Fast Region Merging Algorithms for Image Segmentation

David J. Crisp and Trevor C. Tao
Surveillance Systems Division - 71 Labs
Defence Science and Technology Organisation
Salisbury, South Australia 5108, Australia
david.crisp@dsto.defence.gov.au

Abstract

One of the more successful approaches to image segmentation involves formulating the problem as the minimisation of a Mumford-Shah functional and then using region merging algorithms to approximate the minimiser. Recent work by Redding et al. presented such an algorithm and demonstrated the quality of the segmentations it produces. Here, we extend that work in two ways. First, we describe a variation of the algorithm which leads to the same segmentations but which is an order of magnitude faster to compute. This gain is made by introducing the concept of a locally best merge. Second, and as a natural consequence of considering locally best merges, we describe a new algorithm. While the new algorithm is no longer optimal, it is faster and has other desirable properties. In particular, in the case where the functional to be minimised takes account of region variances as well as means, the new algorithm overcomes the problem of unreliable variance estimates for small regions.

1. Introduction

The Mumford-Shah functional we will be dealing with is discussed in Chapter 5 of the book, [4], by Morel and Solimini. They take an image to be a function $g(x, y)$ of the continuous variables x and y defined on a domain Ω. A segmentation of g, in its most general form, is a collection K of boundaries which partition Ω into a set of regions such that g is "homogeneous" on each. For Mumford-Shah functionals, the term "homogeneous" is given precise meaning by specifying an image model $u(x, y)$. We use the simplest model whereby u is restricted to be constant on each of the segmentation regions. The associated functional is

$$E(u, K) = \int_{\Omega \setminus K} (u(x, y) - g(x, y))^2 dx\, dy + \lambda \ell(K) \quad (1)$$

where $\ell(K)$ is the total length of K and λ is a regularisation parameter. The choice of λ controls the tradeoff between how well the model u fits the image g and the complexity of the boundary K. It must be pre-specified. The suitability of (1) for image segmentation is clearly shown by Morel and Solimini's mathematical analysis.

By design, $E(u, K)$ is small for good segmentations and thus the segmentation problem is to find the minimisers of $E(u, K)$. This problem can be simplified by first observing that a minimiser is fully specified by its boundary K alone. The real problem then is to find the boundary K of the minimiser. While there are many algorithms for doing this region merging is a good choice, see [3] and [4].

Two questions need addressing in region merging algorithms: which segmentation to start from and how to choose the regions to be merged. In answer to the first question, Koepfler *et al.* provide sound arguments for starting with the trivial segmentation. (The **trivial segmentation** is the one in which each pixel is a separate region.) The second question is more interesting. It accounts for the differences between our algorithms and those of Koepfler et al., [3], and others [2]. It is answering this question that underpins the work described here. We will present two different answers.

Our first answer was motivated by attempts to speed up the **full λ-schedule algorithm (FLSA)** reported on in [5]. We have been highly successful in those attempts and have decreased the computation times by an order of magnitude. We call our new version of the algorithm the **optimal locally best merging (OLBM)** algorithm. We emphasis that it produces the same segmentations as FLSA. The speed up is achieved in the search for the next pair of regions to merge - instead of searching the full list of all potential merges we only search the list of locally best merges.

Our second answer, results in a new algorithm which we call the **synchronous locally best merging (SLBM)** algorithm. The idea behind this algorithm is to perform all locally best merges synchronously rather than searching for the best one. While SLBM is not as accurate as OLBM, it is somewhat faster and has other desirable properties.

2. Discretising the functional

In order to apply (1) to digital images it needs to be discretised. To do so, we take Ω to be a set of pixels indexed by a discrete variable $i = 1, \ldots, n$. The image g and its model u are then defined by their values $g(i)$ and $u(i)$ at each pixel. A segmentation region is a connected sub-set of Ω and a segmentation K is a partition of Ω into regions. The boundary of K is the set of pixel edges which separate the regions and its length $\ell(K)$ is the total number of edges in the set. With this notation, the functional (1) becomes

$$E(u, K) = \sum_{i=1}^{n} (u(i) - g(i))^2 + \lambda \ell(K). \qquad (2)$$

Further, since we are only interested in the minimisers of (2) and u is piecewise constant, we can assume

$$u(k) = \frac{1}{|O|} \sum_{i \in O} g(i) \qquad (3)$$

where O is the region of K with $k \in O$ and $|O|$ is its area.

3. Full λ-schedule algorithm (FLSA)

The full λ-schedule algorithm reported on by Redding *et al.* in [5] was an extension of the region merging algorithm developed by Koepfler *et al.*, [3]. At its heart, the Koepfler *et al.* algorithm has two components: a simple strategy for searching for candidate pairs of regions to be merged and a criterion for deciding whether or not to merge them. The search strategy is neither sophisticated nor optimal. However, it is fast and some of its deficiencies were addressed by a clever extension of the basic algorithm. The extension involves choosing an increasing sequence of values

$$0 < \lambda_1 < \lambda_2 < \ldots < \lambda_L \qquad (4)$$

to be used for the regularisation parameter in (2). For each value λ_i the basic algorithm produces a segmentation K_i which "minimises" the functional (2) with $\lambda = \lambda_i$. These segmentations form a chain with K_i being used as the starting point for the algorithm which produces K_{i+1}. The chain begins with K_0 being the trivial segmentation. The final segmentation K_L is the output and so λ_L should reflect the final amount of regularisation required.

The sequence (4) is called a **λ-schedule**. Its effect is to limit the the mistakes made due to the poor searching strategy. Thus, it is natural, as was done in [5], to consider the extreme of choosing the λ-schedule so that each segmentation, K_i, differs from the previous one, K_{i-1}, by exactly one region merge. This schedule was called the **full λ-schedule**. Implementing the full λ-schedule using the algorithm of Koepfler *et al.* is not practical due to the inefficiency of their search strategy. However, an algorithm with an efficient implementation is possible and was described in [5]. It was called the **full λ-schedule algorithm (FLSA)**.

To describe FLSA we first need to describe how the full λ-schedule is calculated. We assume we have the segmentation K_i and we will describe how to calculate λ_{i+1}. Let (O_i, O_j) be a pair of neighbouring regions in K_i. The Koepfler *et al.* criterion for merging this pair is that the functional (2) be decreased by doing so. In other words, the merging criterion is that $E(K \backslash \partial(O_i, O_j)) - E(K) < 0$ where $\partial(O_i, O_j)$ is that part of the boundary K which separates O_i and O_j. In [5], it was shown that an equivalent criterion is that $\lambda > c(O_i, O_j)$ where

$$c(O_i, O_j) = \frac{|O_i||O_j|}{|O_i| + |O_j|} \, \frac{(u_i - u_j)^2}{\ell(\partial(O_i, O_j))}. \qquad (5)$$

Here $|O_i|$ and $|O_j|$ denote the areas of O_i and O_j, and u_i and u_j are the average values of g on O_i and O_j, respectively. We refer to $c(O_i, O_j)$ as the **merge cost** for the pair (O_i, O_j). Clearly then, the desired value of λ_{i+1} is the minimum merge cost amongst all neighbouring pairs in K_i. Further, the segmentation K_{i+1} is obtained from K_i by merging the pair with the minimum merge cost.

Thus the idea underpinning FLSA is to maintain a list of all potential region mergers and to calculate the associated costs according to (5). At each step of the algorithm the cheapest merger is performed. This merging process continues until the desired **stopping lambda**, λ_L is reached (that is, until all merge costs are greater than λ_L). We should remark here that choosing the correct stopping lambda is an important question and although the paper, [5], suggested one answer, we believe a better method is needed. An example of the output of FLSA is shown in Figure 1.

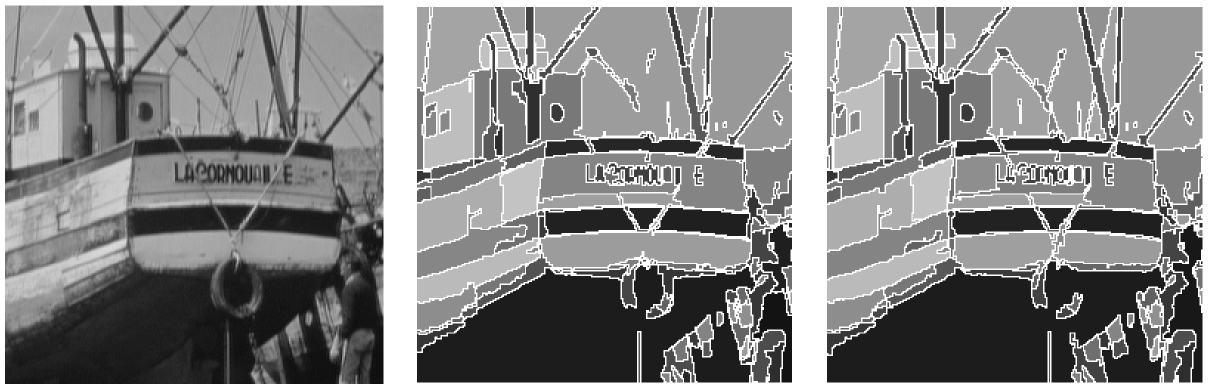
In order to describe our improvements to FLSA we will need to briefly delve into its efficient implementation, as presented in [5]. The implementation depends on maintaining two main lists: the **region list** $\mathcal{R} = R_1, R_2, R_3, \ldots$ and the **region pair (or boundary) list** $\mathcal{B} = B_1, B_2, B_3, \ldots$. The region pair list $\mathcal{B}$ is the list of all potential mergers and we describe it first. Its k-th entry refers to the k-th pair of neighbouring regions or equivalently, the k-th boundary component of the segmentation. If O_i and O_j are the regions concerned then B_k is of the form

$$B_k = (i, j, b_{ij}, c_{ij}), \qquad (6)$$

where $b_{ij} = \ell(\partial(O_i, O_j))$ is the length of the separating boundary and $c_{ij} = c(O_i, O_j)$ is the merge cost as defined in (5). The i-th entry of $\mathcal{R}$ refers to region O_i and is of the form

$$R_i = (a_i, u_i, N_i, P_i), \qquad (7)$$

where a_i is the area of O_i, u_i is its average gray-scale value, $N_i = \{O_{i_1}, \ldots, O_{i_l}\}$ is the list of neighbours of O_i, and



(a) Original image (b) FLSA segmentation (c) SLBM segmentation

Figure 1. Segmentation of the image used by Koepfler *et al.* in [3]. The optimal locally best merge (OLBM) algorithm segmentation is the same as the full λ-schedule algorithm (FLSA) segmentation and so is not illustrated. In both cases the stopping lambda λ_L was chosen so that 200 region remain.

$P_i = \{p_{i_1}, \ldots, p_{i_l}\}$ is the list of corresponding pair set indices for the neighbours. Sorting the complete list $\mathcal{B}$ to find the cheapest merge cost is cumbersome and instead a third list $\mathcal{E}$, called the **merge cost list**, is introduced. It consists solely of the merge costs together with their index in $\mathcal{B}$.

FLSA is then a process of finding the cheapest merge cost in $\mathcal{E}$, performing the merge and updating all three lists. This can be done efficiently as follows. First, $\mathcal{E}$ is sorted using red-black trees. Second, there are simple update formulae (including (5)) for all the quantities involved, see [3] and [5]. Third, the region and boundary components which are affected by the merge are easily found using the complex linking between the lists whereby the entries in $\mathcal{B}$ refer to indexes of $\mathcal{R}$ and *visa-versa*. Full details are given in [5]. Here we merely comment that if O_i and O_j are being merged then the lists N_i and N_j tell us which regions are neighbours and the lists P_i and P_j tell us where the corresponding boundary components are. Further, it is only these regions and boundary components which are affected by the merge and need updating.

4. Optimal locally best merging (OLBM)

Despite the use of red-black trees, sorting the merge cost list $\mathcal{E}$ is still the most expensive computational element of FLSA. The list can be very long (for a rectangular image it is initially approximately twice the number of image pixels) and it needs resorting each time a merge is performed. Moreover, in general, after each merge more than one entry will be out of place. In this section we report on a new idea which reduces the size of $\mathcal{E}$ and thereby makes the sorting process much more efficient.

Our idea is that, since a region merge operation only affects the merge costs in a local area, it makes sense to sort the affected merge costs first and then only enter the best (cheapest) ones into the merge cost list $\mathcal{E}$. In order for this idea to work though, some care is needed in defining what is meant by a locally best merge. We need to ensure that the set of locally best merges is easily updated and that the globally best merge is included. To do this, we first need to be precise about what is meant by best and globally best.

Given a set of potential region merges, the **best merge** is the one with the smallest merge cost and given a segmentation, the **globally best merge** is the best amongst all possible merges. We resolve the ambiguity in the case of tied merge costs by choosing the **best merge** to be the one with the smallest index in the region pair list $\mathcal{B}$. We can now give our main definition. We say a neighbouring pair of regions (O_i, O_j) is a **locally best merge** if it is the best merge in the set of all merges which involve either O_i or O_j (or both). The idea behind the OLBM algorithm then, is to prune the merge cost list $\mathcal{E}$ so that it only contains locally best merges. It is clear from the definitions that the globally best merge is not removed by this pruning process and it follows that the OLBM algorithm will produce exactly the same segmentations as FLSA.

As an indication of the savings the OLBM algorithm leads to, observe that at any stage in the segmentation process the number of locally best merges is at most half the number of regions (since each locally best merge accounts for two regions), whereas the total number of pos-

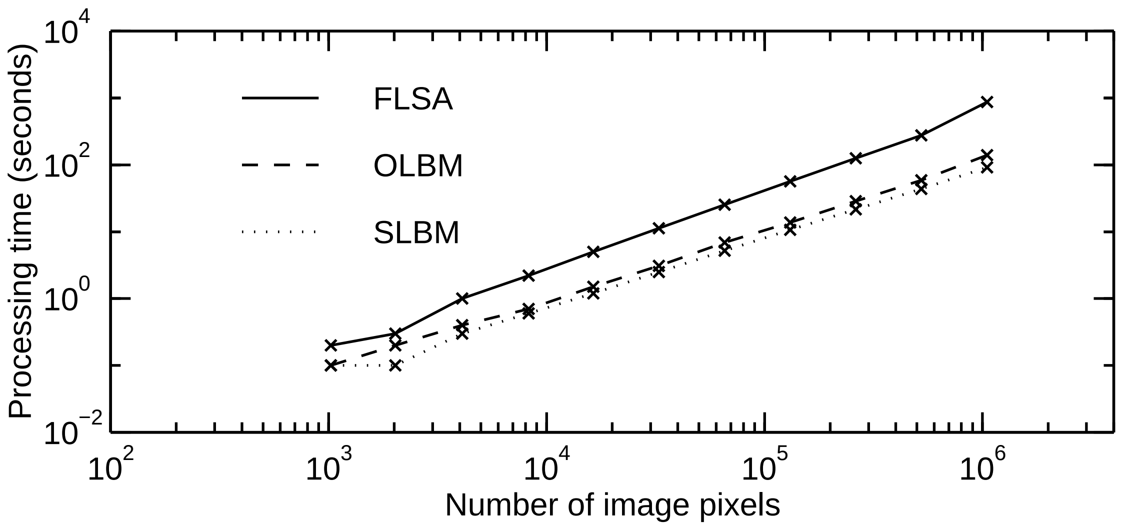


Figure 2. A comparison of the computation times for the full λ-schedule algorithm (FLSA), the optimal locally best merge (OLBM) algorithm and the synchronous locally best merge (SLBM) algorithm.

sible merges is greater than the number of regions less one (since each merge reduces the number of regions by one and merging can continue until only one region is left). Thus by using locally best merges only, we are guaranteed of at least halving the size of the $\mathcal{E}$ list.

It only remains to show that the locally best merges are easily found and hence the pruned merge list is easily updated. Returning to the definition, it is evident that if (O_i, O_j) is a locally best merge then O_j is the best neighbour for O_i to merge with and *visa-versa*. It follows that we can find all the locally best merges by first scanning the list of regions and for each one determining its best merging neighbour. By reviewing the list and looking for instances where the best merging neighbour of a region says likewise that the original region is its best merge, we can find all the locally best merges. Note that, while this provides the thinking behind our implementation, we make the search much more efficient by using our linked data structures $\mathcal{R}$ and $\mathcal{B}$.

Our implementation of OLBM requires an extra variable in each entry of the region list $\mathcal{R}$ and two flags in each entry of the region pair list $\mathcal{B}$. Thus the entries now have the form

$$R_i = (a_i, u_i, N_i, P_i, \mathit{bestp}_i) \tag{8}$$

$$B_k = (i, j, b_{ij}, c_{ij}, \mathit{bflag}_{ij}, \mathit{eflag}_{ij}). \tag{9}$$

The new item, bestp_i, in (8) records the index in $\mathcal{B}$ of the pair (O_i, O_k) where O_k is the best merge for O_i. Region O_k is found by using the list P_i to search the merge cost information in $\mathcal{B}$ for O_i's best merge.

The new item, eflag_{ij}, in (9) records whether or not the pair (O_i, O_j) is a locally best merge and hence is used to maintain the pruned merge cost list $\mathcal{E}$. It is set when both O_i and O_j say the other is its best merging neighbour. To help perform this check, we use the other new item, bflag_{ij}, in (9). This second flag is set when at least one of O_i and O_j says the other is its best merge. Note that if bflag_{ij} is set but eflag_{ij} is cleared then only one of O_i and O_j is saying the other is its best merge. In this case, we can only tell which one that is by examining bestp_i and bestp_j.

We now describe how the OLBM algorithm updates the new quantities after each merge operation. Fortunately, these new updates are essentially independent of the old ones and we do not need to discuss the full details of FLSA. In the following, we assume the regions O_i and O_j are being merged to form O_{ij} and the FLSA updates have already been done. As with FLSA we make good use of the linking in our data structures to obtain speed and efficiency.

OLBM Extension of the FLSA Algorithm

1. Use P_{ij} to process O_{ij}'s neighbours O_k as follows:
   (a) Use *bestp* for O_k to find its old best merge, O_r.
   (b) If $O_r = O_i$ or $O_r = O_j$ then the FLSA updates ensure $(O_k, O_r) = (O_{ij}, O_k)$, so:
      i. Clear *bflag* for the pair (O_{ij}, O_k). (Since (O_i, O_j) was locally best *eflag* is not set.)
      ii. Completely recalculate O_k's best merge by using P_k to scan the merge costs of its neighbours and let O_s be the cheapest.
      iii. Set *bestp* for O_k equal to the pair set index of the pair (O_k, O_s).
      iv. If *bflag* for the pair (O_k, O_s) is cleared then set it and set *eflag* otherwise.

(c) If $O_r \neq O_i$ and $O_r \neq O_j$ then see if O_{ij} is a better merge for O_k than O_r. If so, then:

  i. Clear *eflag* for the pair (O_k, O_r) if it is set and clear *bflag* otherwise.

  ii. Set *bestp* for O_k equal to the pair set index of (O_{ij}, O_k).

  iii. Set *bflag* for the pair (O_{ij}, O_k).

2. Process the region O_{ij} as follows:

  (a) Use P_{ij} to finding its best merge, O_t.

  (b) Set *bestp* for O_{ij} equal to the index of (O_{ij}, O_t).

  (c) If *bflag* for the pair (O_{ij}, O_t) is cleared then set it and set *eflag* otherwise.

This completes our description of the OLBM algorithm. Evidence of the decrease in computation time it affords over the FLSA algorithm is shown in Figure 2. While such results vary depending on the nature of the image being segmented we have found that across a wide range of image types, the computation times of OLBM are consistently an order of magnitude less than those of FLSA.

5. Synchronous locally best merging (SLBM)

The merge cost (5) can be thought of as measuring the significance of the difference between the pair of regions (O_i, O_j). The smaller that difference is, the smaller the merge cost and *visa-versa*. It follows that the regions in a locally best merging pair are separated by the locally least significant image structures. Consequently, we propose the **synchronous locally best merging (SLBM)** algorithm whereby all locally best merges are performed "synchronously" rather than sequentially. By "synchronously", we mean "at the same time".

Since our SLBM algorithm is designed for a serial computer we don't actually execute the merges synchronously. Instead, we obtain the same result by maintaining a queue containing all the locally best merges and we work our way through it sequentially. Obviously this can be done in a batch fashion: the queue is filled with all the current locally best merges; it is processed until empty; and then it is re-filled with the new batch of locally best merges and so on. Surprisingly though, this can also be done in a continuous fashion as happens with OLBM. In fact, only two main changes are needed to convert OLBM into the SLBM.

The first change is that in place of the merge list $\mathcal{E}$ which is sorted according to merge cost, we use a queue $\mathcal{Q}$ which is sorted according to "time of arrival". As with OLBM, a neighbouring pair of regions is added to $\mathcal{Q}$ if and as soon as its *eflag* is set. The second change involves Step 1c of the OLBM algorithm. This step is the only place in the OLBM algorithm where the current merge operation can clear an *eflag* and hence destroy an existing locally best merge. SLBM uses the following alternative:

SLBM Adaption of the OLBM Algorithm

(c) If $O_r \neq O_i$ and $O_r \neq O_j$ then see if O_{ij} is a better merge for O_k than O_r. If so and if also *eflag* for the pair (O_k, O_r) is not set then:

  (a) Clear *bflag* for the pair (O_k, O_r).

  (b) Set *bestp* for O_k equal to the index of (O_{ij}, O_k).

  (c) Set *bflag* for the pair (O_{ij}, O_k).

As with OLBM, the SLBM algorithm is run until all locally best merge costs are greater than the stopping lambda, λ_L.

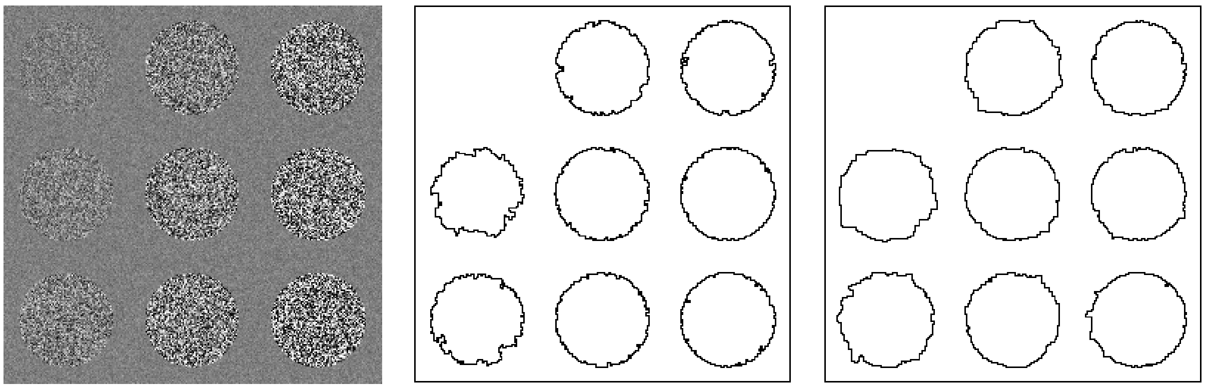
To prove that the SLBM algorithm works, we need to examine the changes to Step 1c more closely. Instead of always executing Steps 1(c)i, 1(c)ii and 1(c)iii of the OLBM algorithm, SLBM only executes them if (O_k, O_r) is not a locally best merge. This means that the SLBM algorithm never destroys a previously existing locally best merge which was our aim. However, it also means that SLBM allows some of the variables *bestp, bflag* and *eflag* to become corrupted. Clearly, the immediate corruption is repaired when the merge (O_k, O_r) finally takes place since then the corrupted components either become redundant or are re-calculated. To complete the proof, we also need to check that the corruption does not spread in the meantime. While this is not trivial, the details are straight forward and we omit them due to a lack of space.

From the timing data in Figure 2, it can be seen that SLBM is faster than OLBM and from the segmentation results in Figure 1 it can be seen that SLBM has similar accuracy. Objectively assessing accuracy requires the use of benchmark segmentation problems and proper accuracy measures We hope to do this in the future, but for the moment, our empirical results suggest that SLBM is not always as accurate as OLBM.

One way of limiting the errors SLBM makes is to stop it early and finish with OLBM. We have had good results with this approach. Another way is to use Koepfler et al.'s trick of introducing a $lambda$-schedule. We have not tried this yet. However, SLBM produces good segmentations without a $lambda$-schedule and hence is a better algorithm than Koepfler et al.'s. If a schedule was to be used we expect it would not need to be very long.

6. An advantage of SLBM

We claim that in general locally best merges will be "common" and "spread evenly" across images. This claim can be justified by considering **chains of best merges**. By this we mean a sequence $O_1, O_2, O_3, \ldots$ such that O_{i+1} is



(a) Original image (b) OLBM segmentation (c) SLBM-OLBM segmentation

Figure 3. Segmentation of a difficult image. Since the differences between the regions lie only in the noise variance, the Mumford-Shah functional is not suitable and the MAP functional of [1] was used instead. In (c), SLBM was only used in the initial stages and the process was completed with OLBM.

the best merging neighbour of O_i for all i. It is not hard to prove that each chain ends at a locally best merge. Moreover, as a chain lengthens the merge costs decrease and so the chain is unlikely to cross significant image boundaries. This justifies (if not proves) our claim. It follows that, SLBM will tend to spread region growth "evenly" across images. We explain why this feature is desirable next.

In recent work, [1], it was shown that a Bayesian setting can be used to re-interpret the Mumford-Shah functional. This setting lead to a MAP functional for segmentation which takes account of region variances as well as means. It was further shown that the FLSA algorithm could be used to approximate its minimisers. The only change required is to replace the merge cost formula (5) with

$$c(O_i, O_j) = \frac{|O_{ij}| \log \sigma_{ij}^2 - |O_i| \log \sigma_i^2 - |O_j| \log \sigma_j^2}{2\, \ell(\partial(O_i, O_j))}$$

where σ_i^2 is the variance over region O_i of the noise $\epsilon(x, y) = g(x, y) - u(x, y)$. However, poor variance estimates for small regions meant the algorithm did not perform well. Similar problems were noted in [2].

In [1], a hybrid update formulae was suggested for dealing with this problem. We have since found a more convenient method. It involves simply initialising the single pixel region variances at some small positive value. The variance of each merged region O_{ij} is then estimated from those of O_i and O_j using the standard update formulae for combined samples. Further help is provided by using SLBM to initialise the segmentation process, see Figure (3). We surmise that the explanation is SLBM's tendency to spread region growth "evenly" across images.

7. Conclusion

The OLBM algorithm affords an order of magnitude speed increase over FLSA while producing exactly the same segmentations. The SLBM algorithm offers a further speed increase but with some loss of accuracy. Unless speed is critical, we recommend SLBM be used only to initialise the segmentation process. On the up side, SLBM helps overcome initialisation problems for MAP functionals. On the down side, it involves choosing an additional parameter to specify when SLBM should stop and OLBM continue.

References

[1] D. Crisp and G. Newsam. A fast, efficient segmentation algorithm based on region merging. In *Proceedings of Image and Vision Computing New Zealand 2000 (IVCNZ'00)*, pages 180–185, 2000. Hamilton, New Zealand, November 2000.

[2] T. Kanungo, B. Dom, W. Niblack, and D. Steele. A fast algorithm for mdl-based multi-band image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 609–616, 1994. Seattle, Washington, June 1994.

[3] G. Koepfler, C. Lopez, and J.-M. Morel. A multiscale algorithm for image segmentation by variational method. *SIAM J. Numer. Anal.*, 31:282–299, 1994.

[4] J.-M. Morel and S. Solimini. *Variational Methods in Image Segmentation*. Birkhäuser, Boston, 1995.

[5] N. Redding, D. Crisp, D. Tang, and G. Newsam. An efficient algorithm for mumford-shah segmentation and its application to sar imagery. In *Digital Image Computing: Techniques& Applications (DICTA'99)*, pages 35–41, 1999. Perth, Australia, December 1999.