

Fast Segmentation of Large Images

David J. Crisp†

Peter Perry‡

Nicholas J. Redding†

†Intelligence, Surveillance & Reconnaissance Division

Defence Science & Technology Organisation,
PO Box 1500, Edinburgh, South Australia 5111,

Email: david.crisp@dsto.defence.gov.au

Email: nick.redding@dsto.defence.gov.au

‡SYDAC Pty. Ltd., 113-115 King William Street, Adelaide S.A. 5000.

Email: peter.perry@dsto.defence.gov.au

Abstract

The processing of large images is a generic problem in wide area surveillance. An important difficulty is that many image processing algorithms are global rather than local and hence can be infeasible due to the required computing time or memory resources when processing very large images. Consequently there is a need to break such image processing problems into smaller pieces. A similar need also arises when the requirement is to process the imagery on-line as it is being collected. Here, we consider the particular problem of image segmentation. The approach we take is to divide the large image into smaller overlapping tiles. We segment each tile separately and then patch the results together. The main contribution of this paper is our answer to the question of how to handle ambiguities in the overlapping areas. Our solution is applicable to any segmentation algorithm which is based on region merging. The particular algorithm we use is known as the Full Lambda Schedule Algorithm (FLSA). We also include a description of some refinements to the original algorithm which provide speed and efficiency improvements.

Keywords: image, segmentation, algorithm.

1 Introduction

In a suite of operational target detection and recognition algorithms we have under development called the Analysts' Detection Support System (ADSS), there is a requirement to segment very large images (in excess of 100 megabytes in size) as part of a processing chain. The requirement of this environment is the rapid detection of objects of interest. This necessitates on-line processing of the image data as it becomes progressively available, and the desire is to make the segmented imagery likewise available on-line with the time lag minimised.

The full lambda schedule algorithm (FLSA) was developed as a fast and efficient method for segmentation of imagery as one early step in a sequence of detection and recognition stages. Initially, it was developed for synthetic aperture radar (SAR) imagery, but it has general applicability. Despite the fact that the FLSA algorithm is mathematically elegant and can be used to perfectly segment arbitrarily large images (given sufficient processing power and memory), it is required that the segmentation be done in a progressive manner because of the extremely large size of the imagery of interest and the need to commence the processing of the imagery before the sensor has finished collecting it. Here we consider the problem of how to apply the FLSA to very large images, although

the method we present here for handling very large images is not specific to this algorithm and could be used by any segmentation algorithm that uses region merging.

2 The FLSA Image Segmentation Algorithm

The Full Lambda Schedule Algorithm (FLSA) of Redding *et al.* (Redding & Crisp & Tang & Newsam 1999) for image segmentation is a highly efficient means of finding the segmentation boundary which best partitions the image into homogeneous regions. It views the image as an underlying model which has been corrupted by noise and not only segments the image into separate regions but also fits an image model within each region. This is done by using a Mumford-Shah energy functional. Such functionals provide a measure of the "goodness" of a segmentation by trading off the total model fitting error across all regions against the total complexity of the boundary. According to this view point, a good segmentation not only allows the image model to be fitted with small total error but also has a simple boundary. Thus the aim is to find the segmentation which minimises the functional. In the book, (Morel & Solomoni 1995), the minimisers of Mumford-Shah functionals are studied in depth and it is proven that they have many desirable properties of segmentations.

In the Mumford-Shah energy functional, the trade-off of model fitting error against boundary complexity is controlled by a regularisation parameter which we denote by λ . The parameter λ is applied as a weighting to the boundary complexity term. Thus, if λ is increased, the penalty on boundary complexity is increased and coarser segmentations result. Conversely, if λ is decreased, the penalty on boundary complexity is reduced and finer segmentations result. We view λ as a scale parameter and choose it to suit the segmentation task at hand.

The FLSA uses a region merging approach to search for minimisers of the Mumford-Shah functional. It begins with the trivial segmentation in which each pixel is considered to be a separate region and proceeds to merge regions with the aim of minimising the Mumford-Shah functional (although not in a greedy manner). The way this is done is explained fully in the papers (Redding & Crisp & Tang & Newsam 1999) and (Crisp & Tao 2002). (See also the paper by Koepfler *et al.* (Koepfler & Lopez & Morel 1994) which provided the original motivation for the FLSA.) Here we merely comment that the Mumford-Shah functional can be used to define a cost for merging a pair of neighbouring regions and that at each step the FLSA algorithm merges the pair of regions which has the smallest merge cost. The algorithm stops when the cheapest merge cost exceeds the value of λ . Note that, while the FLSA algorithm is not guaranteed to find global minimisers of the

Mumford-Shah functional experience has shown that it does produce good local minimisers.

For convenience and robustness we consider only the simplest image model whereby the image is assumed to be piecewise constant on each region. We also assume the corrupting noise is Gaussian with constant variance across the image. In this case fitting the image model with least error is easy; in each region, the value of the image model is simply the average of the region pixel values. Variations on this basic framework are possible and the theory and algorithms can be extended to allow piecewise polynomial image models and noise variances which vary from region to region. The algorithm will also handle multiple image layers (as is the case with colour images for instance).

The algorithm derives its speed and efficiency through the use of sophisticated data structures which allow the neighbours of each region to be easily located and the merge costs to be easily calculated. Note that after each merge operation only the merge costs in the local area are effected and the FLSA data structures allow these to be quickly updated. One of the more computationally expensive aspects of the FLSA algorithm is finding the cheapest merge cost – a sorted list must be maintained. Fortunately, it has been observed that not all merge costs need be entered in the list and in fact only those for which both regions of the pair consider the other its best merge choice need be entered. The consequent reduction in processing time is significant, see (Crisp & Tao 2002). While the algorithmic complexity can only be proved to be at worst $O(n^2 \log n)$ where n is the total number of pixels in the image, it is believed to be $O(n \log n)$, (Redding & Crisp 1999).

Given that the FLSA uses sophisticated data structures to obtain speed and efficiency, it should be no surprise that the memory usage of even an efficient implementation is large. An image of 512×1024 pixels requires in excess of 100Megabytes of memory to represent its data structures. Note further, that because the major structures are trees and lists, they do not localise well, so if the amount of actual memory available is insufficient to keep all of the data memory resident, the speed deteriorates markedly. This provides further motivation for breaking large image segmentation problems into smaller pieces.

The algorithm has as its fundamental output a region map, where the regions are assigned arbitrary distinct numbers, and the values assigned to each pixel is its region number. For practical purposes, 32-bit numbers are used and for large images it is possible that the counter will wrap around. However, the probability of two adjacent pixels from different regions having the same region number is extremely small. Of course for image analysis and interpretation purposes, the underlying image model is also very useful. With this in mind, the algorithm has been designed to also output an image where each pixel is assigned its image model value (which, under our simplifying assumptions is just the region mean value) at no extra cost. An example of the segmentation result can be seen in Figure (1).

3 FLSA Speed Considerations

The algorithm spends much of its time maintaining the major data structures, (Redding & Crisp & Tang & Newsam 1999). The merge-candidate list is maintained as a red-black tree; the $O(n \log n)$ time for operations on this is one of the limiting factors in the speed of the algorithm.

A list of the edges of each region, ordered by region number, is also maintained; most of the operations on



(a) The original image



(b) Segmentation result

Figure 1: Segmentation of a SAR image using the FLSA algorithm. Acknowledgement: Image kindly supplied by InfoSAR Ltd.

these list can be cast in the form “for each element do ...” so that the ordering is of no benefit, but one operation involves the removal of one element from the list and its replacement with another. On average, these lists have between four and six elements (it is possible to prove that the *average* cannot reach six), so that if the average were the determining factor, a simple unordered list would be a better option. However, it is possible to create artificial lists where a single large region is a neighbour to every other region, but not a good merge candidate for any of them (a “comb” pattern with teeth spaced every third pixel, and the space between filled with random pixels of much different brightness to the comb is such a pattern). Tests have shown that this pattern can exhibit $O(n^2)$ behaviour if the edge lists are maintained as ordered lists, while a red-black tree implementation is much faster. This kind of pattern extending throughout the image is not typical of “real” images. Perhaps the closest would be an image containing a meandering river, but even here it is unlikely that the river would be uniform enough to be retained as a single large region. So for practical purposes the implementation with slower worst-case performance but better average-case performance will be used.

The algorithm as it was originally implemented, (Redding & Crisp & Tang & Newsam 1999), pursued the segmentation to completion (that is until only one region remains), producing as its output a merge history. The segmented image was then separately generated at the required λ value. This has a number of advantages, but if the desired λ is known ahead of time the process can easily be stopped at that point, with considerable time saving. Internally, when two regions are merged, one of them (in fact, the one containing the lowest pixel number, where pixels

are numbered starting at the image “top left” corner) is retained as the representative of the new combined region, the other being retired from the merge process. A link from the retained region back to all of its ancestors enables the simple construction of any of the required outputs directly from this data; this is maintained by appending the chain for the region being removed to the end of the chain for the region remaining.

4 A Tiling Approach to Segmenting Large Images

An obvious approach to breaking the segmentation problem down into smaller pieces is to do so by breaking the image into tiles and segmenting each tile separately. However, experience shows that in general, the segmentation boundaries of neighbouring tiles do not match well at the tile edges. It can be argued that the problem here is that segmentation is in effect a global operation with information required at all scales. This is certainly true of FLSA which searches globally for the best merge. Consequently, the decision to merge a pair of regions in one part of the image can effect a similar decision in another distant part of the image. Perhaps an obvious instance of this is when the image contains long regions such as roads or rivers but there may also be more subtle chain effects. It is evident then that when the segmentation process is interrupted by artificial tile edges a less than perfect segmentation may result and mismatches at the tile edges are highly likely.

Having highlighted the global aspects of segmentation we emphasise that we believe that global effects diminish rapidly with distance and that the mismatching at tile edges is mostly a local effect. With this in mind we propose the natural solution of using overlapping tiles so that each image pixel is well inside the edges of at least one tile and hence part of some reasonably accurate segmentation. However, this raises a new issue since there will be at least two differing segmentations in each overlap and we will need to somehow merge them. While we concede that due to the global nature of segmentation there is probably no perfect solution to this dilemma, we propose the following pragmatic approach.

The further inside the tile a segmentation region is, the more accurate it is likely to be. Thus, we assume regions which lie either partially or completely inside the non-overlapping area of a tile have been “accurately segmented” whereas regions which lie entirely within the overlap area have not. This view point leads naturally to the idea of retaining all the “accurate” regions as they are but re-segmenting all the “inaccurate” regions. The later can be done by re-uniting all the “inaccurate” regions, then re-initialising them so that each pixel is once again an individual region and then re-starting the segmentation process. Note that the overall effect of this approach is to replace the artificial tile edges with real image edges derived from the “accurately segmented” regions.

We have still not fully described the algorithm in that we have not yet specified the order in which the tiles are processed. This raises an important point: if we proceed by first segmenting two overlapping tiles and then re-processing the overlap area between them as described above, we encounter a problem. The problem is that the “accurately segmented” regions from neighbouring tiles may extend into the same part of the tile overlap area and hence overlap each other. When this happens, it is likely that the segmented regions will not agree and hence we will still have some of the same ambiguities we were trying to avoid in the first place. Note however that there has

been a gain in that in general, we will now have far fewer ambiguities than before. Moreover, the wider the overlap the fewer ambiguities there will be. Consequently, we are now justified in using heuristic solutions. We propose two natural solutions (although there are obviously more). The first relates to sequential processing of the tiles and is discussed in depth in the remainder of this section and in the next section. The second solution relates to parallel processing and is discussed briefly in the second last section.

In the situation where the need is to segment the imagery as it becomes progressively available, it is clear that the tiles should be segmented sequentially. To consider this case further, we assume (without loss of generality) that the tiles are made available within the image in rows from top to bottom and within the rows from left to right. We process the first row of tiles as follows: segment the current tile in the row; locate its “accurately segmented” regions and output them; re-initialise the remaining pixels as separate regions; wait for the neighbouring tile to the right to become available; combine the remaining pixels of the left tile with the new raw pixels of right tile; segment the resulting collection of pixels and repeat the process. Subsequent rows are processed a similar way: the remaining pixels from the tiles above and to the left of the current tile are combined with the new raw pixels of the current tile; the resulting collection of pixels is segmented; the “accurately segmented” regions are located and output; the remaining pixels are re-initialised as separate regions; the process is moved to the next tile and repeated. Full details of the algorithm are given in the next section and a graphical illustration is given in Figure (2).

As we have indicated, we do not expect this approach (or any other based on tiling) to be perfect. The most obvious defect is that certain parts of the tile edges will become segmentation edges and artificially cut some of the larger regions into two or more pieces. This occurs precisely when a large region straddles a tile overlap area. Because of the asymmetry of our sequential processing technique, these artificial segmentation edges will always lie on the right or bottom edge of the tile overlap areas. Of course, if the tiles are processed in some other way, say from the right to left and bottom to top, then artificial boundaries will lie elsewhere and a different segmentation will result. Fortunately there are two factors mitigating the occurrence of artificial boundaries. First, we can control the problem to some extent through the choice of tile size and overlap area. The larger both are, the fewer artificial boundaries there will be. In fact, if w is the width of the overlap area (in pixels) then we can guarantee that regions which do not extend further than w pixels in any direction will not have artificial edges inserted. Thus, choosing the tile size and overlap area allows us to trade speed off against accuracy.

The second mitigating factor relates to the use to which the segmentation will be put. For applications in the ADSS (and many other image analysis environments) further processing will normally distinguish between two kinds of regions: large regions which are likely to be “background” to items of interest; and small regions which may themselves be items of interest. In this context, the insertion of an artificial boundary into a large region is unlikely to have significant effect on the further processing, while the breaking up of a small region could be detrimental. Consequently, a tiling approach to segmentation is acceptable in the ADSS.

We conclude this section with a final comment on computational complexity. A further advantage of the tiling approach is that, while the segmentation itself may be $O(n \log n)$, or in the worst case

$O(n^2 \log n)$, in the tile size, it becomes linear in the number of tiles, and hence linear in the overall image size.

5 The Algorithm

In this section we present the details of our algorithm for segmenting large images. It is based on partitioning the image into overlapping tiles. It has been designed to require minimal changes to the underlying algorithm and to avoid as much as possible the introduction of artificial segmentation boundaries due to the tile edges.

Algorithm 1

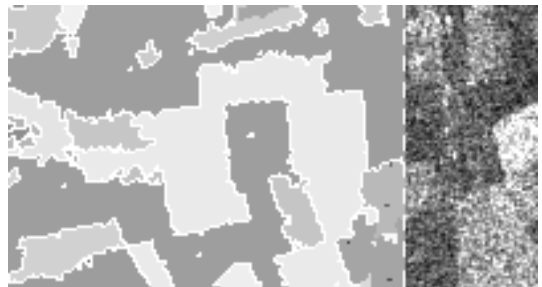
1. Partition the image into overlapping tiles and process them in order by processing each row of tiles from left to right, starting with the top row and working through to the bottom.
2. Load the first tile (the tile at the top left of the image) and segment it as usual.
3. Output the current segmentation region map as follows. Any region which lies entirely within an area which will be overlapped by a subsequent tile is assigned a region number of zero (indicating that it has not been segmented reliably). All other regions are assigned a unique region number greater than 0.
4. Load the next tile and initialise it ready for segmentation as usual by defining each pixel to be a separate region.
5. Before the segmentation process proceeds and any merge costs are calculated, the segmentation region map for the overlap area (at the top of the tile and to the left of it) is also loaded, and any pixels which have already been assigned a region number are pruned from the data.
6. The merge costs and merge candidate list for the remaining regions (pixels) are then calculated, and the segmentation proceeds.
7. Prior to output of the current segmentation region map, the previously output data for the overlap areas is read back – this ensures that the values for the pruned pixels, which will not be stored as output from the current tile, are not destroyed.
8. Repeat from step (4) until all tiles have been processed.

Note that the algorithm allows small regions near the edge of the tile the opportunity to “grow” into the next tile and hence avoid an artificial boundary at the tile edge. On the other hand, a region which extends from outside of the overlap area right to the edge of the tile does not have such an opportunity, and so will have an artificial boundary inserted. This latter defect is mitigated by the fact that if the overlap width w is sufficiently large then such regions are likewise “large”. An example of the output from the algorithm is shown in Figure (2). It is discussed in the next section.

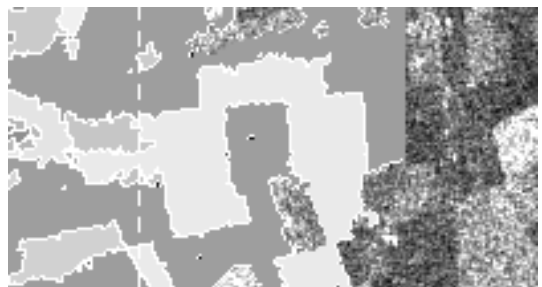
In order to identify which regions lie entirely within the overlap areas the algorithm maintains a bounding box for each region. Maintaining of the bounding boxes is clearly linear in the number of merges, as well as being a very simple computation, and so has very little impact on the overall processing time. The pruning of already merged regions is



(a) Edges of the overlapping tiles



(b) Segmentation of left tile



(c) Initialisation for segmentation of right tile



(d) Final segmentation result

Figure 2: Segmentation of the SAR image using the tiling approach. Two tiles were used with an exaggerated overlap for clarity. See the text for a full discussion of this result.

also a low cost operation, since at this stage the position of the data for each pixel is known. Thus the major overhead of the modified algorithm is the re-processing of some pixels in the overlapped regions, and the need to read back some of the already output data. Both of these are linear in the amount of overlapped image, which itself is linear in the image size n and the overlap size w .

We make the following additional comments:

- While we have mentioned only the segmentation mask, it is possible to have the algorithm output the image model and other details. In the case of the segmentation region map, we could have stored the segment numbers at step (5), but this does not apply to other forms of output.
- If the region number identifier wraps around it is reset to 1. The unlikely possibility that neighbouring regions have the same number can be avoided by appropriate choices of maximum region number and tile size.
- When a complete row of tiles has been produced, we can guarantee that the segmented image will not change above the bottom edge of the tile less the overlap amount and so further image exploitation can proceed in that region.
- We could process the tiles column-wise rather than row-wise. In practice where expediency is important, the smaller dimension would be processed first to ensure the results are available as soon as possible.

Thus we have described a segmentation algorithm which is computationally efficient, exhibits the desirable property of not introducing artificial boundaries into small regions, is linear in the image size, and allows subsequent processing to proceed up to a guaranteed row.

6 Results for the new algorithm

In this section, we first discuss the accuracy of the segmentations produced by our algorithm and then go on to memory and speed considerations.

Recall that Figure (2) illustrates the results of applying our tiling based segmentation algorithm to an image. We have chosen this image because it was used to illustrate the original FLSA algorithm. While it is a low resolution Synthetic Aperture Radar image of agricultural fields, our algorithm is general in nature and can be successfully applied to any imagery.

The final result in Figure (2) should be compared with that in Figure (1). Note the poor segmentation at the bottom right corner of the left tile in panel (b) which has been repaired in panel (d). The artifacts introduced by the modified algorithm consist of spurious straight boundaries at the right hand and bottom edges of the tiles, but only in segments which are larger than the overlap; such an artifact — the vertical division about three quarters of the way across the image — and the reason for its existence, can be clearly seen in the example shown in Figure (2). For our processing, these artifacts do not have serious consequence. The worst case would be if the segment would “really” have ended just a few pixels over; in that case the remaining pixels may form a small separate region, or be incorrectly integrated into a nearby larger region.

While the memory usage for the original algorithm is only linear in the number of pixels to be processed it is still quite high. Since the memory accessed is essentially randomly located, the algorithm thrashes badly if virtual memory is required, so the available memory

determines the largest image that can be practically processed.

The results presented here were obtained on an Athlon XP processor with 500 Mbytes of RAM and a CPU clock at 1733MHz. The largest image that can be “perfectly” segmented by this processor is a little over 2 megapixels, and would take around 30 seconds to process. The λ value selected has some impact on this, with smaller values of λ taking less time and producing a finer segmentation, but once it is large enough to produce a recognisable segmentation the sensitivity to λ is small.

The image (not shown here) used for these runtime tests was 5000×10000 pixels — too large to be segmented as a unit — and taken from a larger SAR image. A fixed tile overlap of 50 pixels was used throughout. Clearly a larger overlap will result in statistically fewer artifacts at the expense of additional processing time.

Figure (3) shows the effect of changing tile size on both memory and CPU time. The $n \log n$ nature of the underlying algorithm is clearly seen in the reduced time taken when processing with smaller tile sizes, despite the fact that more pixels are being reprocessed because of the overlap. The memory used in this test was approximately 1Mbyte — this includes the executable code — plus a little less than 2000 bytes per pixel; the important point here is that memory requirement is determined by tile size, not by image size, and so an image of any size can be processed.

Figure (4) shows the effect of image size on processing time, and clearly shows that the relationship is linear. For this processor, the constant of proportionality is about 13.7 seconds per megapixel (We note that this image represents around 1 second of SAR data at collection, so the processing is still well below real-time on this machine). This data was obtained using the same image, truncating at various points; the tile size used was 5000 pixels wide — the complete image width — by 400 pixels high. This of course avoids introducing vertical artifacts, but as the above results show, a small increase in processing speed could be gained by using narrower tiles.

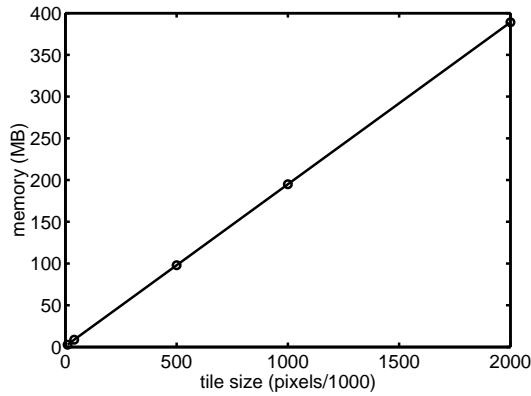
7 Edge Intolerant Processing

Although our current practical requirements for a segmentation algorithm are not sensitive to artificial edges in large regions, some future applications may be. For example, tracking algorithms which attempt to identify roads and other physical features may well be adversely affected by apparent geometric features artificially introduced.

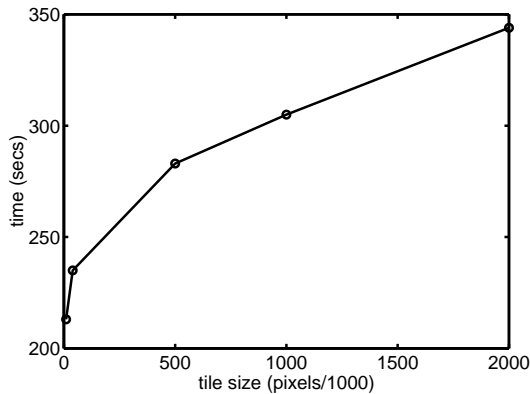
A simple and obvious enhancement of the above tiling algorithm can produce a “perfect” segmentation, although, it must be stressed, not necessarily exactly the same segmentation that would have been produced by processing the entire image as a single entity.

The trick is that only regions that are fully within the non-overlap area of the tile are output; other regions have their current data, including all of their ancestors, retained. When data for the new tile is loaded, already merged pixels are pruned completely as before, but those within retained regions are removed and their neighbours made to link to the retained regions. Segmentation then proceeds. The output image is usable up to the row which is the minimum of the y coordinates of the bounding rectangles of the retained regions.

If this sounds too simple to be true, it is! Firstly, if we give this an image which has a very large region (such as the “comb” image described above) then its data requirement grows linearly with the image size. This could be addressed by storing the merge history



(a) Memory usage



(b) CPU time

Figure 3: Computational results: memory usage and CPU time versus segmentation tile size.

for such regions in a file; they are only ever needed once when the region finally can be output, and the memory resident requirement is simply for its single region representation. Secondly, the process of retaining the regions is itself not simple, although an upper bound on the number can be set. It is the image width plus the vertical size of a tile. Thirdly, the process of outputting at the end of each tile becomes much more complex, although only in the case where a held-over region must be output. Fourthly, there is still the possibility of introducing artificial boundaries, because a held-over region has by definition an artificial boundary at one edge of a tile which will have an impact on the way new pixels will merge with it, and there is a serious likelihood that at least some of these edges will remain. Finally, there is a requirement for the segmentation to progress, and this algorithm cannot be guaranteed to do so, because a region which starts in the first row of the image and continues vertically down it cannot be output until no new pixels are added to it; thus progressive use of the image is thwarted.

It must be noted that as of the time of writing, this modified algorithm has not been implemented. Despite the drawbacks outlined above, this modification does have some distinct possibilities, and we do intend to pursue it.

In the current tiling version, there is a trade-off

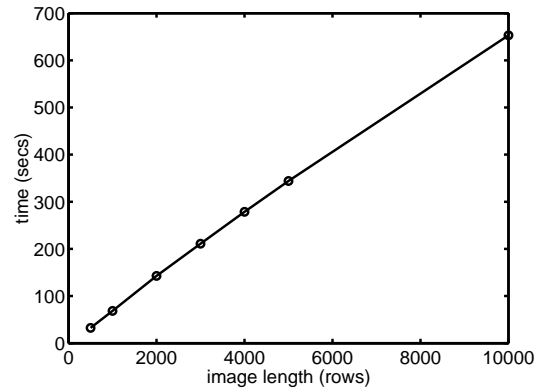


Figure 4: Computational results: CPU time versus image length.

between artifacts and processing speed; a large overlap reduces artifacts but results in more of the image being processed twice. With the modified algorithm, a smaller overlap can be used and held-over regions are forcibly output when they lag some larger amount behind the processing, thus ensuring progress and giving the effect of a much larger overlap. We also believe that the fourth issue outlined above can be addressed by artificially discouraging merges between pixels near the overlap edges using a gradient function; this would of course be removed when re-processing for the next tile.

8 Parallel Processing

Our main aim in this paper has been to describe a way of reducing the computational resources required by the FLSA segmentation algorithm to allow segmentation of very large images to be practical. We have noted that an important advantage of our suggested approach is an improvement in processing time. We now briefly discuss how further speed can be gained by parallel processing. While the segmentation of an individual tile cannot be parallelised (because each step involves the search for the single best merge), the way the collection of tiles is handled can be.

Consider a row of overlapping tiles forming a complete strip across the image. Alternate tiles in such a strip can be processed independently in an initial pass, with small segments in an overlap area at both left and right hand edges being exploded back to single pixels; the interleaving tiles can then be processed independently by pruning the already segmented pixels at both left and right edges. This represents no change to the philosophy of the algorithm, and only very minor changes to the implementation.

With this change, segmenting an image $2n$ tiles wide can be run as an initial pass of n parallel processes on the alternate tiles; as soon as both tiles adjacent to one of the passed-over tiles have been completed, a new process to segment that tile can be initiated. This effectively results in two phases of n parallel processes. In the case of an odd number, $2n + 1$ tiles results in a first phase of $n + 1$ processes and a second phase of n processes.

The same approach could be followed down the image, processing alternate strips in parallel, then processing interleaving strips as soon as both the strip above and the strip below have been completed.

We have not at present implemented this approach.

9 Summary

Some issues with the Full Lambda Schedule Segmentation algorithm have been discussed, and a significant modification to allow the processing of very large images using tiling has been described. This modification allows the generation of a segmentation of arbitrarily large images in a time linear with image size, and with predictable limits on the introduction of artificial boundaries. Further development of the algorithm to overcome the limitations of the tiling process have also been proposed.

References

- Crisp, D.J. & Tao, T.C. (2002), 'Fast region merging algorithms for image segmentation', in 'Proceedings of the Fifth Asian Conference on Computer Vision (ACCV2002)', Melbourne, Australia, January 2002, pp. 412–417.
- Koepfler, G., Lopez, C. & Morel, J.-M. (1994), 'A multi-scale algorithm for image segmentation by variational method', *SIAM Journal of Numerical Analysis* **31**, 282–299.
- Morel, J.-M & Solomini, S. (1995), *Variational Methods in Image Segmentation*, Birkhauser, Boston.
- Redding, N.J., Crisp, D.J., Tang, D. & Newsam, G.N. (1999), 'An efficient algorithm for Mumford-Shah segmentation and its application to SAR imagery', in 'Digital Image Computing: Techniques & Applications (DICTA '99)', Perth, Australia, December 1999, pp. 35–41.
- Redding, N.J. & Crisp, D.J. (2002), 'Fast algorithms for segmentation using Mumford-Shah functionals', *Submitted to IEEE Trans. Pattern Analysis and Machine Intelligence: Special Issue on Energy Minimisation Methods*.