Speeding up RIGOR

Ajmal Kunnummal Ahmad Humayun Fuxin Li James M. Rehg
Georgia Institute of Technology

***Abstract*—RIGOR is a cutting edge algorithm that generates overlapping segment proposals in images that can then be fed into image recognition algorithms to speed up object recognition in images. RIGOR was proposed and implemented in 2014 and at the time was the best performing such algorithm. The long-term goal is to design a GPU driven highly parallelized min-cut algorithm that can produce high quality object proposals for real time object recognition. For this to be feasible, the 'setup' stages of the algorithm where the graphs are generated from the images have to be extremely fast. Initially the setup stages were done in Matlab and took over a second to run. We have improved the running time of the algorithm drastically by implementing the 'setup' part in C++.**

Keywords—RIGOR, image segmentation, object proposals, C++, OpenCL

I. Introduction

Recently image recognition algorithms have imporved by a large margin. One of the main reasons for this is improvements in object proposals which is an intermediate step in most of these algorithms. These algorithms produce a pool of overlapping segments that are used as object proposals by the overall image recognition algorithm. This is orders of magitude faster than traditional algorithms that brute forces object proposals by using different sized sliding windows in the image [2]. Image segmentation algorithms also have applications in other areas like Medical imaging, object detection and traffic control systems.

Fig. 1. Typical object proposals from a segmentation algorithm. These ones are produced by RIGOR. Here it pick out the horse, the women and a car in the background with reasonable accuracy.

RIGOR is one such algorithm that was developed here in Georgia Tech. The initial implementation was developed in MATLAB with certain computationally intensive parts written in C++. We have reimplemented the entire algorithm, particularly the 'setup' stages of the algorithm in C++ to improve running time so it can become more feasible in a real time object recognition pipeline.

II. Literature Review

A. Context

Traditional segmentation algorithms like clustering, region-growing and watershed either produce only non-overlapping segments or have low performance and efficiency. In recent years several graph partitioning based methods have been developed that show much better performance and efficiency.

Several different methods can be used for partitioning. One such algorithm is constrained parametric min-cuts [3] (CPMC), which was developed in 2010. various other algorithms including Rigor (which considerably reduced the running time) improved on CPMC.

B. RIGOR

RIGOR is another algorithm for generating object proposals using parameterized min-cuts. It is based on CPMC but adds several pre-computation steps to improve efficiency. It performs on par with CPMC but runs an order of magnitude faster due to pre-computation steps [1]. It was the best performing algorithm of its kind at the time it was proposed in 2014 but has since been surpassed.

The Matlab implementation of Rigor takes about 1 second to setup up the graphs per image before it runs the min-cut. Although this is a lot better than CPMC, it is still not fast enough to be used in many image recognition pipelines like video analysis, object tracking or robotics. We think we can improve running time by another order of magnitude doing the setup stages in C++ and by running the min-cut algorithm over a GPU without losing any segmentation performance. This was proposed in the paper where RIGOR was introduced [1]. We have implemented the first part, which is setting up the graphs.

III. Implementation Details

Much of the gory details of the algorithm especially about setting up the graphs were left out from the original paper. We will try and summarize some of them here.

Rigor first converts an image to a superpixel map, which is a simple low-level non-overlapping segmentation. The superpixels represent small chunks of the image that are similar enough to be considered as one pixel. This speeds up the algorithm since it reduces the number of nodes in the graph. This map is then partitioned using parameterized minimum cuts [3], which gives us overlapping segments of superpixels. The superpixels are then converted back to images to make object proposals.

The graph consists of one node for each superpixel and two terminal nodes s, for the foreground or object, and t, for the background. The graph has two types of edges: unary edges and pairwise edges. The unaries are edges between the terminals and the superpixels and pairwise edges are the edges between neighboring superpixels.

Seed nodes have infinite weights to the s terminal so they always fall in the same cut. The weights for the rest of the edges are determined in various ways, which are described in a little more detail in the following sections.

A. Superpixels

Rigor uses a 'superpixel graph' instead of a pixels graph to improve computational speed. Finding proposals from such graphs do not reduce computational performance [1]. Producing a superpixel graph is computationally cheap if done efficiently as we will see. Several algorithms exist to produce the initial segmentation like SLIC and Structured Edge Detector that can then be used to generate the superpixel graph.

B. Initial Precomputation Steps

Several metrics are pre-computed for each superpixel that are stored with the superpixel graph like average color, external nodes and number of pixels for each node of the graph. This information is used later on to produce unary edge weights.

Fig. 2. Above: Test image called 'peppers'. The superpixel map of the test image. Each superpixel is drawn using the pre-computed mean color of that superpixel.

C. Seeds

The seeds are generated on a grid. Each seed is the set of superpixels that partially falls within a square of certain side length that is centered on each position in the grid. The proportion of rows and columns are determined by the aspect ratio of the image.

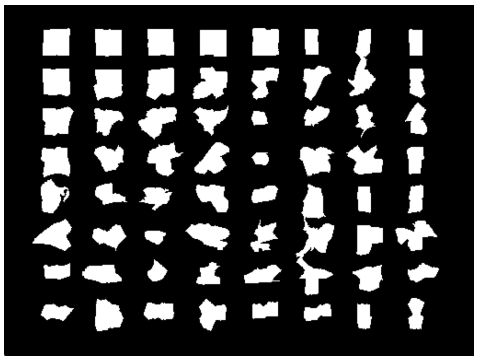

Fig. 3. 64 seeds on the 'peppers' image.

D. Pairwise Edges

The setup stages will produce $3 \times s \times l$ graphs (where s = the number of seeds, and l = the number of lambda values and there are 3 types of graphs) with uniform unaries and the same amount for color unaries but there are only two sets of pairwise edge weights: one for the uniform graphs and one for the color graphs. Therefore it makes sense to pre-compute them and store them separately.

The edge weights are derived from the edges of the image. Several different edge detectors can be used, Structured Edge Detector being the current best performing one. There is a pairwise edge between each pair of neighboring superpixels. For each neighboring pixel pair in the boundary between superpixel pairs, a weight is calculated by taking the average or max edge value. Ten percentile values of these weights are combined with the number of pixel pairs and the average weight of the pairs to build a 12-element feature vector for each superpixel pair. The feature vectors are fed into an efficient piecewise-linear regression tree that produces a single value that represents a metric of how much the boundary of the superpixel represents the boundary of an object with the background. An inverse exponential is then taken of this value (since higher values mean less likely cuts when running min-cut), which is then scaled (against the unary edges) to produce the final pairwise potentials.

E. Graph Objects and Unary Edges

Each graph has its own set of unary edges. These are computed right before each graph is handed over to the min-cut algorithm, as they are unique to each graph.

Rigor currently uses 6 different types of graphs:

Uniform Internal – This graph has uniform unary potentials for all superpixel nodes between both terminal

nodes, normalized by the size of the superpixels. So the algorithm only takes the edges of the image into account while generating proposals with these graphs.

Uniform External and External2 – These graphs try to take into account that the outer borders of the image are less likely to be part of the foreground. The 'internal' superpixels have the same values here but the border superpixels are weighted by how far they are to the seed center.

Color Internal – Here the unaries are weighted by the color difference between the superpixels and the seeds. The distance metric used is Bhattacharya distance since it also takes the variance of the values into account compared to Euclidean distance, which is just a mean. The inverse exponentials of the weights are then normalized by the size of the superpixels, which are then linearly scaled against the pairwise edges.

Color External and External2 – This similar to the uniform extern graphs: the borders of the image are weighted less.

F. Lambda Values

The set of λ values are responsible for generating multiple cuts per seed. The first two values are set at 0 and a very small number l (currently 0.001). $n - 2$ other values are chosen between l and u, the upper bound, in log-space to produce n values. The $\overline{\lambda}$ values are the same values taken in reverse order. There are two sets of $\lambda/\overline{\lambda}$ values: one for uniform graphs (0 -15) and one for color graphs (0-150).

IV. Results

The Matlab implementation of rigor takes about 2 seconds to setup the graphs after generating the superpixel and edge maps. Our current implementation does this in less than 200ms: an improvement of more than an order of magnitude.

V. Future Work

A. Parrallelizing RIGOR

The next step to successfully get proposals in real time is parallelizing the min-cut algorithm since that stage still takes 2-3 seconds to run. Most programs run general calculations on the CPU. But since Nvidia released CUDA, the GPU has been shown to be a powerful parallel coprocessor. This is mostly because modern GPUs have hundreds if not thousands of cores versus 2-8 on CPUs, allowing GPUs to perform large amounts of computations in parallel. The Nvidia GTX 780 for example has 2304 cores.

Of course any kind of optimization comes with it's own challenges. Here, it's the need to parallelize as much of algorithm as we can. As the main advantage of GPUs is the number of cores and because sending data back and forth to GPU come with overhead, parallelizing is crucial to getting significant improvements in running time.

There are numerous algorithms for min-cuts, each of them with benefits and weaknesses. Some are easier than others to parallelize. The current implementation of RIGOR uses Boykov-Kolmogorov [4] for computing min cuts. This algorithm was designed and tuned to specifically to find min-cuts in graphs produced by images and thus performs faster than all other serial min-cut algorithms for our purposes.

Unfortunately Boykov-Kolmogorov is very difficult to parallelize and so we will probably go with another method like push-relabel, which has much slower performance on a single core but can still be faster than Boykov-Kolmogorov if parallelized. This needs to be researched more thoroughly before we start on an implementation.

A parallelized min-cut algorithm along with our improved setup code should be able to generate high quality object proposals in real time.

References

[1] A. Humayun, F. Li, and J. Rehg, 'RIGOR: Reusing Inference in Graph Cuts for generating Object Regions', in IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 336–343.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, 'Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation', in IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–7.

[3] J. Carreira and C. Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *CVPR*, 2010

[4] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–1137, 2004