Superpixel Hierarchy

Xing Wei, Qingxiong Yang†, Member, IEEE, Yihong Gong‡, Fellow, IEEE, Narendra Ahuja, Fellow, IEEE, and Ming-Hsuan Yang†, Senior Member, IEEE

Abstract—Superpixel segmentation has been one of the most important tasks in computer vision. In practice, an object can be represented by a number of segments at finer levels with consistent details or included in a surrounding region at coarser levels. Thus, a superpixel segmentation hierarchy is of great importance for applications that require different levels of image details. However, there is no method that can generate all scales of superpixels accurately in real time. In this paper, we propose the superhierarchy algorithm which is able to generate multi-scale superpixels as accurately as the state-of-the-art methods but with one to two orders of magnitude speed-up. The proposed algorithm can be directly integrated with recent efficient edge detectors to significantly outperform the state-of-the-art methods in terms of segmentation accuracy. Quantitative and qualitative evaluations on a number of applications demonstrate that the proposed algorithm is accurate and efficient in generating a hierarchy of superpixels.

Index Terms—Superpixel, segmentation, Borůvka algorithm.

I. INTRODUCTION

Superpixels are groupings of pixels with perceptual meanings that serve as primitives to facilitate further analysis. In recent years, superpixels have been the key building blocks for numerous algorithms as they significantly reduce the number of image primitives and contain important visual information when compared to raw pixels. In this paper, we propose a computationally efficient hierarchical superpixel segmentation algorithm that can be applied to a wide range of image analysis tasks. In order to achieve such versatile utilities, a superpixel method should have the following properties:

- **Adherent boundary:** Superpixels should adhere well to image boundaries such that each superpixel only overlaps with one object.

- **Computational efficiency:** The computational complexity for an efficient segmentation algorithm should be independent of the number of superpixels and linear/sublinear in the image size.

- **Hierarchical segmentation:** Superpixel segmentation results at different levels should be close to the human vision system. Numerous algorithms can benefit from multi-resolution representations of images and hierarchical superpixels can be applied to these tasks.

- **Preserved topology:** Superpixels should conform to a simple topology such that neighborhood relationships can be maintained.

While the recent years have seen considerable progress in superpixel segmentation [5], [6], [9]–[11], [13]–[17], the state-of-the-art methods possess only one to two of these properties which limit their application domains. For instance, Liu et al. [6] proposed a graph-based method that has high segmentation accuracy. However, it is computationally prohibitive for real-time applications. The SEEDS algorithm [10] achieves a compromise between accuracy and efficiency but its run-time depends on the number of superpixels. One class of approaches [13], [18] generate superpixels that conform to a grid topology which can be conveniently used by several vision tasks. However, the computational complexity of these methods is high and the segmentation accuracy is lower than the state-of-the-art methods [11]. In addition to grid topology, several algorithms [7], [19], [20] use tree structures of regions to represent images. Felzenszwalb and Huttenlocher [5] proposed a method that can accommodate such a structure by adding edges between segments. Nevertheless, its under-segmentation error is high as shown by the recent superpixel benchmark evaluation results [21]. Furthermore, while vision algorithms [22]–[24] benefit from hierarchical or multi-scale segmentation, most superpixel methods do not generate such results directly. Consequently, the same superpixel algorithm needs to be performed several times to generate superpixels at different scales, which increase the computational cost.

In this work, we present a super hierarchy (SH) algorithm that enjoys all the above-mentioned properties (see Table I). Our method efficiently constructs a superpixel hierarchy that can generate any number of superpixels (between one and the number of pixels) on the fly (see Figure 1) in a tree structure. Extensive experiments demonstrate that our algorithm performs favorably against the state-of-the-art methods in terms of accuracy and efficiency.

II. RELATED WORK

There is a rich literature on image segmentation. In this section, we discuss the most relevant methods to this work in...
TABLE I

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Achievable segmentation accuracy (average on BSDS500 [1])</td>
<td>93.3%</td>
<td>94.1%</td>
<td>94.9%</td>
<td>94.7%</td>
<td>95.0%</td>
<td>95.1%</td>
</tr>
<tr>
<td>Under-segmentation error (average on BSDS500)</td>
<td>12.6%</td>
<td>11.5%</td>
<td>10.1%</td>
<td>10.4%</td>
<td>10.0%</td>
<td>9.7%</td>
</tr>
<tr>
<td>Boundary recall (average on BSDS500)</td>
<td>78.0%</td>
<td>67.2%</td>
<td>75.5%</td>
<td>72.9%</td>
<td>78.7%</td>
<td>80.8%</td>
</tr>
<tr>
<td>Semantics segmentation accuracy (using [3] on MSRC [2])</td>
<td>63.8%</td>
<td>65.4%</td>
<td>65.5%</td>
<td>65.8%</td>
<td>65.3%</td>
<td>66.6%</td>
</tr>
<tr>
<td>Property 2: Segmentation speed</td>
<td>328 ms(^i)</td>
<td>108 ms(^i)</td>
<td>689 ms</td>
<td>52 ms</td>
<td>302 ms</td>
<td>31 ms</td>
</tr>
<tr>
<td>Complexity of obtaining m scales of superpixels</td>
<td>O(m)</td>
<td>O(m)</td>
<td>O(m)</td>
<td>O(m)</td>
<td>O(m)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Property 3: Hierarchical segmentation</td>
<td>1810 ms(^i)</td>
<td>554 ms</td>
<td>4160 ms</td>
<td>282 ms</td>
<td>1720 ms</td>
<td>35 ms</td>
</tr>
<tr>
<td>Saliency detection (using [4] on PASCAL-S(^i) [12])</td>
<td>0.187</td>
<td>0.190</td>
<td>0.189</td>
<td>0.186</td>
<td>0.190</td>
<td>0.181</td>
</tr>
<tr>
<td>Run-time of superpixels(^i)</td>
<td>0.187</td>
<td>0.190</td>
<td>0.189</td>
<td>0.186</td>
<td>0.190</td>
<td>0.181</td>
</tr>
<tr>
<td>Mean absolute error(^i)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Property 4: Maintain topology</td>
<td>Stereo matching error (using [7] on Middlebury [8])</td>
<td>8.10%</td>
<td>-</td>
<td>8.6%</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

\(^i\)Reported time includes parameter search \(^i\) Using 5 scales of superpixels \(^i\) Average size: 500 × 356 pixels

Fig. 1. Super hierarchy. Segmentations with 16, 256, 4096, and 65536 superpixels on an image of 1200 × 582 pixels are shown. Superpixels of all scales are obtained at once.

In [25], an image is partitioned into homogeneous regions of all shape, size, as well as degree of photometric homogeneity, and organized in a hierarchical tree. Nodes at the upper levels correspond to large segments while their children nodes capture finer details. A connected segmentation tree [26] includes additional edges to sibling nodes by introducing neighboring Voronoi regions. This segment tree structure has been applied to image classification, semantic image segmentation and object detection [27].

Superpixels can also be grouped based on region contours. In [1] and [28] the output of a contour detector is transformed into a hierarchical region tree. To this end, it uses the oriented watershed transform to construct a set of initial regions followed by an agglomerative clustering procedure to construct a hierarchical representation. This hierarchical segmentation method has been widely used in recognition and detection problems [29], [30]. Jain et al. [31] proposed a method to learn a similarity function using reinforcement learning for agglomerating superpixels hierarchies. The method shows good performance on brain images from serial electron microscopy. Recently, a hierarchical edge-weighted Voronoi tessellation algorithm [32] is proposed for generating multiscale superpixels and supervoxels.

A. Hierarchical Image Segmentation

Hierarchical image segmentation methods generate a set of segments with different details in which the ones at the coarser levels are composed of regions at the finer levels.

B. Efficient Superpixel Extraction

Although image segmentation plays an important role in numerous tasks, accurate segmentation methods are often time-consuming and thereby limit the application domains. On the other hand, numerous methods have been developed with the aim to generate superpixels efficiently with low under-segmentation errors. Superpixel algorithms have received much attention since their naming in [33]. Previous methods such as normalized cuts [34], mean shift [35],
turbopixels [36] and [37] tackle the problem as a graph partition, mode-seeking or level-set based optimization which have high computational cost. Superpixel using quick shift [38] and watershed [39] are faster but do not exhibit good boundary adherence. Significant progress has been made in recently years to improve the segmentation quality and speed. Here we review the state-of-the-art superpixel algorithms that are either based on image partition or region merging.

1) Image Partition: These algorithms start from an initial coarse partition of an image, typically with a regular grid and then refine the segments iteratively. The SLIC [9] method is an adaptation of the k-means clustering scheme for superpixel generation. It limits the search space of each cluster center and results in a significant speed-up over the conventional k-means clustering scheme. The objective function of POISE [15] is also similar to k-means clustering but is optimized in a coarse-to-fine framework. It also employs a priority queue structure for pixel assignment to avoid the post processing for connectivity check in SLIC. The LSC [11] method maps pixel values and coordinates into a high dimensional feature space and proves that optimizing the cost function of normalized cuts [34] can be by iteratively applying simple k-means clustering in the proposed feature space. The SEEDS [10] scheme is based on an objective function that can be maximized by a simple hill-climbing optimization process efficiently. Instead of computing distance from centers, this method directly exchanges pixels between neighboring superpixels. Recently, Shen et al. [16] proposed a fast superpixel method based on the DBSCAN [40] clustering. Methods in these category often start from a regular sampling and have compactness constraints that favor equal-sized or regular-shaped segments. However, the generated superpixels do not adhere to image boundaries well, especially for fine-structured objects as they are not well modeled by regular sampling (see Figure 6 for an example).

2) Region Merging: These algorithms operate on growing regions into segments. The ERS [6] method uses entropy rate of a random walk on a graph as a criteria to generate high-quality superpixels. Humayun et al. [14] proposed a superpixel merging algorithm which can generate a small set of seeds that reliably cover a large number of objects of all sizes. Though these methods have good segmentation accuracies, their computational costs are too high for real-time applications. Felzenszwalb and Huttenlocher (FH) [5] proposed a fast graph-based method in which pixels are vertices and edge weights measure the dissimilarity between vertices. Similar to other region merging methods [41, [42], it uses the Kruskal algorithm [43] to construct a minimum spanning forest in which each tree is a segment. Each vertex is initially placed in its own component, and the FH method merges regions by a criterion that the resulting segmentation is neither too coarse nor too fine. The FH method adaptively adjusts segmentation criterion based on the degree of variability in neighboring regions of the image, such that it obeys certain global properties even though greedy decisions are made. However, the under-segmentation error is high as shown by the recent studies [9], [10]. In contrast, the proposed algorithm dynamically adjusts the weights of the graph by aggregating the attributes of clusters during segmentation. We show that this feature aggregation scheme outperforms the state-of-the-art methods in all evaluation metrics.

III. SUPERPIXEL HIERARCHY

Let \( G = (V, E) \) denote an undirected graph consisted of \( n \) vertices \( v \in V \) and \( m \) edges \( e \in E \subseteq V \times V \) with cardinalities \( n = |V| \) and \( m = |E| \). Each pixel is associated with a vertex and locally connected to its 4 neighbors. Each edge \( e_{ij} = (v_i, v_j) \) is assigned a weight (typically non-negative real value) that measures the dissimilarity between the two vertices. In the superpixel segmentation task, let \( k \) denote the number of superpixels to be extracted, a segmentation \( S \) of a graph \( G \) is a partition of \( V \) into \( k \) disjoint components and each component \( C \in S \) corresponds to a connected subgraph \( G' = (V', E') \), where \( V' \subseteq V \) and \( E' \subseteq E \).

Our algorithm belongs to a class of region merging methods [5], [42]. Different from methods based on the Kruskal algorithm [43] to merge regions, we grow regions based on the the Borůvka method [43]. The advantages are three-fold:

- The Borůvka algorithm has a linear time solution [44] and is parallelizable.
- Neighborhood information can be incorporated into one unified framework. After each iteration, the features are aggregated within the newly formed clusters. This scheme is more robust than methods such as SLIC and LSC that only use per-pixel features to determine its affiliation of the cluster.
- A hierarchy is constructed from which any amount of superpixels can be generated on the fly.

Figure 2 shows on example on superpixels at all scales are generated by the proposed SH algorithm. We first review the Borůvka algorithm, and then address the efficiency and accuracy issues by edge contraction and feature aggregation.

A. Extracting Superpixels via the Borůvka Algorithm

The Borůvka algorithm computes a minimum spanning tree (MST) in a bottom-up manner. Consider a graph as a forest with \( n \) trees, namely one vertex itself is a tree. For each tree, we find its nearest neighbor which is connected by the lightest edge and join them together.

Let \( C_2 \) denote the nearest neighbor of \( C_1 \) (\( C_1 \) may not be the nearest neighbor of \( C_2 \)). We define the distance between two trees as

\[
D(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} \omega((v_i, v_j)).
\]

After the nearest neighbor search, an auxiliary graph is built where each vertex represents a cluster and each edge corresponds to one chosen light edge. If there are mutual nearest neighbors, we use duplicated edges to represent them in the auxiliary graph; on the other hand, edges in the auxiliary graph are distinctive. Then we use deep-first search to find the connected components. The Borůvka algorithm repeats merging trees in this manner until only one tree is left. The major difference between Borůvka and Kruskal algorithms is that the former searches for edges locally and simultaneously while the later sorts the edges globally and
executes sequentially. As such, the Borůvka algorithm can be processed in parallel. In addition, the Borůvka algorithm assumes that clusters are uniformly distributed and alleviates the drawback of the Kruskal algorithm that tends to generate heavily unbalanced clusters [43]. In the proposed superpixel hierarchy method, we adapt the Borůvka algorithm to construct a MST. Meanwhile, the order that each edge is added to the MST is recorded. Once an edge is added to the MST, the number of trees in the forest is reduced by one. Suppose that \( k \) superpixels need to be extracted, we connect vertices by the first \( n - k \) edges and have \( k \) connected components which are the superpixels exactly.

### B. Linear Time Algorithm via Edge Contraction

In this section, we re-formulate the Borůvka algorithm with edge contraction. Instead of maintaining a forest of trees, we contract each tree to a single vertex. This reduces the number of vertices and edges substantially and facilitate achieving significant speed-up.

An edge contraction is illustrated in Figure 3. At each vertex of Figure 3(a), a number denotes the attributes (e.g., pixel intensity). Each edge weight is computed by the absolute distance of the attributes at two ends. An edge contraction is performed between vertex 4 and 2. After contracting the edge, vertex 4 and 2 become a supervertex, resulting in a self-loop and two parallel edges (shown in red edges in Figure 3(b)). A flattening operation is followed (Figure 3(c)) by removing the self-loop and replacing parallel edges with the lightest one.

In the following, we explain the details of our implementation with complexity analysis. We denote the graph at the beginning of the \( i \)-th iteration by \( G_i \) and the number of vertices and edges of this graph by \( n_i \) and \( m_i \), respectively.

#### Lemma 1: The SH algorithm stops in \( O(\log n) \) merging iterations.

**Proof:** Each tree is merged with at least one of its neighbors, and the number of trees in \( G_i \) decreases by at least a factor of two. Thus, the SH algorithm stops in \( O(\log n) \) iterations.

#### Lemma 2: Each merging iteration of SH algorithm runs in \( O(m_i) \) time.

**Proof:** First, the nearest neighbor search for each vertex loops through all edges to determine the lightest edge for the vertex on either endpoint, which takes \( O(m_i) \) time. Next, the histogram sorting [45] process of the \( n_i \) chosen edges (one for each vertex) takes \( O(n_i) \) time. In addition, tree growing uses an auxiliary graph whose vertices are the labels of the original trees and edges correspond to the chosen lightest edges. The auxiliary graph has \( n_i \) vertices and \( n_i \) edges (where the edges may be duplicated). We find the connected components of this graph using depth-first search, which takes \( O(n_i) \) time [45].

The edge contraction process is carried out by histogram sorting edges lexicographically and then removing loops and parallel ones, which takes \( O(m_i) \) time. Thus, each iteration of the SH algorithm takes \( O(m_i + n_i) = O(m_i) \) time.

#### Theorem 1: The SH algorithm takes \( O(n) \) to operate on a planar graph.

**Proof:** When the input is a planar graph, every \( G_i \) is planar as the class of planar graphs is closed under edge contraction [43]. Furthermore, \( G_i \) is also simple (loops and parallel edges have already been removed) such that we can use Euler’s formula on the number of edges of a planar simple graph to obtain \( m_i \leq 3n_i \). From Lemma 1, we know that
Methods are efficient because they search for limited regions which is sensitive to outliers. Second, both SH and SLIC incorporate neighborhood information during segmentation. In contrast, the SLIC method operates on the pixel level and local algorithmic design. Recall that (1) measures the accuracy. This is our advantage over methods such as SLIC and LSC that connectivity must be enforced at the end such that the edge confidence between two regions can be determined directly as shown in Figure 3(e). Thus our method can directly combine an edge detector [46] and absorb rapid progress [47], [48] in this area to improve segmentation accuracy. This is our advantage over methods such as SLIC and LSC that connectivity must be enforced at the end such that it is unclear how to integrate edge information into such procedures efficiently.

By incorporating edge confidence, our distance measure becomes

\[ D(C_1, C_2) = d_c \times d_e, \]  

where \( d_c \) and \( d_e \) are color and edge distance, respectively. The color distance is measured by the absolute difference of mean color. However, the mean color is not sufficient to represent superpixels as they become larger at different scales. For better performance, we measure color difference by the \( \chi^2 \) distance of color histograms after \( j \) iterations (see Section IV for parameter settings). We note that the SEEDS method also adopts a color histogram based distance metric. The difference is that SEEDS uses intersection distance while our method adopts the \( \chi^2 \) distance. The edge distance \( d_e \) is measured by the average edge confidence between the two regions (i.e., the average edge confidence in the red curve in Figure 3(e)).

**IV. EXPERIMENTS**

We present the experimental results of the proposed algorithm against the state-of-the-art methods in terms of segmentation accuracy and run-time. The source code will be made available to the public for accessible reproducible research.

**A. Experimental Setup**

1) **Datasets and Evaluated Methods:** We use the Berkeley Segmentation Dataset (BSDS500) [1], segmentation challenge of Pascal 2012 Visual Object Classes (SegVOC12) [49], Berkeley Semantic Boundaries Dataset (SBD) [50], and Microsoft Common Objects in Context (COCO14) [51] for performance evaluation. We use the BSDS500 dataset for thorough segmentation evaluation as the images contain accurate annotated segments and boundaries. On the other hand, we use the other datasets for object segmentation evaluation. Although these images do not contain accurate boundaries, the adopted achievable segmentation accuracy (ASA) metric provides the upper bound performance when using superpixels as units for object segmentation.
We evaluate the proposed SH algorithm against the FH [5], SLIC [9], ERS [6], SEEDS [10], and LSC [11] methods using the original implementations. The FH and SLIC methods are widely used in the literature due to simplicity and efficiency. The ERS and LSC schemes are considered as the state-of-the-art in terms of accuracy but with high computational load, and the SEEDS approach is the most efficient one among these methods. Evaluation against other methods such as normalized cuts [34] and turbopixels [36] are not presented here as the accuracy or efficiency are far from state-of-the-art methods [21]. We evaluate two versions of proposed scheme: SH with color features only, and SH_E with both color and edge features.

2) Parameter Settings: In this work, the color difference is measured by the $\chi^2$ distance of histograms (equally divided into $k$ bins) after $j$ iterations. We use the structured forest edges (SFE) [46] to compute the edge feature. These parameters are set based on a training database and fixed as $(j, k) = (4, 20)$ in our experiments. For other methods, the default parameters are used for fair comparisons.

B. Evaluation Metrics

We use the widely used metrics [21] to evaluate superpixel segmentation methods including achievable segmentation accuracy, under-segmentation error and boundary recall.

The achievable segmentation accuracy (ASA) measures the fraction of ground truth segment that is correctly labeled by superpixels,

$$ASA(S) = \frac{\sum_k \max_i |s_k \cap g_i|}{\sum_i |g_i|},$$

where $g_i$ is a ground truth segment, $s_k$ is a superpixel and $|\cdot|$ indicates the size of the segment.

The under-segmentation error (UE) measures the extent superpixels cover the ground truth segment border

$$UE(S) = \sum_i \sum_k \min(|s_k \cap g_i|, |s_k - g_i|) \frac{1}{\sum_i |g_i|}.$$ 

The boundary recall (BR) measures the percentage of ground truth edges that fall within superpixel boundaries within a margin $\varepsilon = 2$ pixels. Given a ground truth boundary union sets $B(g)$ and the superpixel boundary sets $B(s)$, the boundary recall of a segmentation $S$ is defined by

$$BR(S) = \frac{TP(S)}{TP(S) + FN(S)},$$

where $TP(S)$ is the number of boundary pixels in $B(g)$ that fall within a boundary pixel $B(s)$ in the range $\varepsilon$, and $FN(S)$ is the opposite case.

C. Segmentation Accuracy

Figure 4(a)-(c) show the quantitative evaluation results on the BSDS500 dataset using the three metrics. The proposed SH and SH_E algorithms (and) perform well in all three metrics. The SH algorithm outperforms the LSC (---) and ERS (----) methods while it generates superpixels of all scales simultaneously and efficiently. With the assistance of the SFE method [46], the SH_E algorithm performs significantly well. Figure 5 shows sample segmentation results on the BSDS500 test set by the proposed SH algorithm.
The results are similar on other three object datasets, as shown in Figure 4(e)-(g). On the SegVOC12 dataset, the SH method performs well against the LSC and ERS schemes in terms of ASA. The SLIC approach (→) does not perform well due to its regular sampling strategy (see Figure 6). The LSC scheme starts with regular sampling but maps pixels into a high dimensional feature space. This helps LSC to capture the global image structure, but the segmentation results also depend on image quality (see Figure 7).

Table II shows the results of the SH algorithm with several edge detection methods. The segmentation accuracy of the SH algorithm can be further improved when more effective edge detectors, e.g., [47], are used. In this work, we choose the SFE scheme [46] for computational efficiency. We note that the POISE [14] http://rehg.org/poise/, parameters: \(\sigma = 5.0\) method also incorporates edge features thus we make a comparison with it. It can be seen from Table II that our method outperforms POISE on several edge detectors.

D. Computational Complexity and Run-Time

We analyze the computational complexity and run-time performance of superpixel algorithm based on the number of superpixels and image size on a machine with one 3.4 GHz i7 CPU. We do not use any parallelization (multi-core, SIMD instructions or GPU).

1) Computational Complexity: The FH method uses the Kruskal algorithm [43] to grow region that runs in \(O(n \log n)\) worst time with low constant factors. The SLIC scheme uses the \(k\)-means clustering procedure with constrained search region which runs in \(O(n \log n)\) time of each iteration but needs several iterations to convergence. The SEEDS algorithm maximizes its energy function via hill-climbing optimization at pixel and block levels. The run-time of block-level optimization depends on the number of superpixels. The ERS approach constructs a submodular and monotonic objective function that can be optimized by a lazy greedy algorithm. The worst case complexity of lazy greedy algorithms is \(O(n^2 \log n)\) while in [6] it is shown that on average the complexity is \(O(n \log n)\). The LSC procedure shares a similar framework with the SLIC method and the complexity is also \(O(n)\). However, it is apparently slower than SLIC because it operates in high dimensional feature space and requires more iterations to achieve higher segmentation accuracy. The recent DBSCAN [16] https://github.com/shenjianbing, parameters: \(a_1 = 0.6, a_2 = 0.4, a_3 = 1, \Psi = 30\) method uses a similar framework to SLIC and also has a \(O(n)\)

| SH and POISE [14] with several edge detectors on the BSDS500 Dataset |
|-------------------------|-----------------|-----------------|-----------------|
| | mean ASA ↑ | mean UB ↓ | mean BR ↑ |
| SH+SCG [51] | 95.4% | 9.1% | 78.4% |
| SH+SPE [45] | 95.5% | 8.9% | 79.8% |
| SH+HED [46] | 95.7% | 8.6% | 80.2% |
| POISE+SCG [51] | 94.3% | 10.7% | 72.6% |
| POISE+SPE [45] | 94.8% | 10.3% | 73.5% |
| POISE+HED [46] | 95.0% | 10.1% | 74.1% |
Fig. 6. SH compared to SLIC with 100 superpixels on the Pascal segVOC12, SBD, and COCO14 datasets. The SLIC method does not perform well on fine-structured objects even when the compact factor $m$ is reduced. The SH$_E$ method improves the SH algorithm by using edge features. (a) input. (b) SLIC ($m = 10$). (c) SLIC ($m = 2$). (d) SH. (e) SH$_E$.

TABLE III

| Method  | Build. | Grass | Tree | Cow | Sheep | Sky | Air | Water | Face | Cat | Bicycle | Flower | Sigs | Bird | Book | Chair | Road | Cat | Dog | Body | Boat | Global | Average |
|---------|--------|-------|------|-----|-------|-----|-----|-------|------|-----|---------|--------|------|------|------|-------|------|-----|------|------|-------|--------|
| FH [5]  | 77.9   | 92.4  | 88.1 | 75.8| 75.4  | 89.3| 48.5| 58.5  | 82.6 | 60.0| 82.2    | 45.0   | 72.6 | 15.2 | 88.0 | 45.0  | 85.0 | 46.3| 36.6 | 70.3 | 04.7  | 78.1%  | 63.8%  |
| SLIC [9]| 78.0   | 93.6  | 83.8 | 87.9| 74.4  | 92.6| 46.8| 68.6  | 84.6 | 57.7| 76.5    | 67.3   | 57.9 | 19.3 | 92.5 | 40.6  | 82.9 | 60.8| 42.9 | 57.6 | 07.5  | 79.5%  | 65.4%  |
| ERS [6] | 78.9   | 92.5  | 85.5 | 83.6| 59.1  | 95.5| 68.6| 67.9  | 86.4 | 54.7| 74.1    | 53.1   | 69.8 | 25.0 | 93.2 | 57.0  | 84.1 | 51.3| 48.6 | 52.7 | 13.8  | 79.6%  | 65.5%  |
| SEEDS [10]| 77.1  | 92.7  | 88.3 | 81.8| 71.8  | 95.3| 57.2| 70.2  | 82.5 | 53.7| 76.6    | 63.5   | 67.0 | 22.8 | 94.3 | 38.4  | 85.5 | 50.0| 48.0 | 55.8 | 10.4  | 80.3%  | 65.8%  |
| LSC [11]| 81.5   | 93.4  | 84.4 | 84.7| 78.0  | 93.4| 46.4| 73.2  | 83.7 | 52.7| 77.4    | 73.8   | 60.0 | 16.3 | 92.5 | 25.2  | 85.9 | 51.5| 48.1 | 57.8 | 11.4  | 80.5%  | 65.3%  |
| SH [7]  | 80.6   | 92.9  | 84.9 | 86.8| 70.6  | 92.1| 58.0| 69.0  | 83.2 | 59.3| 80.0    | 65.5   | 79.3 | 14.2 | 86.4 | 42.0  | 85.5 | 47.3| 52.1 | 58.3 | 10.1  | 80.4%  | 66.6%  |

Fig. 7. Semantic segmentation examples. Top: input image and segmentation with 200 superpixels. Bottom: ground truth (GT) and classification results for the method of [3] with different superpixel algorithms. (a) input/GT. (b) SH. (c) LSC [11]. (d) SLIC [9].

Complexity. As analyzed in Section III-B, the computational complexity of the proposed SH algorithm is $O(n)$. Compared to other linear time methods, the proposed algorithm has $O(1)$ complexity to generate $m$ scales of superpixels while that of other methods is $O(m)$. To reduce computational cost for other methods (e.g., FH) for generating multi-scale representations, it is possible to first generate a large number of superpixels and then merge the neighbors using a similar scheme to ours. However, this approach is more ad-hoc than our unified framework and does not facilitate parallel computation.
Table IV

<table>
<thead>
<tr>
<th>Method</th>
<th>Single</th>
<th>Multi</th>
<th>Time (ms)</th>
<th>Single</th>
<th>Multi</th>
<th>Time (ms)</th>
<th>Single</th>
<th>Multi</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FH [5]</td>
<td>0.227</td>
<td>0.187</td>
<td>1106</td>
<td>0.166</td>
<td>0.144</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLIC [9]</td>
<td>0.225</td>
<td>0.190</td>
<td>554</td>
<td>0.183</td>
<td>0.139</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERS [6]</td>
<td>0.224</td>
<td>0.189</td>
<td>4160</td>
<td>0.182</td>
<td>0.138</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td>0.217</td>
<td>0.181</td>
<td>7590</td>
<td>0.185</td>
<td>0.139</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSC [11]</td>
<td>0.224</td>
<td>0.189</td>
<td>282</td>
<td>0.179</td>
<td>0.137</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td>0.217</td>
<td>0.181</td>
<td>1720</td>
<td>0.185</td>
<td>0.139</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fh [5, 54]</td>
<td>0.227</td>
<td>0.187</td>
<td>1106</td>
<td>0.166</td>
<td>0.144</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2) Run-Time: Figure 4(d) shows the run-time with respect to the number of superpixels on the BSDS500 dataset. The run-time of the SH method is independent of superpixel numbers. The run-time of the SLIC and LSC schemes fluctuates but remains constant in general. The LSC method is 10 times slower than the SH algorithm. The run-time of the SLIC scheme varies significantly and the worst case here is twice slower. The ERS approach is 20 times slower than the SH algorithm and the run-time increases with respect to the number of superpixels. The DBSCAN method has a similar speed to the proposed SH method.

Figure 4(h) shows the run-time with respect to image size. Each set has 10 images and we report the average time. The average size of superpixels is 1024 for all image sets and algorithms. The result with the ERS algorithm is not plotted due to its high computational cost. The FH, SLIC, SEEDS, LSC, DBSCAN, and SH methods all run in time nearly linear in image size in practice.

V. APPLICATIONS

We show how the properties of the SH algorithm as summarized in Table I facilitate three image analysis tasks: semantic segmentation, saliency detection and stereo matching.

A. Semantic Segmentation

Semantic segmentation aims to assign pre-defined class labels to every pixel in an image. One effective approach for this task is to formulate this problem as an energy minimization task on a conditional random field (CRF) [3, 53]. By operating directly on the superpixels rather than pixels, the number of nodes in the CRF is significantly reduced (typically from 10^5 to 10^2 per image [53]). Thus, it requires much less time for inference [53].

Similar to [9], we use the method of [3] to evaluate superpixel algorithms on the MSRC-21 dataset [2]. As the original annotations are imprecise, we use labelings by [55] for performance evaluation. All settings of [3] are fixed for all superpixel methods. Table III shows that the SEEDS, LSC and SH methods perform well on per-class evaluation while the SH algorithm improves the per-class accuracy significantly. Figure 7 shows sample semantic segmentation results.

B. Saliency Detection

The goal of saliency detection is to determine whether a pixel belongs to the most salient object in an image. For this task, we show that a multi-scale image representation is effective for saliency detection. We use the method in [4] to detect salient regions at different scales of an image and apply the fusion method proposed in [4] to combine multiple saliency maps. We use five scales of superpixels (from 100 to 1000) for experiments on the PASCAL-S [12], ECSSD [23] and DUT-OMERON [54] datasets. As shown in Table IV and Figure 8, multi-scale segmentations can be used to effectively improve saliency detection accuracy.

C. Stereo Matching

To demonstrate the usefulness of tree structure provided by the SH algorithm, we integrate it with the non-local
aggregation method [7], [56] for stereo matching. Different from previous local stereo methods, the approach [7], [56] performs cost aggregation over the entire image with a MST in a non-local manner. The method is computationally efficient, with the complexity comparable to uniform box filtering but has edge-preserving performance. In addition, this method has also been applied to depth upsampling [56], image filtering [57], background subtraction [58] and saliency detection [59].

Similar to [20], we quantitatively evaluate the aggregation accuracy with the MST, FH, ERS methods and the SH algorithm on the Middlebury dataset. We follow the steps in [20] to build a tree structure for FH and ERS. First, image pixels are grouped into a set of subtrees (around 200 superpixels). Second, all the subtrees are linked by the rest of light edges to produce the final tree. The MST and SH already produce a whole tree structure which is used directly. All the methods use the same cost volume and do not use any post-processing. The disparity error rates in non-occlusion regions are used for evaluation. Table V shows the experimental results where the subscripts represent relative rank of the methods on each dataset. All segmentation-based structures improve the performance of the basic MST approach. The performance of the SH algorithm is higher than that of the other tree methods. It achieves the lowest average error rate on 13 (out of 19) stereo image pairs.

VI. CONCLUSIONS

In this work, we propose an effective hierarchical superpixel segmentation algorithm that can be used in a wide range of computer vision tasks. Extensive experimental results demonstrate that the proposed algorithm is accurate and efficient in generating a hierarchy of superpixels that can be applied to numerous tasks. Our future work includes speeding up the proposed method on GPUs and applying it to point cloud and video segmentation.

REFERENCES

IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 27, NO. 10, OCTOBER 2018

E. Akbas and N. Ahuja, “From ramp discontinuities to segmentation


Xing Wei received the B.S. degree in control science and engineering from Xi’an Jiaotong University, Shanxi, China, in 2013, where he is currently pursuing the Ph.D. degree with the Institute of Artificial Intelligence and Robotics. He is interested in computer vision, image processing, and machine learning. Specifically, his research interests include 3D scene understanding, large-scale visual recognition, object detection, and segmentation.

Qingxiong Yang (M’11) received the B.E. degree in electronic engineering and information science from the University of Science and Technology of China, Hefei, China, in 2004, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2010. From 2011 to 2016, he was an Assistant Professor with the Computer Science Department, City University of Hong Kong, Hong Kong. His current research interests include computer vision and computer graphics. He is a recipient of the Best Student Paper Award at MMSP in 2010 and the Best Demo Award at CVPR in 2007.

Yihong Gong (F’18) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Tokyo, Japan, in 1987, 1989, and 1992, respectively. In 1992, he joined Nanyang Technological University, Singapore, as an Assistant Professor with the School of Electrical and Electronic Engineering. From 1996 to 1998, he was a Project Scientist with the Robotics Institute, Carnegie Mellon University, USA. Since 1999, he has been with the Silicon Valley Branch, NEC Labs America, as a Group Leader, the Department Head, and the Branch Manager. In 2012, he joined Xian Jiaotong University, China, as a Distinguished Professor. His research interests include image and video analysis, multimedia database systems, and machine learning.

Narendra Ahuja (F’92) received the B.E. degree (Hons.) in electronics engineering from the Birla Institute of Technology and Science, Pilani, India, in 1972, the M.E. degree (Hons.) in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1974, and the Ph.D. degree in computer science from the University of Maryland, College Park, MD, USA, in 1979. From 1974 to 1975, he was a Scientific Officer with the Department of Electronics, Government of India, New Delhi, India. From 1975 to 1979, he was with the Computer Vision Laboratory, University of Maryland. Since 1979, he has been with the University of Illinois at Urbana-Champaign, Urbana, IL, USA, where he is currently a Donald Biggar Willett Professor with the Department of Electrical and Computer Engineering, Beckman Institute, Urbana, and the Coordinated Science Laboratory. His current research interests include extraction and representation of spatial structure in images and video, integrated use of multiple image-based sources for scene representation and recognition, versatile sensors for computer vision, and applications including visual communication, image manipulation, and information retrieval. He is a fellow of the ACM, AAAI, AAAS, IAPR, and SPIE.

Ming-Hsuan Yang (SM’06) was a Senior Research Scientist with the Honda Research Institute, where he was involved in vision problems of humanoid robots. He is currently a Professor in electrical engineering and computer science with the University of California at Merced (UC Merced), Merced. He is a Senior Member of the ACM. He received the U.S. National Science Foundation CAREER Award in 2012, the Senate Award for Distinguished Early Career Research at UC Merced in 2011, and the Google Faculty Award in 2009. He served as an Associate Editor for the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE from 2007 to 2011. He is an Associate Editor of the International Journal of Computer Vision, Image and Vision Computing, and the Journal of Artificial Intelligence Research.