PAPER *Special Section on Machine Vision and its Applications*

Homogeneous Superpixels from Markov Random Walks

Frank PERBET[†], **Björn STENGER**[†], ***and*** **Atsuto MAKI**[†a)],

SUMMARY This paper presents a novel algorithm to generate homogeneous superpixels from 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 graph pruning strategy called *compact pruning* in order to capture intrinsic local image structure. The resulting superpixels are homogeneous, i.e. uniform in size and compact in shape. The original MCL algorithm does not scale well to a graph of an image due to the square computation of the Markov matrix which is necessary for circulating the flow. The proposed pruning scheme has the advantages of faster computation, smaller memory footprint, and straightforward parallel implementation. Through comparisons with other recent techniques, we show that the proposed algorithm achieves state-of-the-art performance.

key words: *superpixels, Markov clustering, compact pruning, sparse matrix computation*

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 (a) that they should be computed efficiently, and (b) that they are perceptually meaningful with local coherency.

One of the first methods to compute superpixels appeared in [18] where the Normalized Cuts criterion [20] was used based both on contour and texture cues. The resulting oversegmentation successfully generates a homogeneous representation of superpixels, which was later applied to guide model search [14]. However, for a preprocessing stage it introduces significant computational cost. An alternative method is to employ mode seeking techniques such as mean shift [1], [2], [4], medoid shift [19], or the recently introduced quick shift [22] which was employed for example in the context of localising object classes in images [5]. Other methods include top-down approaches such as superpixel lattices [13] or TurboPixels [8], and bottom-up approaches such as the graph-based approach in [3] which defines a predicate for measuring the evidence for a boundary between two regions while representing an image as a graph. A number of recent vision applications compute superpixels as a preprocessing stage in order to reduce the computational burden during later stages, see [7], [23] for examples.

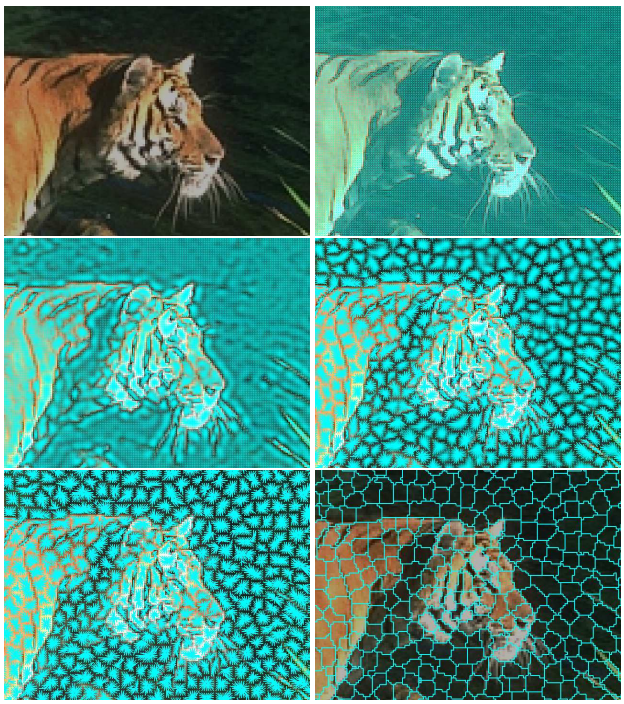


Fig. 1 **MCL-superpixels process: overview.** Top-left: the input image. Top-right: the input image and an overlaid graph with a similarity function (graph edges overlaid). Middle: intermediate states. Bottom-left: the result, i.e. a set of disjoint trees. Bottom-right: the borders between those trees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

An important remaining challenge is, however, to efficiently generate homogeneous, i.e. uniform in size and compact, superpixels, which is the goal of this work. Homogeneous superpixels are preferred for some vision algorithm while superpixel representations should be accommodated to different tasks; see [9], [15] for a few examples. In object detection and tracking [15], a probability is assigned to each superpixel to be a part of the full object. Homogeneous superpixels allow us to consider the parts as similar building blocks and thus to make the assembling step more straightforward and accurate. In noise reduction [9], the noise is estimated within each superpixel by smoothing the brightness and extracting the residual, and then reduced by a bilateral

Manuscript received October 5, 2011.

[†]The authors are with Cambridge Research Laboratory, Toshiba Research Europe.

a) E-mail: atsuto.maki@crl.toshiba.co.uk (Corresponding author)

This work first appeared in a shortened form in Perbet and Maki [17].

DOI: 10.1587/trans.E0.??.1

filtering whose smoothing power is adapted to the estimates. Homogeneity of superpixels is essential for consistently estimating the noise over the entire image. The nice property of producing homogeneous superpixels is also discussed in a few recent articles [8], [10].

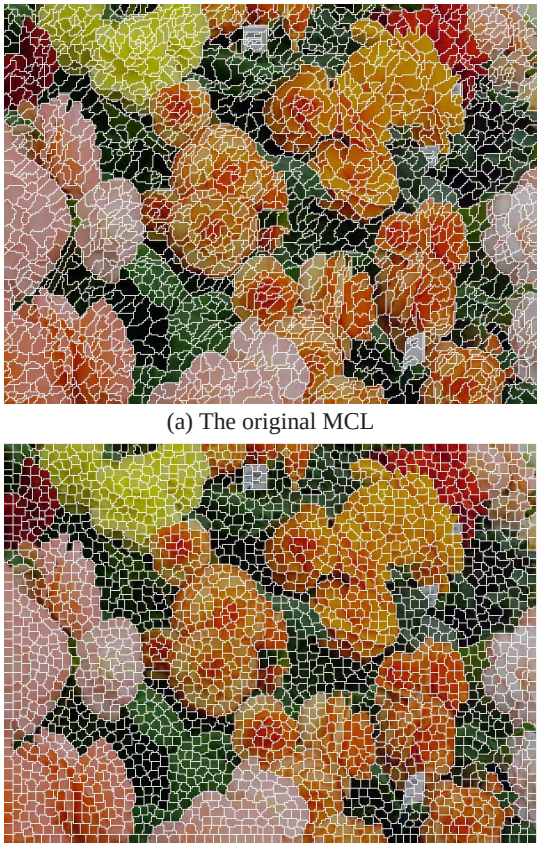
In this paper, we present a new and efficient approach to compute superpixels using Markov random walks on the graph representation of an image. The use of random walks in computer vision traces back to the early work on texture discrimination [24], and more recently the work of [6] motivated it for interactive image segmentation using seed labels. It was also shown that Normalized Cuts [20] can be viewed as a process of random walks [12]. Although we also utilise the stochastic matrix representation, our approach differs in that we do not perform any spectral analysis of the adjacency matrix and mainly exploit the fact that random walks can capture intrinsic local image structure.

We base our approach on Markov Clustering (MCL) [21], a general purpose graph clustering algorithm using stochastic flow circulation. The motivations for using MCL as a clustering algorithm are primarily in its suitability and ability to extract local image structure that are expected by its nature [21] as well as in the previous success in its application to video segmentation [16]. 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. See Figure 2 (a) for an example. 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 mega-pixel image, the size of this matrix contains 10^{12} 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 computation scheme* which is capable of dealing with large matrix sizes and efficiently runs on parallel computing architectures such as GPUs.

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* for generating more homogeneous superpixels, and (iii) a new *sparse matrix computation scheme* which allows us to lower the computation time and the memory consumption, and to exploit parallel architectures. We also compare the performance of our approach with other recent techniques for computing superpixels [3], [13], [18], [22], both in terms of the characteristics of output superpixels and the computational speed.

2. MCL: the Markov Clustering Algorithm

We first briefly review the Markov Clustering (MCL) algorithm [21]. Given a stochastic graph, the main idea of MCL is to repeatedly apply two operators on it. The first operator, called *expansion*, consists of flow circulation which tends to



(a) The original MCL

(b) MCL-superpixels

Fig. 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.

smooth areas of similar appearance. Intuitively speaking, 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 resulting graph is a disjoint set of trees, i.e. clusters (see Fig 1 bottom-left).

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 (or Markov matrix),

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

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

The *expansion* operator corresponds to the squaring of $\mathbf{M}$ whereas the *inflation* operator corresponds to taking the Hadamard power of a matrix (applying power function

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, $G = (V, E)$, the 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 the difference observed between $\mathbf{M}$ and $\mathbf{M}_{new}$ is below a small threshold. 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 large inflation value produces a large number of smaller clusters and vice versa. It should be noted that the number of clusters generated by MCL is emergent (i.e. not set directly). In practice the convergence time of MCL greatly depends on the target resolution of clustering; the coarser the expected clusters are, the longer the computation. Moreover, the convergence of MCL is known to be more stable for fine resolution [21]. 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 initialised 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 2 (a)). Indeed, MCL does not prevent clusters from noticeably varying in size (e.g. from several pixels to hundreds), nor does it prevent cluster shapes from becoming complex (not compact).

Slow computation time The bulk of the process 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 very high. As a way to keep the computation tractable, the standard MCL implementation [21] 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 representations of images which are typically sparse but of an extremely large dimension. A possible reason is that the simple process of removing the smallest entries of the matrix does not decrease the cost for computing $\mathbf{M}^2$ in a systematic manner and is therefore not sufficient for certain clustering tasks although it helps solving a problem with a relatively small scale. For example, the result in Figure 2 (a) required 58 seconds to produce around 2000 superpixels on the 300×225 pixels image and ran out of memory when using a 1024×768 pixels image as input.

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

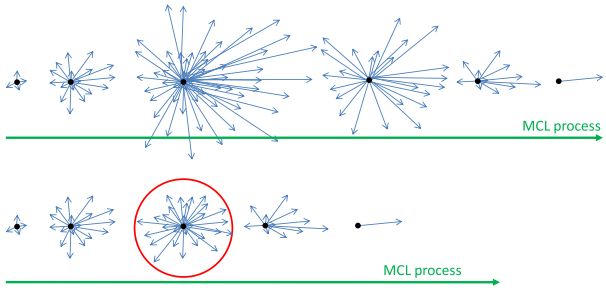


Fig. 3 **Evolution of flow around each node.** The black dot and the arrows schematically represent the node and the edges (pointing their destinations) of the graph, respectively. The evolution of the MCL process is shown from left to right. Top: During the MCL process, the edges originating from a given node typically spread at the beginning (by *expansion*) and finally converge at the end. Bottom: Our new compact pruning strategy bounds the length of the new edges (generated by *expansion*) by the radius (represented with the red circle) and thereby limits the initial spread due to flow circulation, achieving faster convergence.

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

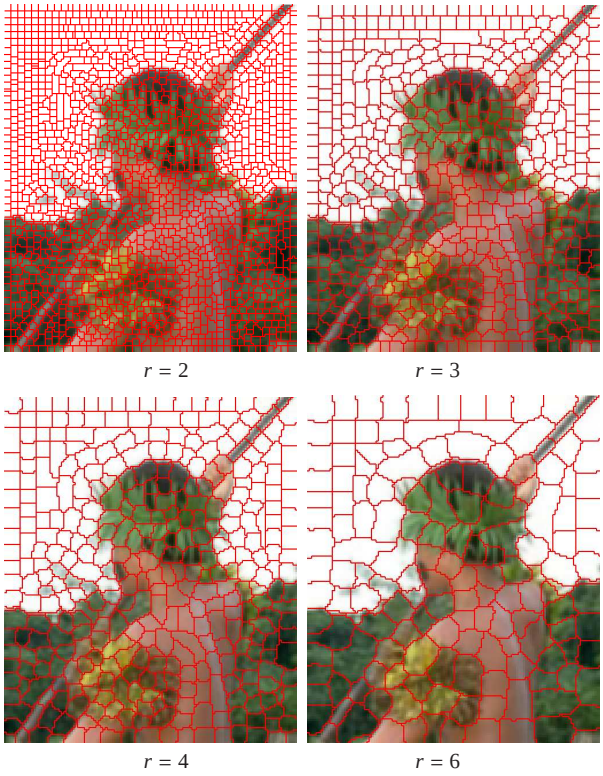


Fig. 4 **MCL-Superpixels with different compact pruning.** MCL-superpixels computed with different distance threshold, i.e. pruning radius r. The inflation parameter is set to $p = 1.4$. A greater radius r produces larger superpixels.

flow does not circulate globally in the whole graph but instead stays nearby a given node. Therefore, one can provide a reasonable upper bound on the length of the new edges which are created during the expansion step of the MCL process. Let r be a simple threshold on the straight-line distance between the centers of pixels ensuring the following condition during each *expansion* step:

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

Figure 3 shows the concept of using compact pruning where the initial spread due to flow circulation is limited by inhibiting edges longer than a threshold, r. See Figure 2 (b) for an example of MCL-superpixels.

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 sum 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 [11]. This is not surprising as the original MCL has been shown to be robust to all types of pruning strategies which manipulate the inflation operator to enforce matrix sparsity [21].

The distance thresholding can be seen as another pruning strategy (hence the name *compact pruning*), which manipulates the inflation operator to enforce matrix sparsity. Figure 4 demonstrates the effect of changing the compact pruning radius: the greater it is, the larger 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. We will evaluate the behaviour of superpixels for different values of r and p in Section 6.2.

5. Sparse Matrix Computation Scheme

5.1 Node-centric matrix encoding

As stated in Section 3, given a large 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 this requirement, we opt for a node-centric representation which 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. See Figure 5 for a schematic.

The edge entry, $\mathcal{L}[i, j; \mathtt{e}]$, starts from the node $v_{i,j}$ to point a node at (i, j) + `offset[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 `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 this 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:

$$w'^{\beta}_{\alpha} = \sum_{\gamma=1}^{N} w^{\gamma}_{\alpha} w^{\beta}_{\gamma}. \tag{7}$$

Let us review the meaning of (7) from a graph point of view. The weight, w^{β}_{α}, of an edge, e^{β}_{α}, 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 very expensive. Instead, the new encoding $\mathcal{L}$ allows us to quickly determine which edges depart from $v_{i,j}$.

We introduce a more effective alternative by precomputing 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

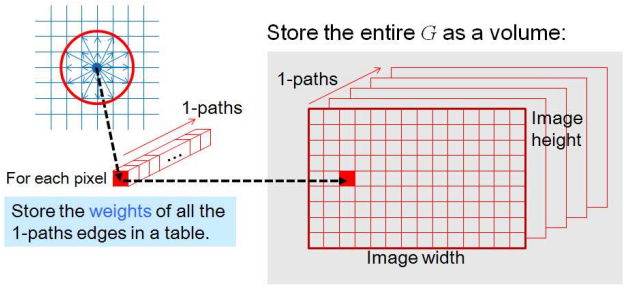


Fig. 5 **Node-centric matrix encoding.** Left: The weights of all the 1-path edges starting from a pixel are stored in a table (the distance threshold is set to $r = 2.5$ in this figure). Right: Edge weights in the entire graph $\mathcal{G}$ are stored in a volume.

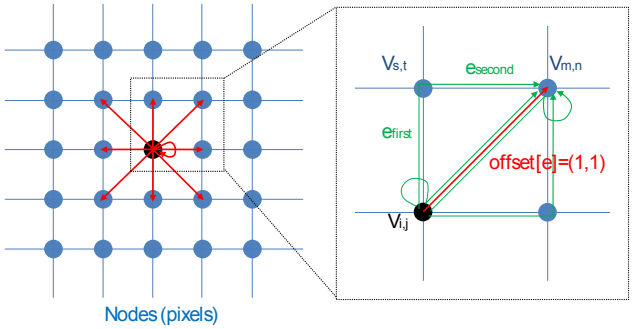


Fig. 6 **Precomputation of 2-paths.** Nodes (pixels) are shown in blue dots. Left: The table `offset` is indexed (with zero-based indexing) by $\mathtt{e} \in [0, N_e - 1]$ and contains the 2D jumps $\mathtt{offset[e]} = (o_x, o_y)$ allowing to jump from a given node $v_{i,j}$ to its neighbour $v_{m,n}$ with $(m, n) = (i, j) + \mathtt{offset[e]}$. In this case, `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 $\mathtt{detour}[2] = [(1, 3), (3, 1), (0, 2), (2, 0)]$.

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 $\mathtt{e} \in [0, N_e-1]$ (note the use of zero-based indexing) need be updated for each node $v_{i,j}$. The `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 $\mathtt{e} \in [0, N_e-1]$ (see Figure 6). We denote the table as `detour` since it represents 2-paths via a third node.

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($\mathcal{L}$, $\mathcal{L}_{new}$)
. *for*// $(i, j) \in [0, n_x-1]\times[0, n_y-1]$ # parallel execution
. . *for* $\mathtt{e} \in [0, N_e - 1]$
. . . $(m, n) = (i, j) + \mathtt{offset}[\mathtt{e}]$
. . . $w_{i,j}^{m,n} = 0$
. . . *for* $(\mathtt{e}_{\mathtt{first}}, \mathtt{e}_{\mathtt{second}}) \in \mathtt{detour}[\mathtt{e}]$
. . . . $(s, t) = (i, j) + \mathtt{offset}[\mathtt{e}_{\mathtt{first}}]$
. . . . $w_{i,j}^{s,t} = \mathcal{L}[i, j; \mathtt{e}_{\mathtt{first}}]$
. . . . $w_{s,t}^{m,n} = \mathcal{L}[s, t; \mathtt{e}_{\mathtt{second}}]$
. . . . $w_{i,j}^{m,n} \mathrel{+}= w_{i,j}^{s,t} \cdot w_{s,t}^{m,n}$
. . . $\mathcal{L}_{new}[i, j; \mathtt{e}] = w_{i,j}^{m,n}$

5.3 Computational complexity analysis

The computational complexity of our algorithm is Nr^4 where N is the number of pixels and r the pruning radius. The term r^4 comes from the fact that for a given nodes, the set of detours are all the non repetitive choices of 2 neighbours within the neighbourhood, that is, $\binom{k^2}{2}$, which is asymptotically equivalent to r^4. Another important factor is the inflation parameter which influences the number of iterations required to converge. The influence of inflation is not shown in the complexity and hidden in the constant: in practice, it can slow down the process by a factor of three. This complexity analysis shows one weakness of our algorithm: it will not perform well for a large radius r. As a consequence, it means that generating superpixels with large area will perform poorly. In practice we found that MCL performs well when generating superpixels with an area of less than 150 pixels.

Note that the complexity of the naive original MCL is N^3. Nevertheless, using some optimisation techniques such as keeping the graph sparse, this complexity becomes Nk^2 where k is the maximum number of edges per node; in our case, $k = r^2$, which leads to the same expression of Nr^4. This fact leads to an interesting discussion about the reason why our method offers such a significant speed up over the original MCL. The complexity of the original MCL algorithm is governed by the cost of a matrix-matrix multiplication. This operation can be implemented in parallel if using a full storage of the adjacency matrix but as stated before, doing so would be infeasible for a large graph. The original MCL algorithm resorts to traditional sparse matrix representations: while gaining a big performance improvement on a single thread, those schemes typically do not map well to highly parallel architectures. By adopting our new sparse matrix representation which naturally maps well to highly parallel architectures, we keep the best of both worlds and achieve similar results up to one hundred times faster.

5.4 GPU implementation

The algorithm to compute $\mathbf{M}^2$ using node-centric matrix encoding maps well to parallel architectures like GPUs by running a single thread per pixel. Computing the inflation is also straightforward, one thread per pixel. We have therefore implemented the entire MCL process on a GPU. On

small images, we observed a 10-fold 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. Performance Evaluation

We evaluate the MCL-superpixels through qualitative comparison with four other existing methods which we refer to as: NCuts (the Normalized Cuts) [18], Quick shift [22], Lattice [13] and Graph-based [3]. We employ the Berkeley database [11] and use all 300 images in the experiments. See Figure 8 for the quality of superpixels for a few example images; 'Tiger' image (part of the original, 133×100 pixels) and 'Starfish' image (481×321 pixels).

Our comparison focuses on the generation of a large quantity of small superpixels; therefore we keep the number of clusters resulting from each method similar. In this evaluation, we compute the MCL-superpixels using the MCL inflation parameter, $p = 1.4$, and the distance threshold for compact pruning, $r = 4.5$. For the evaluation we average the measures across all images in the database.

6.1 Evaluation criteria

As the evaluation criteria of homogeneity, we study the variance of areas and compactness. Using ground truth annotations provided with the database, we also examine the undersegmentation error and boundary recall as metrics of superpixels. We commonly denote superpixels as s_j, $j = 1, ..., K$ and their areas in number of pixels as $A(s_j)$, respectively.

Variance of Area One measure of homogeneity of superpixels is in terms of the area since the size of superpixels is an important factor. 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.

Compactness We evaluate the compactness of superpixels in terms of the *isoperimetric quotient*, Q_j, based on their shapes,

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

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.

Undersegmentation Error We compute a quantified *undersegmentation error* which measures the total amount of leaking caused by superpixels against overlapping ground truth segment. That is,

$$E = \frac{[\sum_{s_j | s_j \cap t_k \neq \emptyset} A(s_j)] - A(t_k)}{A(t_k)} \tag{11}$$

where $t_1, ..., t_K$ are ground truth segments. We compute the average of the measures in (11) across all $t_k, k = 1, ..., K$.

Boundary Recall We also employ standard boundary recall which is defined by the fraction of the ground truth edges that falls within a small distance threshold from at least one superpixel boundary. We set the threshold strictly to 1 pixel in our experiments and call it $recall_1$.

Note that both, undersegmentation error and boundary recall, are somewhat subjective due to the manually annotated boundaries that serve as ground truth.

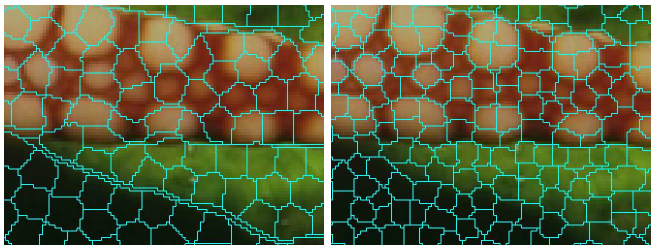
	Time[s]	*VoA*	*Q*	*E*	$recall_1$
MCL-superpixels:	21.45	0.33	0.81	0.46	0.79
NCuts [18]:	231.5	0.13	0.83	0.46	0.74
Quick shift [22]:	2.556	0.82	0.63	0.52	0.85
Lattice [13]:	0.537	0.93	0.57	0.50	0.81
Graph-based [5]:	0.232	2.75	0.49	0.47	0.93

Table 1 **Evaluation of different approaches.** The results are averaged for all the images of Berkeley database [11]. The average area of superpixels is 100 pixels. See Section 6.1 for the evaluation criteria. Note that MCL-superpixels achieve similar performance to NCuts but with a higher speed.

6.2 Evaluations

Table 1 shows our study on the performance of different approaches. Figure 8 demonstrates the quality of generated superpixels. Although Quick shift, Lattice and Graph-based are considerably faster, these methods are less suitable to produce small and homogeneous superpixels. They also result in high values of variance of area *VoA* and low compactness *Q*.

NCuts produces apparently similar superpixels to MCL-superpixels. One difference is that NCuts superpixels appear to better follow linear colour boundaries at the cost of generating long and thin superpixels in some parts. Figure 7 shows a zoomed-in view of the results for the 'starfish' image by NCuts and MCL-superpixels; it can be observed that MCL-superpixels are generated more faithfully to local edges although NCuts achieves a slightly better overall score of compactness *Q*. We also tested our algorithm and NCuts on a larger image of size 1024×768 pixel; our algorithm completed in a minute and NCuts could not complete



NCuts MCL-superpixels

Fig. 7 **Zoomed-in views.** Superpixels computed in a part of 'Starfish' image is shown. Left: NCuts. Right: MCL-superpixels (proposed). Regular superpixels are generated by both methods, but boundaries of MCL-superpixels are faithfully aligned to local edges.

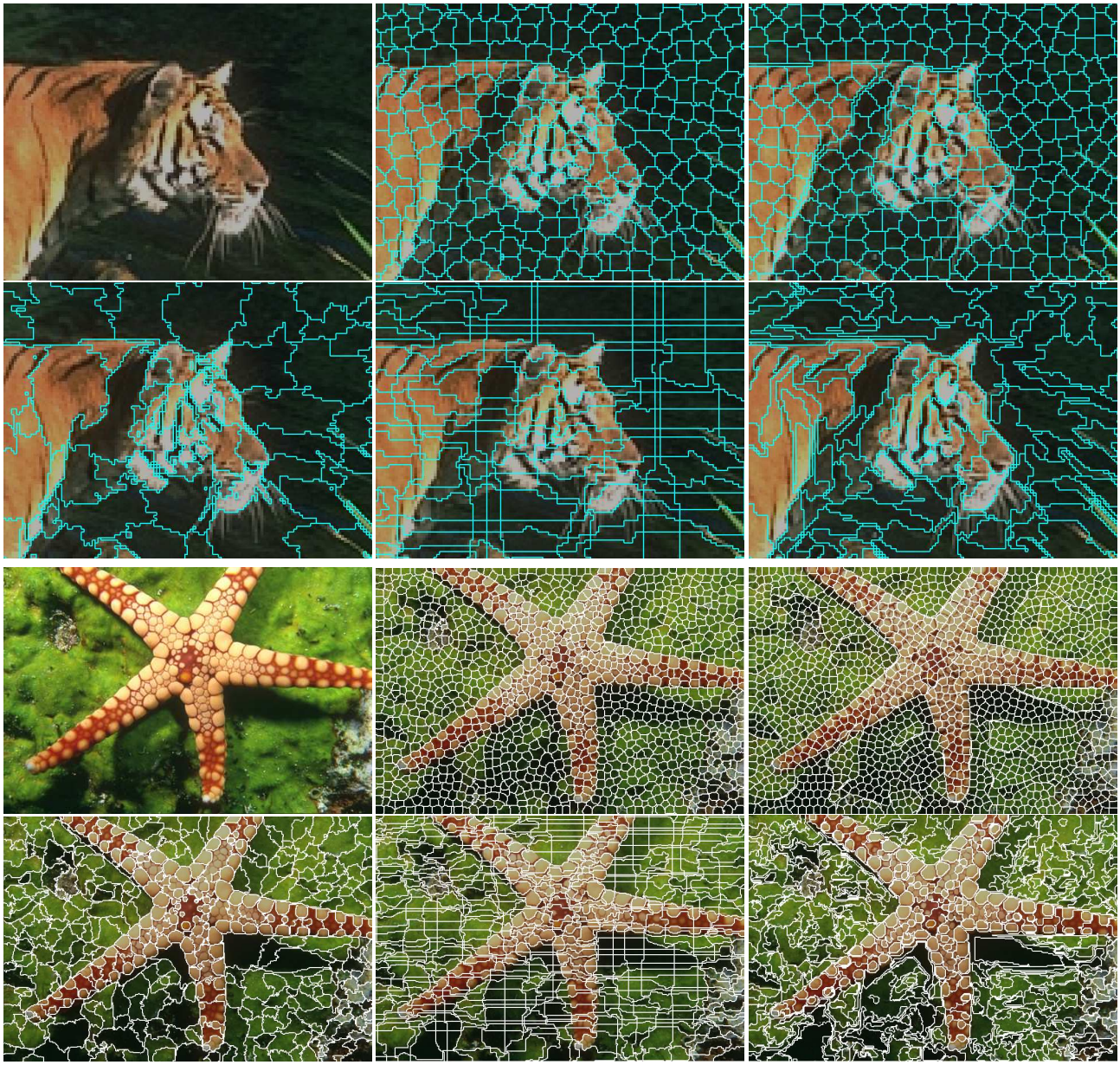


Fig. 8 Superpixels by different approaches. Two examples are shown ('Tiger' and 'Starfish'). From top-left to bottom-right: Input image, MCL-superpixels (proposed), NCuts, Quick shift, Lattice, and Graph-based. Note that the size and shape characteristics of superpixels are different from one another. MCL-superpixels and NCuts generate relatively regular superpixels.

due to an out of memory error caused in dealing with large matrices for the spectral analysis.

One of the evaluation criteria based on the ground truth is the undersegmentation error, E. MCL-superpixels and NCuts both achieve the lowest value of E among the five methods. Intuitively, the compact nature of superpixels helps to avoid strong boundary bleeding and can decrease the error, which explains why the other three methods result in slightly larger errors.

Graph-based superpixels and Quickshift on the other hand score higher boundary recall $recall_1$ than Lattice, MCL-superpixels or NCuts, which all take on certain regularities. This reflects the fact that the ground truth segments are not necessarily compact and the two methods perform well capturing non-compact regions as superpixels while trading off the score of compactness.

Although we have used fixed values for distance threshold (the pruning radius), r, and the inflation parameter, p, for fair comparisons, as discussed in Section 4, both of the two parameters play the role of controlling the resolution of superpixels. Figure 9 shows the behaviour of superpixels obtained with various values of r and p. Again,

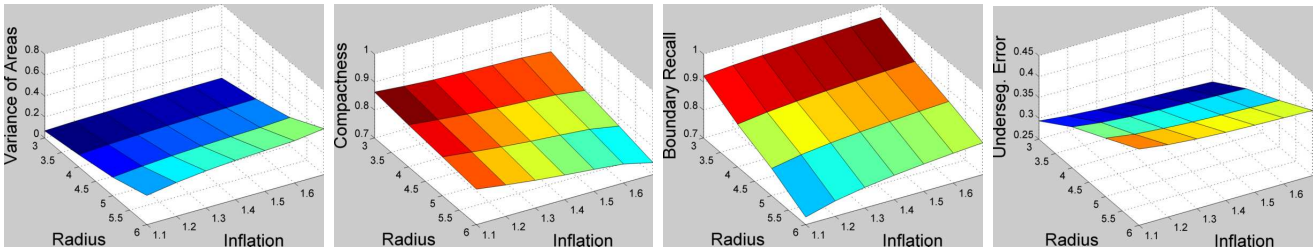


Fig. 9 **Characteristics of MCL-superpixels with different parameter settings.** The pruning radius, r, and the inflation parameter, p, are being changed. From left to right, plotted are variance of areas, compactness, undersegmentation error and boundary recalls. The results are averaged for all the images of Berkeley database [11].

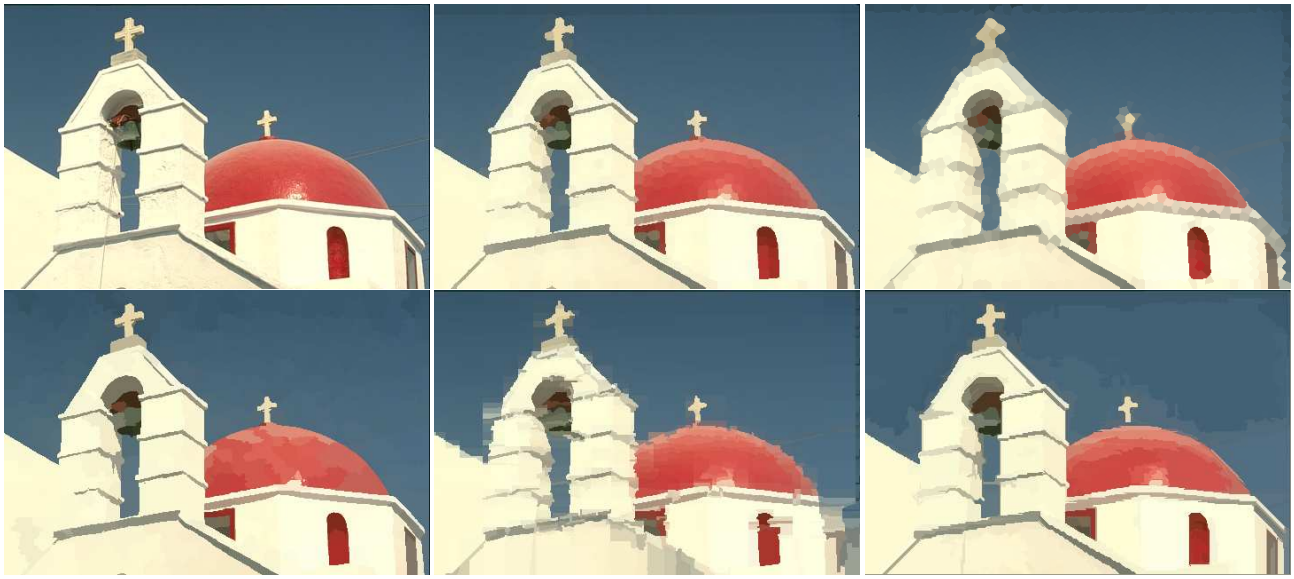


Fig. 10 **Mosaic images generated by different superpixels.** From top-left to bottom-right: Input image, MCL-superpixels (proposed), NCuts, Quick shift, Lattice, and Graph-based. Notice the difference in quality for example at the crosses and the region of red dome.

the shown results are the averages for all the images of the Berkeley database [11]. While it is possible to use p as well as r for changing the size of clusters, the influence in terms of the subjective evaluations are observed to be similar. The homogeneity appears to be generally higher for the smaller value of p. Ideally, those two parameters should be reformulated so that one controls the size of the superpixels and the other their homogeneity; this is left for future work.

Superpixels are also useful for an approximate image representation by replacing values of pixels, $I[i, j]$, for example with the mean intensity within a superpixel. Figure 10 shows an example of such a representation according to superpixels generated by different methods. In comparison to the original image (top-left), some artifacts due to this approximation are observed in all the five cases. However, MCL-superpixels qualitatively show the best reconstruction of the original. This is clearly visible in the areas of the red dome and the edges of the building. Note that the cross is significantly better reconstructed using MCL-superpixels than NCuts.

In sum, MCL-superpixels achieve desirable properties, a high score of compactness, Q, and relatively low variance of area, VoA, while keeping a low undersegmentation error, E, at a speed that is ten times faster when compared to NCuts (for which no parallel implementation is available).

7. Conclusion

We have presented a novel method to generate superpixels using the MCL process, which improves the generation of superpixels in four ways. First, it makes the shape of superpixels more homogeneous by keeping the flow local. Second, the computation is faster due to a sparser stochastic matrix representation. Third, the memory consumption is lower (for the same reason). Fourth, it is easily amenable to a parallel implementation due to a static graph topology. The key advance is that we exploited the nature of random walks that can capture intrinsic local image structure while introducing the strategies of *compact pruning* and *sparse matrix multiplication* scheme.

An important aspect for further study is the stability of superpixels across video frames between which small image variations are exhibited. Also, interesting future work will include a comparison of our method to Entropy rate superpixels [10] which was introduced after the original publication of this work [17]. In summary, we have demonstrated the performance of our algorithm in comparison with other relevant techniques, and shown that its capability in generating similarly homogeneous superpixels to those by Normalized Cuts but by a faster computation – a feature unique to MCL-superpixels.

Acknowledgment The authors wish to thank Dr. Oliver Woodford and Professor Roberto Cipolla for valuable discussions.

References

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

[2] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE-PAMI*, 24(5):603–619, 2002.

[3] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004.

[4] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975.

[5] B. Fulkerson, A. Vedaldi, and S. Soatto. Class segmentation and object localization with superpixel neighborhoods. In *ICCV*, pages 670–677, 2009.

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

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

[8] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE-PAMI*, 31(12):2290–2297, 2009.

[9] C. Liu, W. T. Freeman, R. Szeliski and S. B. Kang. Noise Estimation from a Single Image. In *CVPR*, pages 901–908, 2006.

[10] M.-Y. Liu, O. Tuzel, S. Ramalingam and R. Chellappa. Entropy Rate Superpixel Segmentation. In *CVPR*, pages 2097–2104, 2011.

[11] 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.

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

[13] A. P. Moore, S. Prince, J. Warrell, U. Mohammed, and G. Jones. Superpixel lattices. In *CVPR*, pages 1–8, 2008.

[14] G. Mori. Guiding model search using segmentation. In *ICCV*, pages 1417–1423, 2005.

[15] G. Mori, X. Ren, A. Efros and J. Malik. Recovering Human Body Configurations: Combining Segmentation and Recognition. In *CVPR*, pages 326-333, 2004.

[16] F. Perbet, B. Stenger and A. Maki. Random Forest Clustering and Application to Video Segmentation. In *BMVC*, pages 100.1-100.10, 2009.

[17] F. Perbet and A. Maki. Homogeneous Superpixels from Random Walks. In *IAPR MVA*, pages 26–30, 2011.

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

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

[20] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE-PAMI*, 22(8):888–905, 2000.

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

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

[23] A. Vazquez-Reinai. S. Avidan, H. Pfister and E. Miller. Multiple Hypothesis Video Segmentation from Superpixel Flows. In *ECCV*, pages 268–281, 2010.

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