SEEDS: Superpixels Extracted via Energy-Driven Sampling

Michael Van den Bergh[1], Xavier Boix[1], Gemma Roig[1], Benjamin de Capitani[1], and Luc Van Gool[1,2]

[1] Computer Vision Lab, ETH Zurich, Switzerland
[2] KU Leuven, Belgium
{vamichae,boxavier,gemmar,vangool}@vision.ee.ethz.ch, decapitb@ee.ethz.ch

Abstract. Superpixel algorithms aim to over-segment the image by grouping pixels that belong to the same object. Many state-of-the-art superpixel algorithms rely on minimizing objective functions to enforce color homogeneity. The optimization is accomplished by sophisticated methods that progressively build the superpixels, typically by adding cuts or growing superpixels. As a result, they are computationally too expensive for real-time applications. We introduce a new approach based on a simple hill-climbing optimization. Starting from an initial superpixel partitioning, it continuously refines the superpixels by modifying the boundaries. We define a robust and fast to evaluate energy function, based on enforcing color similarity between the boundaries and the superpixel color histogram. In a series of experiments, we show that we achieve an excellent compromise between accuracy and efficiency. We are able to achieve a performance comparable to the state-of-the-art, but in real-time on a single Intel i7 CPU at 2.8GHz.

Keywords: superpixels, segmentation.

1 Introduction

Many computer vision applications benefit from working with superpixels instead of just pixels [1,2,3]. Superpixels are of special interest for semantic segmentation, in which they are reported to bring major advantages. They reduce the number of entities to be labeled semantically and enable feature computation on bigger, more meaningful regions.

At the heart of many state-of-the-art (s-o-a) superpixel extraction algorithms lies an objective function, usually in the form of a graph. The trend has been to design sophisticated optimization schemes adapted to the objective function, and to strike a balance between efficiency and performance. Typically, optimization methods are built upon gradually adding cuts, or grow superpixels starting from some estimated centers. However, these superpixels algorithms come with a computational cost similar to systems producing entire semantic segmentations. For instance, Shotton *et al.* [4] report s-o-a segmentation within tenths of a second per image, which is as fast as s-o-a algorithms for superpixel extraction alone. Recent superpixel extraction methods emphasize the need for efficiency [5,6], but still their run-time is far from real-time.

A. Fitzgibbon et al. (Eds.): ECCV 2012, Part VII, LNCS 7578, pp. 13–26, 2012.

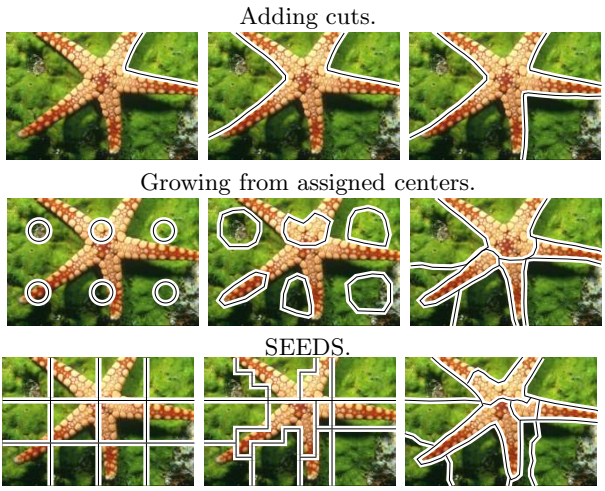


Fig. 1. Comparison of different strategies. Top: the image is progressively cut; Middle: the superpixels grow from assigned centers. Bottom: the presented method (SEEDS) proposes a novel approach: it initializes the superpixels in a gird, and continuously exchanges pixels on the boundaries between neighboring superpixels.

We try another way around the superpixel problem, which surprisingly, received little attention so far. Instead of incrementally building the superpixels by adding cuts or growing superpixels, we start from a complete superpixel partitioning, and we iteratively refine it. The refinement is done by moving the boundaries of the superpixels, or equivalently, by exchanging pixels between neighboring superpixels. We introduce an objective function that can be maximized efficiently, and is based on enforcing homogeneity of the color distribution of the superpixels, plus a term that encourages smooth boundary shapes. The optimization is based on a hill-climbing algorithm, in which a proposed movement for refining the superpixels is accepted if the objective function increases.

We show that the hill-climbing needs few operations to evaluate the energy function. In particular, it only requires one memory look-up when a single pixel from the boundary is moved. We will show this efficient exchange of pixels between superpixels enables the algorithm to run significantly faster than the s-o-a. We tested our approach on the Berkeley segmentation benchmark [7] and show that, to the best of our knowledge, the presented method (SEEDS) is faster than the fastest s-o-a methods and its performance is competitive with the best non-real-time methods. Indeed, it is able to run in real-time (30Hz) using a single CPU Intel i7 at 2.8GHz without GPUs or dedicated hardware.

2 Towards Efficiently Extracted Superpixels

In this Section, we revisit the literature on superpixel extraction, with special emphasis on their compromise between accuracy and run-time. The existing methods either work based on a gradual addition of cuts, or they gradually grow

superpixels starting from an initial set. We add a third approach, as illustrated in Fig. 1, which moves the boundaries from an initial superpixel partitioning.

Gradual Addition of Cuts. Typically, these methods are built upon an objective function that takes the similarities between neighboring pixels into account and use a graph to represent it. Usually, the nodes of the graph represent pixels, and the edges their similarities. Shi and Malik introduced the seminal Normalized Cuts algorithm [8]. It globally minimizes the graph-based objective function, by finding the optimal partition in the graph recursively. Normalized Cuts is computationally demanding, and there have been attempts to speed it up [9].

The algorithm by Moore *et al.* [10,11] finds the optimal cuts by using precomputed boundary maps. Yet, Achanta *et al.* [12] pointed out that the performance of this algorithm depends on the quality of such boundary maps. Veksler and Boykov [13] place overlapping patches over the image and assign each pixel to one of those by inferring a solution with graph-cuts. Based on this work, Zhang *et al.* [5] proposed an efficient algorithm that achieves 0.5 s per image. Another strategy to improve the efficiency of graph-based methods was introduced by Felzenszwalb and Huttenlocher [14]. They presented an agglomerative clustering of the nodes of the graph, which is faster than Normalized Cuts. However, [15,13] show that it produces superpixels of irregular size and shapes.

Recently, Liu *et al.* [6] introduced a new graph-based energy function and surpassed the previous results in terms of quality. Their method maximizes the entropy rate of the cuts in the graph, plus a balancing term that encourages superpixels of similar size. They show that maximizing the entropy rate favors the formation of compact and homogeneous superpixels, and they optimize it using a greedy algorithm. However, they also report that the algorithm takes about 2.5 s to segment an image of size 480×320.

Growing Superpixels from an Initial Set. Other methods not based on graphs, are Watersheds [16], Turbopixels [15], a Geodesic distances-based approach [17], SLIC [12], Consistent Segmentation [18] and Quick Shift [19]. The first two are based on growing regions until the superpixels are formed. The geodesic distance algorithm, SLIC and Consistent Segmentation start from a regular grid of centers or segments, and grow the superpixels by clustering pixels around the centers. At each iteration, the centers are updated, and the superpixels are grown again. The geodesic distance algorithm also accepts adding new centers. Quick-Shift performs fast, non-parametric clustering with a non-iterative algorithm. Even though these methods are more efficient than graph-based alternatives, they do not run in real-time, and in most cases they obtain inferior performance.

Our approach is related to some of these methods in the sense that it also starts from a regular grid. Yet, it does not share their bottleneck of needing to iteratively grow superpixels. Growing might imply computing some distance between the superpixel and all surrounding pixels in each iteration, which comes at a non-negligible cost. Our method bypasses growing superpixels from a center, because it directly exchanges pixels between superpixels by moving the boundaries.

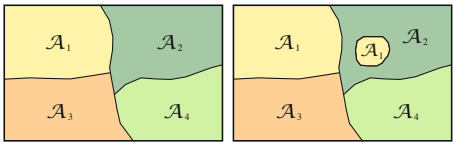


Fig. 2. Left: an example partitioning in $\mathcal{S}$, where the superpixels are connected. Right: the partitioning is in $\mathcal{C}$ but not in $\mathcal{S}$ as it is an invalid superpixel partitioning.

3 Superpixels as an Energy Maximization

The quality of a superpixel is measured by its property of grouping similar pixels that belong to the same object, and by how well it follows object boundaries. Therefore, a superpixel segmentation needs to strike a balance between consistent appearance inside superpixels and regular shape of the superpixel boundaries. We introduce the superpixel segmentation as an energy maximization problem where each superpixel is defined as a region with a color distribution and a shape of the boundary.

Let N be the number of pixels in the image, and K the number of superpixels that we want to obtain[1]. We represent a partitioning of the image into superpixels with the mapping

$$s : \{1, \ldots, N\} \rightarrow \{1, \ldots, K\}, \tag{1}$$

where $s(i)$ denotes the superpixel to which pixel i is assigned. Also, we can represent an image partitioning by referring to the set of pixels in a superpixel, which we denote as $\mathcal{A}_k$:

$$\mathcal{A}_k = \{i : s(i) = k\}, \tag{2}$$

and thus, $\mathcal{A}_k$ contains the pixels in superpixel k. The whole partitioning of the image is represented with the sets $\{\mathcal{A}_k\}$. Since a pixel can only be assigned to a single superpixel, all sets $\mathcal{A}_k$ are restricted to be disjoint, and thus, the intersection between any pair of superpixels is always the empty set: $\mathcal{A}_k \cap \mathcal{A}_{k'} = \emptyset$. In the sequel, we interchangeably use s or $\{\mathcal{A}_k\}$ to represent a partitioning of the image into superpixels.

A superpixel is valid if spatially connected as an individual blob. We define $\mathcal{S}$ as the set of all partitionings into valid superpixels, and $\bar{\mathcal{S}}$ as the set of invalid partitionings, as shown in Fig. 2. Also, we denote $\mathcal{C}$ as the more general set that includes all possible partitions (valid and invalid).

The superpixel problem aims at finding the partitioning $s \in \mathcal{S}$ that maximizes an objective function, or so called energy function. We denote the energy function as $E(s, I)$, where I is the input image. In the following, we will omit

[1] The number of desired superpixels K is assumed to be fixed, as is usual in most previous work, which allows for a comparison with the s-o-a.

the dependency of the energy function on I for simplicity of notation. Then, we define $s^\star$ as the partitioning that maximizes the energy function:

$$s^\star = \arg\max_{s \in \mathcal{S}} E(s). \tag{3}$$

This optimization problem is challenging because the cardinalities of $\mathcal{S}$ and $\mathcal{C}$ are huge. In fact, $|\mathcal{C}|$ is the Stirling number of the second kind, which is of the order of $\frac{K^n}{K!}$ [20]. What also renders the exploration of $\mathcal{S}$ difficult, is how $\mathcal{S}$ is embedded into $\mathcal{C}$. For each element in $\mathcal{S}$ there exists at least one element in $\bar{\mathcal{S}}$ which only differs in one pixel. This means that from any valid image partitioning, we are always one pixel away from an invalid solution.

4 Energy Function

This section introduces the energy function that is optimized, and which is defined as the sum of two terms. One term $H(s)$ is based on the likelihood of the color of the superpixels, and the other term $G(s)$ is a prior of the shape of the superpixel boundaries. Thus, the energy becomes

$$E(s) = H(s) + \gamma G(s), \tag{4}$$

where γ weighs the influence of each term, and is fixed to a constant value in the experiments.

4.1 Color Distribution Term: $H(s)$

The term $H(s)$ evaluates the color distribution of the superpixels. In this term, we assume that the color distribution of each superpixel is independent from the rest. We do not enforce color neighboring constraints between superpixels, since we aim at over-segmenting the image, and it might be plausible that two neighboring superpixels have similar colors. This is not to say that the neighboring constraints are not useful in principle, but our results suggest that without them we can still achieve excellent performance.

By definition, a superpixel is perceptually consistent and should be as homogeneous in color as possible. Nonetheless, it is unclear which is the best mathematical way to evaluate the homogeneity of color in a region. Almost each paper on superpixels in the literature introduces a new energy function to maximize, but none of them systematically outperforms the others. We introduce a novel measure on the color density distribution in a superpixel that allows for efficient maximization with the hill-climbing approach.

Our energy function is built upon evaluating the color density distribution of each superpixel. A common way to approximate a density distribution is discretizing the space into bins and building a histogram. Let λ be an entry in the color space, and $\mathcal{H}_j$ be a closed subset of the color space. $\mathcal{H}_j$ is a set of λ's that defines the colors in a bin of the histogram. We denote $c_{\mathcal{A}_k}(j)$ as the color histogram of the set of pixels in $\mathcal{A}_k$, and it is

$$c_{\mathcal{A}_k}(j) = \frac{1}{Z} \sum_{i \in \mathcal{A}_k} \delta(I(i) \in \mathcal{H}_j). \tag{5}$$

$I(i)$ denotes the color of pixel i, and Z is the normalization factor of the histogram. $\delta(\cdot)$ is the indicator function, which in this case returns 1 when the color of the pixel falls in the bin j.

Let $\Psi(c_{\mathcal{A}_k})$ be a quality measure of a color distribution, and we define $H(s)$ as an evaluation of such quality in each superpixel k, *i.e.* $H(s) = \sum_k \Psi(c_{\mathcal{A}_k})$. $\Psi(c_{\mathcal{A}_k})$ can be a function that enforces that the histogram is concentrated in one or few colors, *e.g.* the entropy of $c_{\mathcal{A}_k}$ might be a valid measure. We found that the following measure is advantageous:

$$\Psi(c_{\mathcal{A}_k}) = \sum_{\{\mathcal{H}_j\}} (c_{\mathcal{A}_k}(j))^2. \tag{6}$$

In the sequel we will show that this objective function can be optimized very efficiently by a hill-climbing algorithm, as histograms can be evaluated and updated efficiently. Observe that $\Psi(c_{\mathcal{A}_k})$ encourages homogeneous superpixels, since the maximum of $\Psi(c_{\mathcal{A}_k})$ is reached when the histogram is concentrated in one bin, which gives $\Psi(c_{\mathcal{A}_k}) = 1$. In all the other cases, the function is lower, and it reaches its minimum in case that all color bins take the same value. The main drawback of this energy function is that it does not take into account whether the colors are placed in bins far apart in the histogram or not. However, this is alleviated by the fact that we aim at over-segmenting the image, and each superpixel might tend to cover an area with a single color.

4.2 Boundary Term: $G(s)$

The term $G(s)$ evaluates the shape of the superpixel. We call it boundary term and it penalizes local irregularities in the superpixel boundaries. Depending on the application, this term can be chosen to enforce different superpixel shapes, *e.g.* $G(s)$ can be chosen to favor compactness, smooth boundaries, or even proximity to edges based on an edge map. It seems subjective which type of shape is preferred, and in the end the only objective metric is how well object boundaries are recovered. We introduce $G(s)$ as a local smoothness term, and even though it is simple and local, we will show in experiments that it is competitive with compactness-based methods.

Our boundary term places a $N \times N$ patch around each pixel in the image. Let $\mathcal{N}_i$ be the patch around pixel i, *i.e.* the set of pixels that are in a squared area of size $N \times N$ around pixel i. Each patch counts the number of different superpixels present in a local neighborhood. In analogy to the color distribution term, we use a quality measure based on a histogram. We define the histogram of superpixel labels in the area $\mathcal{N}_i$ as

$$b_{\mathcal{N}_i}(k) = \frac{1}{Z} \sum_{j \in \mathcal{N}_i} \delta(j \in \mathcal{A}_k). \tag{7}$$

Note that this histogram has K bins, and each bin corresponds to a superpixel label. The histogram counts the amount of pixels from superpixel k in the patch.

Near the boundaries, the pixels of a patch can belong to several superpixels, and away from the boundaries they belong to one unique superpixel. We consider

that a superpixel has a better shape when most of the patches contain pixels from one unique superpixel. We define $G(s)$ using the same measure of quality as in $H(s)$, because, as we will show, it yields an efficient optimization algorithm. Thus, it becomes

$$G(s) = \sum_i \sum_k (b_{\mathcal{N}_i}(k))^2. \tag{8}$$

If the patch $\mathcal{N}_i$ contains a unique superpixel, $G(s)$ is at its maximum. Observe that it is not possible that such maximum is achieved in all pixels, because the patches near the boundaries contain multiple superpixel labelings. However, penalizing patches containing several superpixel labelings reduces the amount of pixels close to a boundary, and thus enforces regular shapes. Furthermore, in the case that a boundary yields a shape which is not smooth, the amount of patches that take multiple superpixel labels is higher. A typical example to avoid is a section as thin as 1 pixel extending into neighboring superpixels. The smoothing term penalizes such cases, among others, and thus encourages a smooth labeling between superpixels.

5 Superpixels via Hill-Climbing Optimization

We introduce a hill-climbing optimization for extracting superpixels. Hill-climbing is an optimization algorithm that iteratively updates the solution by proposing small local changes at each iteration. If the energy function of the proposed partitioning increases, the solution is updated. We denote $s \in \mathcal{S}$ as the proposed partitioning, and $s_t \in \mathcal{S}$ the lowest energy partitioning found at the instant t. A new partitioning s is proposed by introducing local changes at s_t, which in our case consists of moving some pixels from one superpixel to its neighbors. An iteration of the hill-climbing algorithm can be extremely efficient, because small changes to the partitioning can be evaluated very fast in practice.

An overview of the hill-climbing algorithm is shown in Fig. 3. After initialization, the algorithm proposes new partitionings at two levels of granularity: pixel-level and block-level. Pixel-level updates move a superpixel boundary by 1 pixel, while block-level updates move a block of pixels from one superpixel to another. We will show that both types of update can be seen as the same operation, at a different scale.

5.1 Initialization

In hill-climbing, in order to converge to a solution close to the global optimum ($s^\star$), it is important to start from a good initial partitioning. We propose a regular grid as a first rough partitioning, which obeys the spatial constraints of the superpixels to be in $\mathcal{S}$. In experiments, we found that when evaluating a grid against the standard evaluation metrics, the performance is respectable: the grid achieves a reasonable over-segmentation, but of course fails at recovering the object boundaries. However, observe that object boundaries are maximally $S/2$ pixels away from the grid boundaries, where S is the width of each grid cell.

```
s_t=initialize();
while t < t_stop do
    s = Propose(s_t);
    if E(s) > E(s_t) then
        s_t = s;
    end
end
s* = s_t;
```

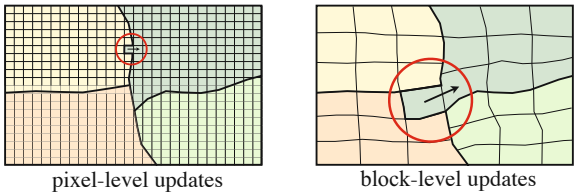


Fig. 3. Left: algorithm. Right: movements at pixel-level and at block-level

We think that this is a good reason to use a grid of superpixels to initialize s_t; besides, it justifies using hill-climbing optimization for extracting superpixels, since the initialization is relatively close to the optimal solution.

5.2 Proposing Pixel-Level and Block-Level Movements

In each iteration, the algorithm proposes a new partitioning s based on the previous one s_t. The elements that are changed from s_t to s are either single pixels or blocks of pixels that are moved to a neighboring superpixel. We denote $\mathcal{A}_k^l$ as a candidate set of one or more pixels to be exchanged from the superpixel $\mathcal{A}_k$ to its neighbor $\mathcal{A}_n$. In the case of pixel-level updates $\mathcal{A}_k^l$ contains one pixel (singleton), and in the case of block-level updates $\mathcal{A}_k^l$ contains a small set of pixels, as illustrated in Fig 3. At each iteration of the hill-climbing, we generate a new partitioning by randomly picking $\mathcal{A}_k^l$ from all boundary pixels or blocks with equal probability, and we assign the chosen $\mathcal{A}_k^l$ to a random superpixel neighbor $\mathcal{A}_n$. In case it generates an invalid partitioning, which can only happen when a boundary movement splits a superpixel in two parts, it is discarded.

Block-level updates are used for reasons of efficiency, as they allow for faster convergence, and help to avoid local maxima. In order to define the blocks of pixels $\mathcal{A}_k^l$, we initially divide the superpixels in regions of $R \times R$ pixels. The bigger the size of R, the faster the hill-climbing optimization might converge, because we consider bigger movements. Yet, when making R bigger, the block of pixels have higher chances to contain multiple colors, and hence, not to be perceptually homogeneous. In the experiments section, we show the benefit of block-level updates, and we determine the optimal R.

5.3 Evaluating Pixel-Level and Block-Level Movements

The proposed partitioning s is evaluated using the energy function (Eq. 4). In the following we describe the efficient evaluation of $E(s)$, and the efficient updating of the color distributions in case s is accepted. The proofs of the propositions in this section are provided in the supplementary material.

Color Distribution Term. We introduce an efficient way to evaluate $H(s)$ based on the intersection distance. Recall that the intersection distance between two histograms is

$$\mathbf{int}(c_{\mathcal{A}_a}, c_{\mathcal{A}_b}) = \sum_j \min\{c_{\mathcal{A}_a}(j), c_{\mathcal{A}_b}(j)\}, \tag{9}$$

where j is a bin in the histogram. Observe that it only involves $|\{\mathcal{H}_j\}|$ comparisons and sums, where $|\{\mathcal{H}_j\}|$ is the number of bins of the histogram. Recall that $\mathcal{A}_k^l$ is the set of pixels that are candidates to be moved from the superpixel $\mathcal{A}_k$ to $\mathcal{A}_n$. We base the evaluation of $H(s) > H(s_t)$ on the following Proposition.

Proposition 1. *Let the sizes of $\mathcal{A}_k$ and $\mathcal{A}_n$ be similar, and $\mathcal{A}_k^l$ much smaller,* i.e. $|\mathcal{A}_k| \approx |\mathcal{A}_n| \gg |\mathcal{A}_k^l|$. *If the histogram of $\mathcal{A}_k^l$ is concentrated in a single bin, then*

$$\mathbf{int}(c_{\mathcal{A}_n}, c_{\mathcal{A}_k^l}) \geq \mathbf{int}(c_{\mathcal{A}_k \setminus \mathcal{A}_k^l}, c_{\mathcal{A}_k^l}) \iff H(s) \geq H(s_t). \tag{10}$$

Proposition 1 can be used to evaluate whether the energy function increases or not by simply computing two intersection distances. However, it makes two assumptions about the superpixels. The first is that the size of $\mathcal{A}_k^l$ is much smaller than the size of the superpixel, and that both superpixels have a similar size. When $\mathcal{A}_k^l$ is a single pixel or a small block of pixels, it is reasonable to assume that this is true for most cases. The second assumption is that the histogram of $\mathcal{A}_k^l$ is concentrated in a single bin. This is always the case if $\mathcal{A}_k^l$ is a single pixel, because there is only one color. In the block-level case it is reasonable to expect that the colors in each block are concentrated in few bins. In the experiments section, we show that when running the algorithm these assumptions hold in 93% of the cases.

Interestingly, in the case of evaluating a pixel-level update, the computation of the intersection can be achieved with a single access to memory. This is because the color histogram of a pixel has a single bin activated with a 1, and hence, the intersection distance is the value of the histogram of the superpixel.

Boundary Term. During pixel-level updates, $G(s)$ is evaluated based on the following proposition.

Proposition 2. *Let $\{b_{\mathcal{N}_i}(k)\}$ be the histograms of the superpixel labelings computed at the partitioning s_t (see Eq. (7)). $\mathcal{A}_k^l$ is a pixel, and $\mathcal{K}_{\mathcal{A}_k^l}$ the set of pixels whose patch intersects with that pixel,* i.e. $\mathcal{K}_{\mathcal{A}_k^l} = \{i : \mathcal{A}_k^l \in \mathcal{N}_i\}$. *If the hill-climbing proposes moving a pixel $\mathcal{A}_k^l$ from superpixel k to superpixel n, then*

$$\sum_{i \in \mathcal{K}_{\mathcal{A}_k^l}} (b_{\mathcal{N}_i}(n) + 1) \geq \sum_{i \in \mathcal{K}_{\mathcal{A}_k^l}} b_{\mathcal{N}_i}(k) \iff G(s) \geq G(s_t). \tag{11}$$

Proposition 2 shows that the difference in $G(s)$ can be evaluated with just a few sums of integers.

In case of block-level updates, the block boundaries tend to be smooth. Block boundaries are fixed unless they coincide with a superpixel boundary, in which case they are updated jointly in the pixel-level updates. However, when assigning a block to a new superpixel, a small irregularity might be introduced at the junctions. Smoothing these out requires pixel-level movements, thus they are smoothed in subsequent pixel-level iterations of the algorithm.

Updating the Color Distributions. Once a new partition has been accepted, the histograms of $\mathcal{A}_k$ and $\mathcal{A}_n$ have to be updated efficiently. In the pixel-level case, this update can be achieved with a single increment and decrement of bin j of the the respective histograms. In the block-level case, this update is achieved by subtracting $c_{\mathcal{A}_k^l}$ from $c_{\mathcal{A}_k}$ and adding it to $c_{\mathcal{A}_n}$.

5.4 Iterations

The hill-climbing updates s iteratively. Block-level updates are more expensive but move more pixels at the same. Hence, it is better to do more block-level updates at the beginning of the algorithm. Ideally, the size of R would increase as the algorithm progresses. However, the computational overhead of recomputing the histograms for a variable R is too high. In our implementation, the algorithm begins with a sweep of block-level updates over the entire image, and then alternates between pixel-level and block-level updates.

5.5 Termination

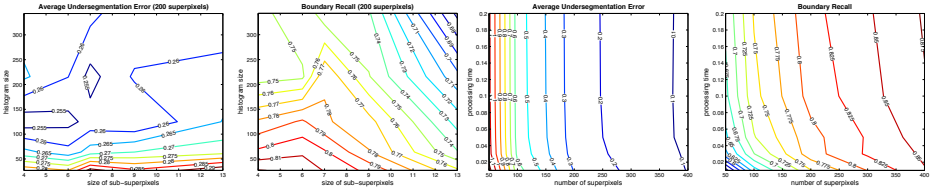
When stopping the algorithm, one obtains a valid image partitioning with a quality depending on the allowed run-time. The longer the algorithm is allowed to run, the higher the value of the objective function will get. We can set t_{stop} depending on the application, or we can even assign a time budget *on the fly*.

We believe this to be a crucial property for on-line applications, but nonetheless one that has received little attention in the context of superpixel extraction so far. In graph-based superpixel algorithms, one has to wait until all cuts have been added to the graph, and in methods that grow superpixels, one has to wait until the growing is done, the cost of which is not negligible. The hill-climbing approach uses a lot more iterations than previous methods, but each iteration is done extremely fast. This enables stopping the algorithm at any given time, because the time to finish the current iteration is negligible.

6 Experiments

We report results on the Berkeley Segmentation Dataset (BSD) [7], using the standard metrics to evaluate superpixels, as used in most recent superpixel papers [6,12,13,15,17]. The BSD consists of 500 images split into 200 training, 100 validation and 200 test images. We use the training images to set the few parameters that need to be tuned and report the results based on the 200 test images. We compare SEEDS to defined baselines and to the current s-o-a methods. We compute the standard metrics used to evaluate the performance of superpixel algorithms, which are undersegmentation error (UE), boundary recall (BR) and achievable segmentation accuracy (ASA). We use exactly the same metrics as used in [6][2]. Recall that for the UE the lower the better, and for BR and ASA the higher the better. For completeness we also report the precision-recall curves for the contour detection benchmark proposed in [21]. This standard benchmark

[2] We found that in previous works, the evaluation of UE slightly changes depending on the paper, because it is not clear in this measure how to treat the pixels that lie on the boundaries. For instance, [12] reports a 5% tolerance margin for the overlap; and in [6] the boundaries are removed from the labeling before computing the UE. See the supplementary material for more details.



(a) (b)

Fig. 4. (a): Evaluation of SEEDS on the training set of BSD, changing the number of bins in the histogram and the size of the block of pixels (R) for 200 superpixels. (b): Evaluation on the test set of BSD: UE and BR versus time and number of superpixels.

allows for an additional evaluation of the boundary performance of the different superpixel algorithms. All experiments are done using a single CPU (2.8GHz i7). We do not use any parallelization, GPU or dedicated hardware. For further details on the metrics we refer to the supplementary material.

6.1 Parameters

We use LAB color space, which in our experiments yields the highest performance. The choice of weight γ of $G(s)$ and size of the local neighborhood $N \times N$ is difficult to evaluate because there is no standard metric for smoothness or compactness of a superpixel in the literature. In fact, there is a trade-off between increasing the smoothness and the performance on the existing metrics (UE, BR and ASA). Therefore, in order to maximize the performance, we set γ to 1 and $N \times N$ to the minimum size 3×3. In the next subsection we will show that this choice recovers object boundaries well, as it matches the s-o-a border recall performance. Also note that the more the algorithm iterates, the smoother the boundaries become.

Only two parameters need to be tuned: the number of bins in the histograms and R. These parameters are tuned on a subset of the BSD training set. In Fig. 4a, we report the UE and the BR while changing the histogram size and the value of R using 200 superpixels. We set the number of bins to 5 bins per color channel (125 bins in total), and we found that the blocks of pixels should best be 5×5 pixels in size for the 481×321 image size of the BSD.

We show the performance of SEEDS on the test set, depicted in Fig. 4b. The UE and BR are shown as a function of run-time and number of superpixels. The longer the algorithm runs, the better the performance gets in all cases. Also, the performance is better when adding more superpixels, which is not surprising because the image is more over-segmented. We evaluated the assumptions from Proposition 1 over all the updates when segmenting the training set, by explicitly computing the energy function in each iteration and comparing it to the intersection distance. This experiment shows that the approximation holds for 97% of the pixel-level updates, and for 89% of the block-level updates.

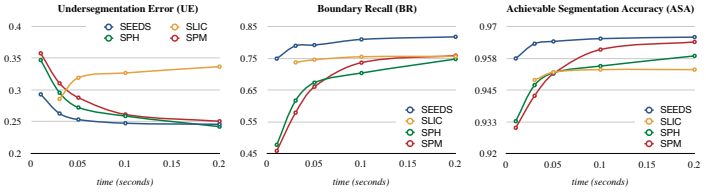


Fig. 5. Evaluation of SEEDS, the baselines SPH and SPM, and SLIC, versus run-time

6.2 Histograms and Block-Level Updates

In order to demonstrate the speed and performance benefit of block-level updates, we introduce a baseline method without block-level updates called *SPH* (Pixel-level using Histograms). This method is identical to SEEDS, except that it only uses pixel-level updating. To demonstrate the benefit of using histograms as a color distribution, we introduce a second baseline using the mean-based distance measure from SLIC [12], called *SPM* (Pixel-level using Means).

The results of this experiment are presented in function of available processing time, shown in Fig. 5. The results show that SEEDS converges faster than SLIC: where SLIC requires 200 ms to compute 10 iterations, SEEDS only takes 20 ms to produce a similar result. The experiment also shows that SEEDS using histograms (SPH) converges faster than using means (SPM), and that both converge to similar results. Furthermore, it shows that SEEDS converges faster when using block updates (SEEDS) than without (SPH), and to a better result, as it is less prone to getting stuck in local maxima. There is an anomaly where SLIC's UE seems to get worse with each iteration. We believe that this caused by SLIC's stray labels, which are only removed at the end of all iterations and might affect the performance during the iterations.

6.3 Comparison to State-of-the-Art

We compare SEEDS to s-o-a methods Entropy Rate Superpixels[3] [6] (ERS), to SLIC[4] [12], and to Felzenszwalb and Huttenlocher (FH)[5] [14]. ERS is considered s-o-a in terms of performance, and SLIC is the fastest method available in the literature at 5 Hz. We evaluated two versions of SEEDS, one which runs at 30Hz, and another at 5Hz with more iterations of the hill-climbing. ERS ran at less than 1Hz in this experiment. The results (Fig. 6) show that SEEDS matches the UE and BR of ERS, and slightly outperforms the ASA. FH has a better BR, but a significantly worse UE. Note that, as FH does not output a fixed number of superpixels, the parameters are set such that the desired number of superpixels with the best performance were obtained. We also show the performance of a plain grid (GRID) as a baseline to validate it as an initialization.

[3] Code available at `http://www.umiacs.umd.edu/~mingyliu/research`
[4] Code available at `http://ivrg.epfl.ch/supplementary_material/RK_SLICSuperpixels`
[5] Code available at `http://www.cs.brown.edu/~pff/segment/`

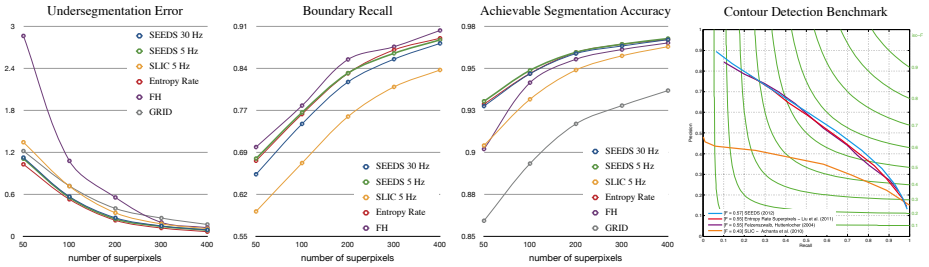


Fig. 6. Evaluation of SEEDS versus s-o-a on the BSD test set. In all experiments we used one Intel i7@2.8GHz CPU.

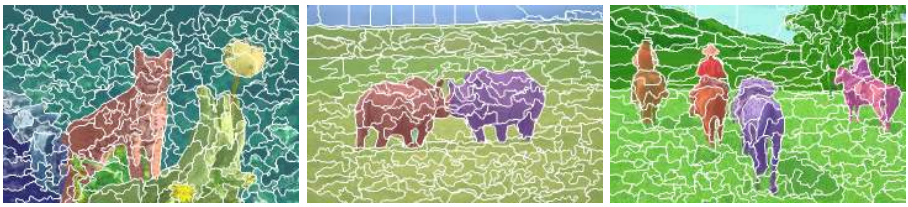


Fig. 7. Example SEEDS segmentations with 200 superpixels. The ground truth segments are color coded and blended on the images. The superpixel boundaries are shown in white.

Additionally, we present results based on the BSDS300 contour detection benchmark [21], by running the superpixel algorithms as a contour detector. This is achieved by extracting superpixels on 12 different scales, ranging from 6 to 600 superpixels, and averaging the resulting boundaries. This is repeated for each superpixel algorithm. SEEDS outperforms the other superpixel methods on this metric while being orders of magnitude faster. Some examples of the segmentation results with 200 superpixels are shown in Fig. 7.

7 Conclusions

We have presented a superpixel algorithm that achieves an excellent compromise between accuracy and efficiency. It is based on a hill-climbing optimization with efficient exchanges of pixels between superpixels. The energy function that is maximized is based on enforcing homogeneity of the color distribution within superpixels. The hill-climbing algorithm yields a very efficient evaluation of this energy function by using the intersection distance between histograms. Its runtime can be controlled *on the fly*, and we have shown the algorithm to run successfully in real-time, while staying competitive with the s-o-a on standard benchmark datasets. We use a single CPU and we do not use any GPU or dedicated hardware. The source code is available online[6].

[6] Code available at `http://www.vision.ee.ethz.ch/software`

Acknowledgments. This work has been in part supported by the European Commission projects RADHAR (FP7 ICT 248873) and IURO (FP7 ICT 248314). We thank the authors of [6] for discussion about the evaluation benchmark.

References

1. Wang, S., Lu, H., Yang, F., Yang, M.H.: Superpixel tracking. In: ICCV (2011)
2. Fulkerson, B., Vedaldi, A., Soatto, S.: Class segmentation and object localization with superpixel neighborhoods. In: ICCV (2009)
3. Boix, X., Gonfaus, J.M., van de Weijer, J., Bagdanov, A., Serrat, J., Gonzàlez, J.: Harmony potentials. IJCV (2011)
4. Shotton, J., Johnson, M., Cipolla, R.: Semantic texton forests for image categorization and segmentation. In: CVPR (2008)
5. Zhang, Y., Hartley, R., Mashford, J., Burn, S.: Superpixels via pseudo-boolean optimization. In: ICCV (2011)
6. Liu, M.Y., Tuzel, O., Ramalingam, S., Chellappa, R.: Entropy rate superpixel segmentation. In: CVPR (2011)
7. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: ICCV (2001)
8. Shi, J., Malik, J.: Normalized cuts and image segmentation. PAMI (2000)
9. Mori, G.: Guiding model search using segmentation. In: ICCV (2005)
10. Moore, A., Prince, S., Warrell, J., Mohammed, U., Jones, G.: Superpixel lattices. In: CVPR (2008)
11. Moore, A., Prince, S., Warrell, J.: Lattice cut. In: CVPR (2010)
12. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süsstrunk, S.: SLIC superpixels compared to state-of-the-art superpixel methods. PAMI (2012)
13. Veksler, O., Boykov, Y., Mehrani, P.: Superpixels and Supervoxels in an Energy Optimization Framework. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part V. LNCS, vol. 6315, pp. 211–224. Springer, Heidelberg (2010)
14. Felzenszwalb, P., Huttenlocher, D.: Efficient graph-based image segmentation. IJCV (2004)
15. Levinshtein, A., Stere, A., Kutulakos, K., Fleet, D., Dickinson, S., Siddiqi, K.: Turbopixels: Fast superpixels using geometric flows. PAMI (2009)
16. Vincent, L., Soille, P.: Watersheds in digital spaces: An efficient algorithm based on immersion simulations. PAMI (1991)
17. Zeng, G., Wang, P., Wang, J., Gan, R., Zha, H.: Structure-sensitive superpixels via geodesic distance. In: ICCV (2011)
18. Zitnick, C., Jojic, N., Kang, S.: Consistent segmentation for optical flow estimation. In: ICCV (2005)
19. Vedaldi, A., Soatto, S.: Quick Shift and Kernel Methods for Mode Seeking. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part IV. LNCS, vol. 5305, pp. 705–718. Springer, Heidelberg (2008)
20. Sharp, H.: Cardinality of finite topologies. J. Combinatorial Theory (1968)
21. Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. PAMI (2011)