Abstract
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.

1 Introduction

Many computer vision applications benefit from working with superpixels instead of just pixels [1–3].
However, these superpixels algorithms come with a computational cost similar to systems producing entire semantic segmentations.
Instead of incrementally building the superpixels by adding cuts or growing superpixels, we start from a complete superpixel partitioning, and we iteratively refine it.
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 decreases.

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.

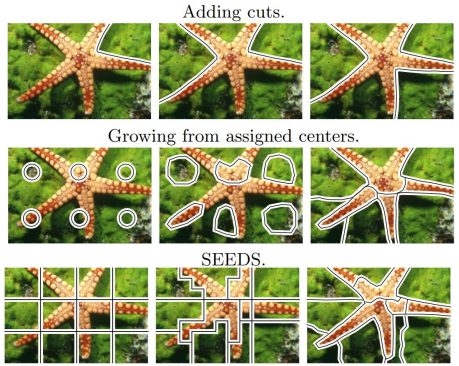


Fig. 1. Comparison of different strategies. Top: the image is progressively cut; Middle: the superpixels grow from assigned centers. Bottom: the presented method (SEEDS) proposes a novel approach: it initializes the superpixels in a gird, and continuously exchanges pixels on the boundaries between neighboring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

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.

Growing Superpixels from an Initial Set.
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.

Ours.
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.

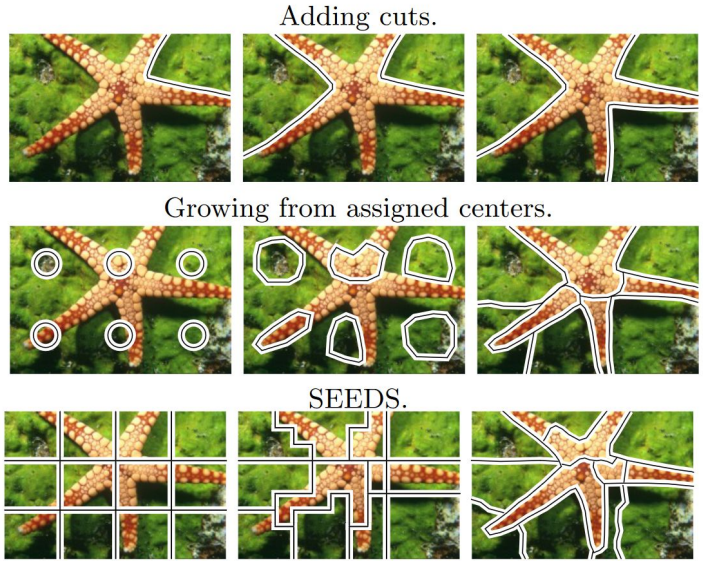


Fig. 1. Comparison of different strategies. Top: the image is progressively cut; Middle: the superpixels grow from assigned centers. Bottom: the presented method (SEEDS) proposes a novel approach: it initializes the superpixels in a gird, and continuously exchanges pixels on the boundaries between neighboring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

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.

$$s : \{1, \ldots, N\} \rightarrow \{1, \ldots, K\}, \quad (1)$$

: mapping from image partitioning into superpixels
N : the number of pixels in the image
K : the number of superpixels that we want to obtain

$s(i)$: superpixel to which pixel i is assigned

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

: pixels in superpixel k

$\{\mathcal{A}_k\}$: the whole partitioning of the image

A superpixel is valid if spatially connected as an individual blob.

$\mathcal{S}$: the set of all partitionings into valid superpixels

$\bar{\mathcal{S}}$: the set of invalid partitionings

$\mathcal{C}$: the more general set that includes all possible partitions (valid and invalid)

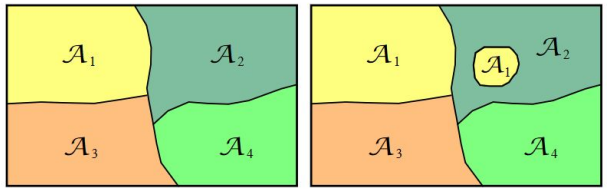


Fig. 2. Left: an example partitioning in $\mathcal{S}$, where the superpixels are connected. Right: the partitioning is in $\mathcal{C}$ but not in $\mathcal{S}$ as it is an invalid superpixel partitioning.

The superpixel problem aims at finding the partitioning s ∈ S that maximizes an objective function, or so called energy function.

$$s^\star = \arg\max_{s \in \mathcal{S}} E(s). \qquad (3)$$: the partitioning that maximizes the energy function

$E(s, I)$: $E(s)$: energy function

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

$$E(s) = H(s) + \gamma G(s), \qquad (4)$$: energy function

4.1 Color Distribution Term: H(s)

$H(s)$: 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.

By definition, a superpixel is perceptually consistent and should be as homogeneous in color as possible.
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.

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

: color histogram of the set of pixels in $\mathcal{A}_k$

$I(i)$ 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.

$\mathcal{H}_j$: closed subset of the color space : set of λ's that defines the colors in a bin of the histogram

λ : entry in the color space

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

: quality measure of a color distribution
: function that enforces that the histogram is concentrated in one or few colors

$H(s) = \sum_k \Psi(c_{\mathcal{A}_k})$: evaluation of such quality in each superpixel k

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

4.2 Boundary Term: G(s)

$G(s)$: evaluates the shape of the superpixel

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.
We introduce G(s) as a local smoothness term.

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

: histogram of superpixel labels in the area $\mathcal{N}_i$

$\mathcal{N}_i$: set of pixels that are in a squared area of size N × N around pixel i

Each patch counts the number of different superpixels present in a local neighborhood.
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.

$$G(s) = \sum_i \sum_k (b_{\mathcal{N}_i}(k))^2. \quad (8)$$: measure of quality

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 decreases, the solution is updated.

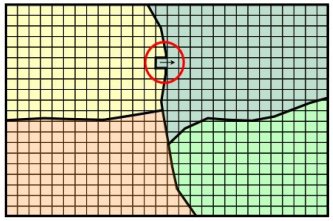
$s \in \mathcal{S}$: the proposed partitioning

$s_t \in \mathcal{S}$: the lowest energy partitioning found at the instant t.

A new partitioning s is proposed by introducing local changes at st, 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.

$s_t = \text{initialize}();$
while $t < t_{stop}$ **do**
  $s = \text{Propose}\ (s_t);$
  if $E(s) < E(s_t)$ **then**
    $s_t = s;$
  end
end
$s^\star = s_t;$


pixel-level updates

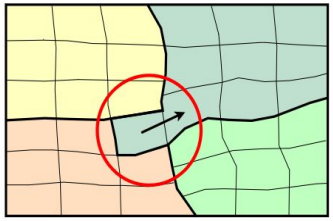

block-level updates

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

5.1 Initialization

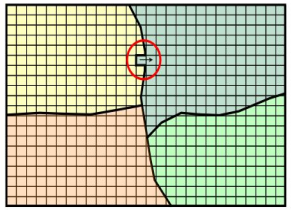
In hill-climbing, in order to converge to a solution close to the global optimum $(s^\star)$, it is important to start from a good initial partitioning.
We propose a regular grid as a first rough partitioning, which obeys the spatial constraints of the superpixels to be in $\mathcal{S}$.

5.2 Proposing Pixel-level and Block-level Movements

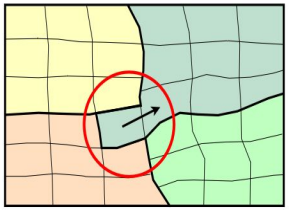
$\mathcal{A}_k^l$: candidate set of one or more pixels to be exchanged from the superpixel $\mathcal{A}_k$ to its neighbor $\mathcal{A}_n$

SEEDS: Superpixels Extracted via Energy-Driven Sampling

```
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;
```



pixel-level updates



block-level updates

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 detere 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.

$$E(s) = H(s) + \gamma G(s), \qquad (4)$$

Color Distribution Term.

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

: intersection distance between two histograms

$|\{\mathcal{H}_j\}|$: the number of comparisons and sums : the number of bins of the histogram

j : bin in the histogram

$\mathcal{A}_k^l$: candidate set of one or more pixels to be exchanged from the superpixel $\mathcal{A}_k$ to its neighbor $\mathcal{A}_n$

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

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

Proposition 1 can be used to evaluate whether the energy function increases or not by simply computing two intersection distances.

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. (8)). $\mathcal{A}_k^l$ 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*

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

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

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

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*.

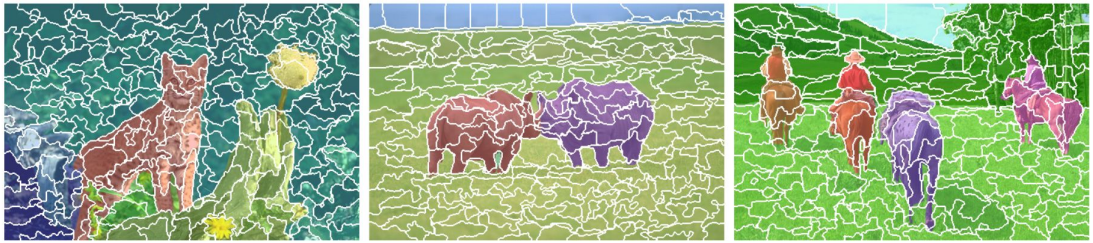


Fig. 7. Example SEEDS segmentations with 200 superpixels. The ground truth segments are color coded and blended on the images. The superpixel boundaries are shown in white.

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.