the PQ more than once. However, in the event that a pixel removed from the highest priority LIFO queue – removed from the PQ – already has a region label then it is not relabelled.

\footnotesize
\begin{itemize}
  \item[2] Refer to Breen and Monro (1994) for a discussion and evaluation of different data structures for implementing a priority queue.
\end{itemize}
The pseudo code for our implementation is as follows:

```
PQ  Ascending priority queue: list of LIFO queues ordered according to δ.
FQ  First (highest priority) queue in the PQ.
NHQ Neighbours holding queue: used to hold pixels that neighbour one or more regions.
HQ  Holding queue: used to accumulate pixels removed from the NHQ.
LQ  Labels queue: used to hold region labels corresponding to the pixels in the HQ.
REGION_MEAN[]
    An array used to hold the current mean grey value for each region.
DELTA
    ABS(REGION_MEAN[j] − grey_value(pixel)).
TIED Label denoting a tied pixel.
INQUEUE Label indicating that the pixel is in the HQ or NHQ.
IN_PRIORITY_QUEUE
    Label indicating that the pixel is in the PQ.
```

label seed regions and initialise REGION_MEAN[]
add the pixels neighbouring the seed regions to the NHQ and label them as IN_QUEUE
while the PQ is not empty or the NHQ is not empty
    while the NHQ is not empty
        remove pixel from the NHQ
        examine all of its neighbours to find the minimum DELTA
        insert the pixel into the PQ with priority DELTA
        add pixel to the HQ and label it as IN_QUEUE
    endwhile
    if the PQ is not empty
        remove FQ from the PQ
        while the FQ is not empty
            remove pixel from the FQ
            if pixel has the label IN_PRIORITY_QUEUE or the label TIED
                examine all of its neighbours that have a region label
                if they all have the same label
                    add this label to the LQ
            else
                examine all of the pixel’s neighbours to find the minimum DELTA
                if there is no tie
                    add corresponding region label to the LQ
                else
                    add TIED to the LQ
                    if the pixel is not already labelled TIED
                        insert pixel into the PQ with a priority of ∞
                    endif
                endif
            endif
        endwhile
        if the label is not TIED
            update REGION_MEAN[label]
            examine the pixel’s neighbours
            if a neighbour is unlabelled or has the label IN_PRIORITY_QUEUE
                add it to the NHQ and label it as IN_QUEUE
            endif
        endif
    endif
while the HQ is not empty
    remove label from the LQ
    remove pixel from the HQ
    assign label to pixel
    if the label is not TIED
        update REGION_MEAN[label]
    endif
endwhile
```

4.1. A comment on boundary flagging

The ISRG algorithm produces an image in which each individual region has a unique numeric label. If desired, for display purposes, it is a relatively straightforward task to flag a single pixel wide boundary between regions. This can be done either within the algorithm or as a post-processing step. Unresolved ties are of course natural candidates for boundary flagging. Where two or more regions abut, however, an arbitrary decision has to be made as to which region’s border pixel should be flagged as a boundary pixel. This introduces an order dependency. The
implication of this is that there may be slight discrepancies in the borders produced by a raster order processing as opposed to an anti-raster order processing.

5. Discussion and conclusion

The improved seeded region growing algorithm we have proposed offers the same benefits as the algorithm proposed by Adams and Bischof (1994) but with the added advantage of pixel order independence. The algorithm was motivated by the need to accurately segment the chromatin within images of cell nuclei. The limited pixel resolution of these images – e.g. Fig. 3(a) is 88H × 85V pixels – and the similarity of adjacent chromatin clumps meant that the SRG algorithm produced markedly different segmentations for raster order and anti-raster order processing (see Fig. 3, (c) and (d)). The proposed ISRG algorithm, however, because it is not dependent on the order of pixel processing, produces a consistent segmentation (see Fig. 3, (e) and (f)). The added complexity of the ISRG algorithm, as compared to the SRG algorithm, is reflected in increased execution time when implemented on a computer. We have implemented both the SRG algorithm and the ISRG algorithm in C on a DEC3000 workstation. To counter floating point imprecision we defined a threshold difference below which two δ values are deemed to be the same. The priority queue is implemented using a binary tree. For a 256 × 256 8-bit test image and 4 small seeds, the execution time for the SRG implementation was 3 seconds whilst that for the ISRG implementation was 15 seconds. Further optimisation of our implementation of the ISRG algorithm is possible. For example, according to Breen and Monro (1994), the speed performance of the priority queue – and hence the ISRG implementation – is improved if a SplayQ data structure is used.

Fig. 3. Order independence of the ISRG implementation. (a) High resolution micrograph of the nucleus of a cell. (b) Seed regions (local extrema). (c) SRG implementation applied to (a) in raster order (Adams and Bischof, 1994) (p. 643). (d) SRG implementation applied to (a) in anti-raster order. A careful comparison with (c) reveals significant differences. (e) ISRG implementation (boundary/tied pixels shown in white) applied to (a) in raster order. (f) ISRG implementation applied to (a) in anti-raster order. Minor discrepancies between (e) and (f) are an artifact of boundary flagging.
References


