Homogeneous Superpixels from Random Walks

Frank Perbet and Atsuto Maki
Toshiba Research Europe, Cambridge Research Laboratory
{frank.perbet,atsuto.maki}@crl.toshiba.co.uk

Abstract

This paper presents a novel algorithm to generate homogeneous superpixels from the process of Markov random walks. We exploit Markov clustering (MCL) as the methodology, a generic graph clustering method based on stochastic flow circulation. In particular, we introduce a new graph pruning strategy called compact pruning in order to capture intrinsic local image structure, and thereby keep the superpixels homogeneous, i.e. uniform in size and compact in shape. Further, this new pruning scheme comes with three advantages: faster computation, smaller memory footprint, and straightforward parallel implementation. Through comparisons with other recent standard techniques, we show that the proposed algorithm achieves an optimal performance in terms of qualitative measure at a decent computational speed.

1 Introduction

The unsupervised over-segmentation of an image results in small patches of pixels commonly called *superpixels*. The objective of superpixels is to encode an input image in a compact manner at a low-level preprocessing stage while reflecting most of the structural information to facilitate a higher-level task such as classification. Thus, two of the important requirements for superpixels are that (a) they should be computed efficiently, and that (b) they are perceptually meaningful with local coherency.

Superpixels first appeared in [9] where the Normalized Cuts criterion [11] was applied using both contour and texture cues. An alternative method is to employ mode seeking techniques such as mean shift [1], or the recently introduced quick shift [13] which was employed for example in the context of localising object classes in images. Other possible methods include a top-down approach by superpixel lattices [7], and a bottom-up graph-based approach [2]. Generally, superpixels form a basis for many other recent vision applications. See [4, 14] for a few examples. A remaining important challenge is, however, to generate homogeneous[1] superpixels at a reasonable computational cost, which is the goal of this work.

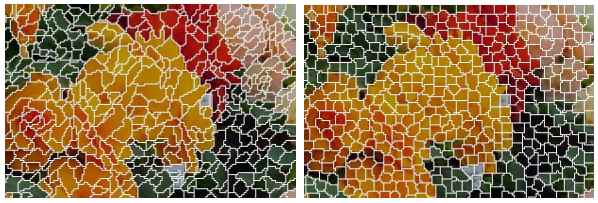
In this paper, we present a new and efficient approach to compute superpixels using Markov random walks over the graph representation of an image. The use of random walks in computer vision traces back to the early work on texture discrimination [15], and more recently the work of [3] motivated it for interactive image segmentation using seed labels. It was also shown that Normalized Cuts [11] can be viewed as a process of random walks [6]. Although we also utilise the representation of stochastic matrix, our approach differs in that we do not perform any potentially expensive spectral analysis of the adjacency matrix and mainly exploit the fact that random walks can capture intrinsic local image structure.

Figure 1. **MCL-superpixels process: overview.** Top-left: the input image initially interpreted as a graph with a similarity function (graph edges overlaid) Top-right: an intermediate state. Bottom-left: the result, i.e., a set of disjoint trees. Bottom-right: the borders between those trees showing clusters.

We base our approach on Markov Clustering (MCL) [12]. This is a general purpose graph clustering method using stochastic flow circulation, which has also been successfully applied to video segmentation [8]. See Figure 1 for an overview (we refer to our approach as 'MCL-superpixels' for convenience). However, MCL, in its original form, produces non homogeneous superpixels. Furthermore, it does not scale well to large images as it fails to compute the square of the stochastic matrix in a reasonable time, in spite of using a standard sparse matrix scheme; for a megapixel image, the size of this matrix is one trillion elements. To address these two limitations, we extend MCL with the technique of *compact pruning*, the main idea of which is to enforce the flow circulation to be local, therefore producing more homogeneous superpixels and making the flow computation tractable at the same time. This results in a new *sparse matrix scheme* which is capable of dealing with huge matrix sizes and efficiently runs on parallel computing architecture such as GPU.

Hence, the contributions of this paper are (i) a novel method to generate superpixels using MCL, (ii) a new pruning strategy for MCL called *compact pruning* which allows us to generate more homogeneous superpixels, and (iii) a new sparse matrix scheme which allows us to lower the computation time and the memory consumption, and to exploit massively parallel ar-

[1] We call superpixels 'homogeneous' when they are uniform in size and compact in shape.



(a) The original MCL (b) MCL-superpixels

Figure 2. **A comparison of clustered pixels.** (a) The superpixels generated by the original MCL process are not homogeneous in size and shape. (b) Extending MCL with our new *compact pruning* results in more homogeneous superpixels.

chitecture. We also compare the performance of our approach with other recent techniques for computing superpixels [2, 7, 9, 13], both in terms of the characteristics of output superpixels and the computational speed.

2 MCL: the Markov Clustering Algorithm

We first review the Markov Clustering (MCL) algorithm [12] briefly. The main idea of MCL is to repeatedly apply two operators on a given stochastic graph. The first operator, called *expansion*, consists of flow circulation which tends to mix area of similar appearance. The second operator, called *inflation*, makes strong edges stronger and weak edges weaker, serving the dual purpose of creating cluster boundaries and electing a representative of each cluster at the same time. After convergence (i.e. when the graph is stable under those two operators), the graph has became a disjoint set of trees (i.e. clusters).

More precisely, let us define an undirected graph $G = (V, E)$ with nodes $v \in V$ and edges $e \in E$. We denote an edge, e, spanning two nodes, v_α and v_β, as e_α^β and the value of its weight as $w(e_\alpha^\beta)$, or simply w_α^β. First, G is transformed to a Markov graph, i.e. a graph where for all nodes the weights of out-edges are positive and sum to one. Let us also consider the stochastic matrix,

$$\mathbf{M} = (w_\alpha^\beta, \alpha, \beta \in [1, N]), \tag{1}$$

which corresponds to the Markov graph (also called Markov matrix), such that each entry is the edge weight, w_α^β, and N is the total number of nodes.

The *expansion* operator is to compute the square of $\mathbf{M}$ whereas the *inflation* operator is to take the Hadamard power of a matrix (taking powers element-wise) followed by a scaling step, such that the resulting matrix is stochastic again. In sum, given a non-negative stochastic matrix, $\mathbf{M}$, of a Markov graph, $G = (V, E)$, those steps can be formulated as

$$\mathbf{M}_2 = \mathbf{M}^2 \qquad \textit{expansion} \tag{2}$$

$$\mathbf{M}_1 = \mathcal{H}_p(\mathbf{M}_2) \qquad \textit{inflation} \tag{3}$$

$$\mathbf{M}_{new} = \mathcal{N}(\mathbf{M}_1) \tag{4}$$

where $\mathcal{H}_p(\cdot)$ and $\mathcal{N}(\cdot)$ represent element-wise power operation with a power coefficient, p, and column-wise normalisation, respectively. The steps are repeated while updating $\mathbf{M}$ with $\mathbf{M}_{new}$. The process stops when it reaches an equilibrium where no difference is observed between $\mathbf{M}$ and $\mathbf{M}_{new}$. At this stage, the resulting graph, described by the resultant stochastic matrix, appears as a set of disjoint trees whose union covers the whole graph. Each tree defines a cluster which can be uniquely represented by the tree root. Thus, a given node can simply retrieve the identity of the cluster to which it belongs by tracing the tree up to its root.

The most important parameter governing the behaviour of the MCL process is the inflation parameter, p, which influences the resolution of the output. A high inflation value produces a higher number of smaller clusters. The reader should note that the number of clusters generated by MCL is emergent (i.e. not set directly). Practically, the convergence time of MCL greatly depends on the target resolution of clustering; the coarser the expected clusters are, the longer it takes. Moreover, the convergence of MCL is known to be more stable for fine resolution [12]. It is therefore well suited to the computation of superpixels for which a fine resolution is typically required.

3 Clustering Image Pixels Using MCL

We interpret an input image, $\mathbf{I}$, with size $n_x \times n_y$ pixels as a graph $G = (V, E)$. Each pixel corresponds to a node in the set $V = \{v_{f(i,j)} \mid f(i,j) \in [1, n_x] \times [1, n_y]\}$. The *flat index* function $f(i,j) = j \cdot n_x + i$ returns a one dimensional index to the node (i, j). The number of nodes, N, is the total number of pixels; $N = n_x n_y$. The set of edges, $E = \{e_\alpha^\beta\}$, connect neighbouring nodes, e.g. $v_{\alpha=f(i,j)}$ and $v_{\beta=f(m,n)}$.

In order to reflect the image structure in the graph, a common feature of graph based image analysis is to define a function that maps a difference in image intensities to edge weights. Although various weighting functions can be used, in this paper, the adjacency matrix is initialized using a simple similarity measure considering an 8-neighbourhood using a typical function given by:

$$w_\alpha^\beta = exp(-\mu \, \|I[m,n] - I[i,j]\|^2), \tag{5}$$

where $I[i,j] = (r, g, b)$ denotes the intensity of the image over the available channels. The value of μ is a free parameter (we use $\mu = 10$ in our experiments).

As explained earlier, applying the original MCL process to produce superpixels is straightforward but it suffers from two limitations:

Non homogeneous pixels: The shape of the resulting superpixels are not homogeneous in size and shape (see Figure 1(a)). Indeed, MCL does not prevent clusters from noticeably varying in size (e.g. from a couple of pixels to hundreds), nor does it prevent cluster shapes from becoming complex (not compact).

Slow computation time: The bulk of the work of the MCL algorithm is spent on computing the square of the stochastic matrix, $\mathbf{M}^2$. For large images (e.g. one megapixel) the computation time and memory footprint becomes too high. As a way to keep the computation tractable, the standard MCL implementation [12] comes with several *pruning strategies*, which aim at approximating the matrix $\mathbf{M}$ by keeping it as sparse as possible. Unfortunately, our experience shows that these pruning strategies do not perform well when dealing with graph representation of images which are typically sparse but of an extremely large dimension. For example, the result in Figure 2 (a) took 58 seconds

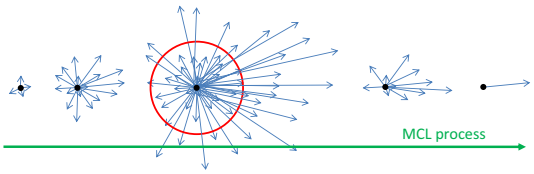


Figure 3. **Compact pruning.** During the convergence of the MCL process, the edges of a given node typically spread at the beginning (by *expansion*) and finally shrink at the end (by *inflation*). Our new compact pruning strategy consists of limiting the initial spread due to flow circulation by inhibiting edges longer than r (represented by the red circle).

to produce around 2000 superpixels on the 300×225 pixels image and runs out of memory when using a 1024×768 pixels image as input.

We deal with those two limitations by extending MCL with our new *compact pruning* method as described next.

4 Compact Pruning

We develop a new scheme to allow a better control of superpixel homogeneity and guarantee sparsity of the stochastic matrix that allows a fast computation of $\mathbf{M}^2$. The scheme, called *compact pruning*, is primarily based on the observation that for a fine resolution (to generate small clusters), the flow does not circulate globally in the whole graph but instead stays nearby a given node. Therefore, one can give a reasonable upper bound on the length of the new edges which are created during the expansion step of the MCL process (see Figure 3). Let r be a simple distance threshold in pixels insuring the following condition during each *expansion* step:

$$\|(m,n)-(i,j)\| > r \;\Rightarrow\; w_{f(i,j)}^{f(m,n)} = 0 \qquad (6)$$

Note that this is an approximation whereas the MCL process comes with a theoretical proof of convergence; a stochastic matrix, when taken to any power, remains a stochastic matrix, which means that the elements of each column sums to one. When this approximation is used, some entries corresponding to long edges will be missing and the sum of the elements of a column can then be lower than one. In practice, however, our modified MCL converges for all the images of the Berkeley database [5]. This is not surprising as the original MCL has been shown to be robust to all sorts of pruning strategy which manipulates the inflation operator to enforce matrix sparsity [12]. This distance thresholding can be seen as another pruning strategy (hence the name *compact pruning*), which manipulates the inflation operator to enforce matrix sparsity. See Figure 2 (b) for an example of MCL-superpixels.

5 Sparse Matrix Scheme

5.1 Node-centric matrix encoding

As stated in Section 3, given a huge stochastic matrix $\mathbf{M}$ with dimensions equal to squared image size, a proper strategy for matrix encoding is indispensable in order to keep the algorithm feasible both in terms of computation time and memory consumption. For

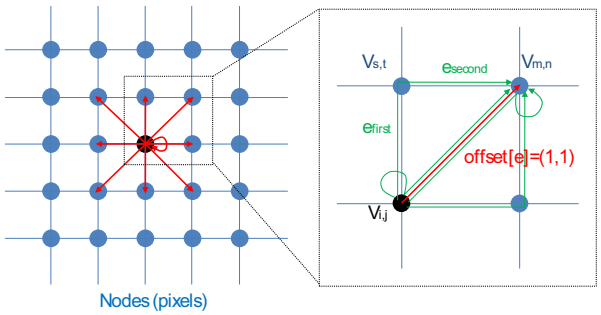


Figure 4. **Precomputation of 2-paths.** Nodes (pixels) are shown in blue dots. Left: The table `offset` is indexed (with zero-based indexing) by $\mathtt{e} \in [0, N_e[$ and contains the 2D jumps $\mathtt{offset}[\mathtt{e}] = (o_x, o_y)$ allowing to jump from a given node $v_{i,j}$ to its neighbour $v_{m,n}$ with $(m,n) = (i,j) + \mathtt{offset}[\mathtt{e}]$. In this case, $\mathtt{offset}$=[(0,0), (1,0), (1,1), (0,1), (−1,1), (−1,0), (−1,−1), (0,−1), (1,−1)] (in red). Right: For a given edge indexed by `e`, the table `detour[e]` contains the indices $(\mathtt{e}_{\mathtt{first}}, \mathtt{e}_{\mathtt{second}})$ allowing to jump from $v_{i,j}$ to $v_{m,n}$ via $v_{s,t}$ with the paths in (8) and (9). For example, the red edge is indexed by $\mathtt{e} = \mathtt{offset}[2] = (1,1)$. The corresponding 2-paths are $\mathtt{detour}[2] = [(1,3), (3,1), (0,2), (2,0)]$.

this requirement, we opt for a node-centric representation which essentially retains the image as the basic 2D structure of the graph: each node contains the edges which are departing from it.

Consequently, edge weights are stored in a volume $\mathcal{L}$ whose size is $n_x \times n_y \times N_e$ where $n_x \times n_y$ is the size of the input image and N_e is the number of edges departing from each node (i.e. pixel). Thanks to the *compact pruning* scheme, the maximum number of non null edges departing from a given node is known in advance, allowing us to allocate the volume only once at initialisation.

The edge entry, $\mathcal{L}[i,j;\mathtt{e}]$, starts from the node $v_{i,j}$ to point a node at $(i,j) + \mathtt{offset}[\mathtt{e}]$ where the table `offset` represents an offset defined by a small precomputed table containing all the 2D jumps which can be made from a given node. For example, for $r = 1$, the table is $\mathtt{offset} = [(0,0), (-1,0), (+1,0), (0,-1), (0,+1)]$. Due to the regular nature of an image graph, a unique table is shared by all the nodes instead of computing it specifically for each node.

5.2 Sparse matrix multiplication scheme

A notable benefit of our new matrix encoding is that it substantially facilitates the square computation of the stochastic matrix, $\mathbf{M}$. Each element of $\mathbf{M}_2 = \mathbf{M}^2$ in its original form is given by:

$$w_\alpha'^\beta = \sum_{\gamma=1}^{N} w_\alpha^\gamma w_\gamma^\beta. \qquad (7)$$

Let us review the meaning of equation 7 from a graph point of view. The weight, w_α^β, of an edge, e_α^β, is replaced by the sum of the products of weights on all the 2-paths (i.e. 2 consecutive edges) linking the node v_α and v_β via a third node, v_γ. Retrieving those 2-paths at computation time would be too expensive. Instead, the new encoding $\mathcal{L}$ allows us to quickly determine which edges depart from $v_{i,j}$.

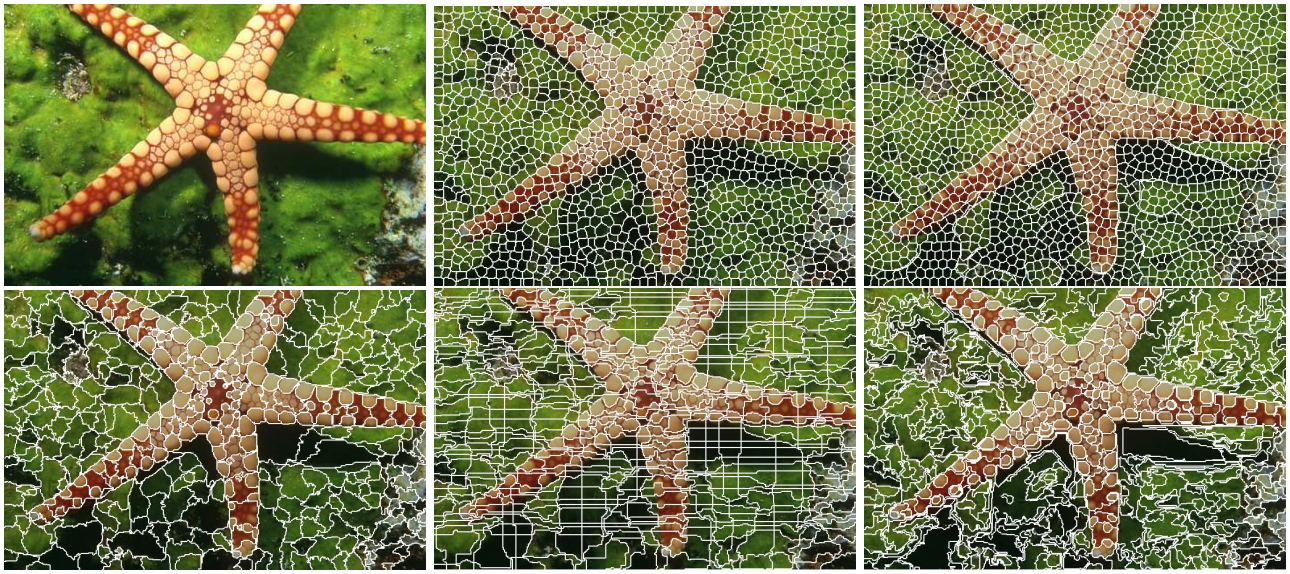


Figure 5. **Superpixels by different approaches.** From top-left to bottom-right: Input image, MCL-superpixels (proposed), NCuts, Quick shift, Lattice, and Graph-based. Note that the size and shape characteristics of superpixels are different from one another. MCL-superpixels and NCuts generate relatively regular superpixels.

We introduce a more effective alternative by pre-computing all those 2-paths. This pre-computing would be too memory expensive in an irregular graph because each node would need its own set of 2-paths. Fortunately, the regular nature of an image graph allows us to "factorise" those sets into a single look-up-table. During the computation of $\mathbf{M}^2$ with the new matrix encoding, the weights on edges indexed by $\mathtt{e} \in [0, N_e[$ (note the use of zero-based indexing) need be updated for each node $v_{i,j}$. The $\mathtt{e}$-th edge starts from $v_{i,j}$ and arrives at $v_{m,n}$ where $(m,n) = (i,j) + \mathtt{offset[e]}$. In this context, a 2-path connecting $(i,j) \rightarrow (s,t) \rightarrow (m,n)$ can be described using the indices $[\mathtt{e}_{\mathtt{first}}, \mathtt{e}_{\mathtt{second}}]$ where

$$(s,t) = (i,j) + \mathtt{offset}[\mathtt{e}_{\mathtt{first}}], \quad (8)$$

$$(m,n) = (s,t) + \mathtt{offset}[\mathtt{e}_{\mathtt{second}}]. \quad (9)$$

Therefore, we can precompute a look-up-table which encodes 2-paths by two indices, $\mathtt{e}_{\mathtt{first}}$ and $\mathtt{e}_{\mathtt{second}}$, for each $\mathtt{e} \in [0, N_e[$ (see Figure 4). We denote the table as `detour` since it represents 2-pathes via a third node. The complexity of our algorithm is $O(Nr^4)$ as opposed to $O(N^3)$ for the original MCL. The pseudo code of the algorithm below clarifies the process that each element of $\mathcal{L}$ is systematically updated in $\mathcal{L}_{new}$:

```
function computeFlow(L, L_new)
  for// (i,j) ∈ [0,n_x[×[0,n_y[  # parallel execution
    for e ∈ [0,N_e[
      (m,n) = (i,j) + offset[e]
      w_{i,j}^{m,n} = 0
      for (e_first, e_second) ∈ detour[e]
        (s,t) = (i,j) + offset[e_first]
        w_{i,j}^{s,t} = L[i,j;e_first]
        w_{s,t}^{m,n} = L[s,t;e_second]
        w_{i,j}^{m,n} += w_{i,j}^{s,t} · w_{s,t}^{m,n}
      L_new[i,j;e] = w_{i,j}^{m,n}
```

5.3 GPU implementation

The algorithm to compute $\mathbf{M}^2$ using node-centric matrix encoding maps well to a parallel architecture like GPUs (one thread per pixel). Computing the inflation is also straightforward (one thread per pixel as well). We have therefore implemented the entire MCL process on a GPU. On small images, we observed a 10 times speedup (on larger images, the original MCL runs out of memory).

We implemented our algorithm using Cuda on a GPU NVIDIA Quadro Fx 4600 using single precision floating point (345 GFLOPs). The original MCL runs on the CPU, an 8-cores 2.3 GHz computer with 3.25 GB of RAM.

6 Experimental Results

We evaluate our algorithm both in terms of the quality of generated superpixels and the computation time through a comparison with four other methods which we refer to as: NCuts (the Normalized Cuts) [9], Quick shift [13], Lattice [7] and Graph-based [2]. We compute the MCL-superpixels using the MCL inflation parameter, $p = 1.4$, and the distance threshold for compact pruning, $r = 4.5$. Our comparison focuses on the generation of a large quantity of small superpixels; therefore we tried to keep the number of clusters resulting from each methods to an approximately similar value.

Figure 5 demonstrates the quality of superpixels for an example of 'starfish' image (481×321 pixels). Also, Table 1 shows our study on the performance of those different approaches using all the 300 images of the Berkeley database [5] (see Appendix for the evaluation criterion of homogeneity: *Area*, *VoA* and the *isoperimetric quotient*, Q). Although Quick shift, Lattice and Graph-based are considerably faster, those methods are less suitable to produce small and homogeneous superpixels as observed subjectively, and also in their high values of *VoA* and low scores of Q. NCuts produces similar superpixels to MCL-superpixels and they appear better at sticking to blurred edges but this is

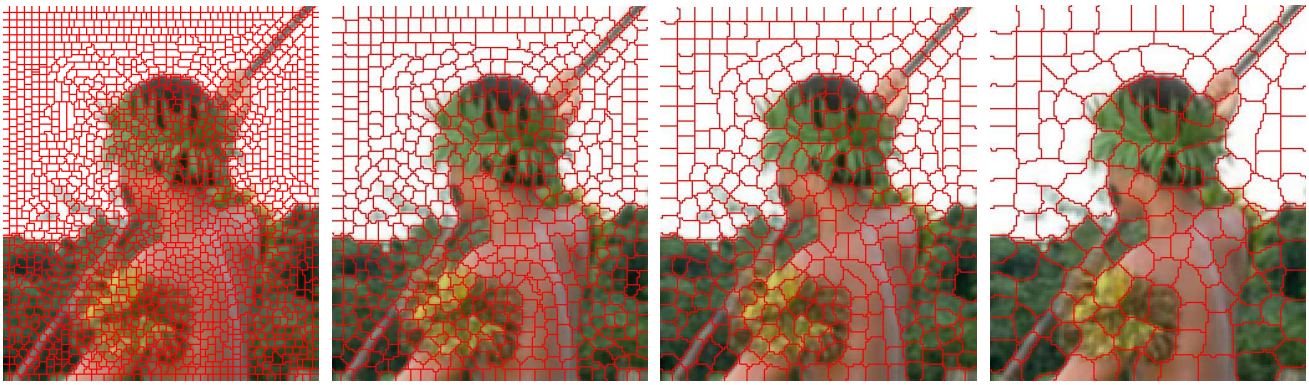


Figure 6. **MCL-Superpixels with different compact prunning.** From left to right: MCL-superpixels computed with a distance threshold for compact radius r equal to 2,3,4 and 6. The inflation parameter is set at $p = 1.4$. A greater compact radius produces bigger supperpixels.

at a cost of generating long, thin superpixels in some parts. We also tested our algorithm and NCuts on a bigger image of size 1024×768 pixel; our algorithm completed in a minute and NCuts could not complete due to an out of memory error. In summary, MCL-superpixels achieves desirable properties, high score of Q and relatively low VoA, at a speed that is ten times faster when compared to NCuts (for which no GPU implementation is known).

Figure 6 demonstrates the effect of changing the compact pruning radius: the greater it is, the bigger are the superpixels. In that sense, the distance threshold r and the inflation parameter p play an overlapping role: both control the resolution of the clusters. Ideally, those two parameters should be reformulated so that one control the size of the superpixels and the other their homogeneity; this is left for future work.

	Time [sec]	*Area*	*VoA*	*Q*
MCL-superpixels:	21.45	68	0.33	0.81
NCuts:	231.5	95	0.13	0.83
Quick shift:	2.556	174	0.82	0.63
Lattice:	0.537	100	0.93	0.57
Graph-based:	0.232	75	2.75	0.49

Table 1. **Evaluation of different approaches.** The results are averaged for all the images of Berkeley database [5]. See the appendix for the evaluation criterion.

7 Conclusion

We have presented a novel method to generate superpixels using the MCL process and a new pruning strategy, *compact pruning*, which eases the generation of superpixels in four ways. First, it makes the shape of superpixels more homogeneous by keeping the flow local. Second, the computation are faster due to sparser stochastic matrix. Third, the memory consumption is much lower (for the same reason). Fourth, it is easily amenable to a heavily parallel implementation due to a static graph topology. We have demonstrated the performance of our algorithm in comparison with other relevant techniques, and shown its capability in generating relatively homogeneous superpixels at a reasonable computational speed – a feature unique to MCL-superpixels.

A Evaluation Criterion

We denote superpixels in an image as $s_j, j = 1, ..., K$ and their areas as $A(s_j)$. We compute *Areas*, the average of $A(s_j)$, and *VoA*, the *variance* of $A(s_j)$ normalized by *Areas* as a measure of size. We also evaluate the homogeneity of superpixels by the *isoperimetric quotient*, Q_j,

$$Q_j = \frac{4\pi A(s_j)}{L^2(s_j)} \tag{10}$$

where $L(s_j)$ is the perimeter of s_j. Notice that, $0 \leq Q_j \leq 1$. The closer the shape of s_j is to a circle, the higher Q_j is. We compute $Q = (1/K) \sum_{j=1}^{K} Q_j$ for each image.

References

[1] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE-PAMI*, 17(8):790–799, 1995.

[2] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2), 2004.

[3] L. Grady. Random walks for image segmentation. *IEEE-PAMI*, 28(11):1768–1783, 2006.

[4] D. Hoiem, A. Efros, and M. Hebert. Automatic photo pop-up. *ACM Trans. Graph.*, 24(3):577–584, 2005.

[5] D.R. Martin, C. Fowlkes, D. Tal and J. Malik. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. *Tech. report*, 2001.

[6] M. Meila and J. Shi. Learning segmentation by random walks. In *NIPS*, pages 873–879, 2000.

[7] A. P. Moore, S. Prince, J. Warrell, U. Mohammed, and G. Jones. Superpixel lattices. In *CVPR*, 2008.

[8] F. Perbet, B. Stenger and A. Maki. Random Forest Clustering and Application to Video Segmentation. *BMVC*, 2009.

[9] X. Ren and J. Malik. Learning a classification model for segmentation. In *ICCV*, pages 10–17, 2003.

[10] Y. Sheikh, E. A. Khan, and T. Kanade. Mode-seeking by medoidshifts. In *ICCV*, pages 1–8, 2007.

[11] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE-PAMI*, 22:888–905, 2000.

[12] S. van Dongen. *Graph Clustering by Flow Simulation.* PhD thesis, University of Utrecht, 2000.

[13] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *ECCV*, pages 705–718, 2008.

[14] A. Vazquez-Reinai. S. Avidan, H. Pfister and E. Miller. Multiple Hypothesis Video Segmentation from Superpixel Flows. In *ECCV*, 2010.

[15] H. Wechsler and M. Kidode. A random walk procedure for texture discrimination. *IEEE-PAMI*, 1(3):272–280, 1979.