RIGOR: Reusing Inference in Graph Cuts for generating Object Regions

Ahmad Humayun Fuxin Li James M. Rehg
Georgia Institute of Technology
http://cpl.cc.gatech.edu/projects/RIGOR/

Abstract

Popular figure-ground segmentation algorithms generate a pool of boundary-aligned segment proposals that can be used in subsequent object recognition engines. These algorithms can recover most image objects with high accuracy, but are usually computationally intensive since many graph cuts are computed with different enumerations of segment seeds. In this paper we propose an algorithm, RIGOR, for efficiently generating a pool of overlapping segment proposals in images. By precomputing a graph which can be used for parametric min-cuts over different seeds, we speed up the generation of the segment pool. In addition, we have made design choices that avoid extensive computations without losing performance. In particular, we demonstrate that the segmentation performance of our algorithm is slightly better than the state-of-the-art on the PASCAL VOC dataset, while being an order of magnitude faster.

1. Introduction

In recent years, figure-ground object segmentation has re-ignited interest in using unsupervised segmentation in an object recognition pipeline. Without supervision, systems such as [9, 6, 19, 34, 31] are able to generate a pool of overlapping segment proposals, where most foreground objects are captured by at least *one* segment from the pool. In the challenging PASCAL VOC benchmark, a spatial coverage of about 80% has been achieved using hundreds of segments per image. Because of the figure-ground setting, the resulting segments are often *holistic*, in that they may contain significant internal texture and edges. This is in stark contrast to superpixels, which tend to be homogeneous in appearance. The segment proposal pool generated by these methods is usually smaller than the number of hypotheses considered in conventional bounding box-based methods, and are more informative because they delineate object boundaries. Recognition pipelines based on these segment proposals have significantly advanced the state-of-the-art in both object detection and semantic segmentation [27, 1, 11].

However, segmentation-heavy approaches are generally computationally intensive. When enumerating bounding boxes and computing their features, tricks such as integral image and distance transform [10, 23] can help to speed-up the computation, even achieving real-time performance in certain tasks [3, 29]. From this perspective, segmentation-heavy approaches lose in the first stage, since state-of-the-art methods usually need from half a minute up to several minutes to compute segment proposals for a single image. This makes segmentation-based methods impractical for large-scale or real-time applications, such as video analysis [24], object tracking or robotics.

In this paper, we devise a method to generate segment proposals in a computationally efficient manner. Our approach is based on two main contributions: (1) a precomputation step which reuses computation for multiple related parametric min-cuts (§3), and (2) a set of design choices to avoid lengthy computations to produce a segment pool (§4).

State-of-the-art figure-ground segmentation methods perform parametric min-cut on multiple *seed graphs* created by enumerating foreground seeds at different image locations [6, 9, 19]. Each seed covers a small image region that is designated to be within the foreground object, and the parametric min-cut on the resulting seed graph finds object segments that encapsulate the seed region. Using a large set of diverse seed locations increases the chances of finding objects of interest. Fig. 1 shows that with more seeds, even small background objects can be captured. However, a large number of seeds means that many parametric min-cuts need to be solved, which slows down the algorithm.

Intuitively, computing multiple min-cuts from different seed graphs should result in redundant computations that could be cached to improve efficiency. We propose a graph precomputation scheme in which a single residual graph is built for all foreground seeds. This residual graph is then converted for warm starting the parametric min-cut for each specific seed accordingly. Such a precomputation scheme over all foreground seeds helps ameliorate the cost of min-cut when using a large number of seeds.

We have also introduced speed-ups via algorithm design. We show that segmentation performance does not suffer by using a superpixel graph and simple unary/pairwise poten-

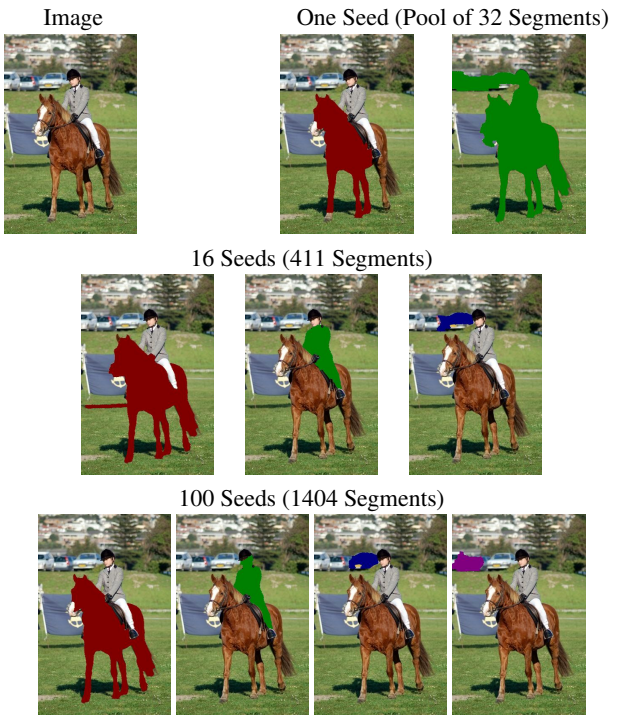


Figure 1: Effect of segment seeds (best viewed in color). With one seed at the center the algorithm is able to segment the horse. With more seeds it finds the person and gradually starts to obtain segments on the cars in the background.

tials, as opposed to the pixel graph or complex potentials used in previous methods. For computing the pairwise potential, we propose an efficient piecewise-linear regression tree that is used in a boosting framework.

We test our algorithm, RIGOR, on PASCAL VOC 2012. Experiments show that our method achieves slightly better results than previous methods, while being at least an order of magnitude faster. RIGOR can compute segment hypotheses for most images within 2-4 secs, with the prospect of being real-time with future GPU implementations.

2. Related Work

Graph cuts have been a mainstream approach for image segmentation over the last two decades [16, 30]. However, the original min-cut approaches either tend to find foreground regions with a very small size [35] or attempt to solve an NP-hard problem [30, 33]. [22, 21] characterized the submodular functions that can be optimized globally with graph cut, especially the parametric max-flow formulation [21] that produces cuts with multiple unary terms at the same time. This led to multiple image segmentation approaches such as CPMC [6], object proposals [9] and contour completion [26] that produce objects of different sizes by enumerating multiple unary parameters related to object size. Later, shape priors were introduced to improve the segmentation of objects of known shape [19, 34].

The resulting pool of multiple figure-ground segment proposals has been applied successfully in semantic segmentation [27, 1] as training/testing examples, leading to top scores in the PASCAL VOC segmentation challenge. It has also been used to generate additional features for bounding box detectors [11] and in video segmentation [24].

Due to the popularity of min-cuts for vision tasks, the idea of reusing max-flow computation of one graph for another similar graph is not new. Boykov and Jolly [4] proposed a scheme to reparameterize a graph when only a small amount of unary capacities change. In their application, these changes were induced by user scribbles during interactive segmentation. Kohli and Torr [20] identified how max-flow could be reused even when edge capacities change between graphs. Equipped with these reparameterizations, they showed an efficient way to compute object-background segmentation in videos, where the capacities in the graph change slowly from one frame to the next.

Unlike these two methods, we explore the idea of reusing max-flow computation among graphs where *the pairwise capacities remain the same, but unary capacities change drastically*, which is the relevant case for generating a pool of segments for an image. Here, the pairwise costs do not change because they depend on constant edge potentials. On the other hand, unaries change significantly because each different seed adds high capacity t-edges to different parts of the graph. Furthermore, when using color unary terms [6], unary capacities can change for all other nodes because a separate color prior is adopted for each seed.

In addition, the graph reparameterization methods of [4, 20] are iterative approaches, where the max-flow for a graph needs to be computed before it can be utilized for another graph. In contrast, our method first runs a relatively light precomputation stage, which takes all seed regions into account. Once complete, max-flow for all seed graphs can be computed in parallel, while obtaining increased efficiency due to the precomputation stage.

Verma and Batra [32] quantitatively tested different max-flow algorithms in vision applications. Our choice of building on the Boykov-Kolmogorov algorithm (BK) [5] was partially motivated by their results, since BK tends perform equally or better than push-relabel [15] and Pseudo-flow (PF) [17] in small density graphs like the ones we use.

3. Reusing Graph Cut Computations

Our segmentation algorithm, like other methods [9, 6], uses graph min-cuts from multiple seeds to compute segments. For each seed i, a directed seed graph $\mathcal{G}_i = \langle \mathcal{V}, \mathcal{E} \rangle$ is created, which has a set of nodes, $\mathcal{V}$ and edges, $\mathcal{E}$. The energy function minimized by min-cut is of the form:

$$E^i_\lambda(\mathbf{X}) = \sum_{u \in \mathcal{V}} D^i_\lambda(x_u) + \sum_{(u,v) \in \mathcal{E}} V_{uv}(x_u, x_v)\,, \qquad (1)$$

Figure 2: The use of superpixel edges in multiple min-cuts. Given parametric min-cuts from all seed graphs, the color of each edge indicates how many times it was included in a cut (edges separating fg from bg). White indicates an edge that was never used for a cut. A saturated red means the edge was included in many cuts.

where $\mathbf{X} = \{x_u\}_{u=1}^{|\mathcal{V}|}$ is the set of binary labels $\{0, 1\}$ for each superpixel u. In order to minimize this energy function, min-cut employs two special terminal nodes: the source $\mathbf{s}$ and the sink $\mathbf{t}$. The graph edges connecting nodes to $\mathbf{s}$ or $\mathbf{t}$ are known as t-edges, whose weights represent the unary potentials D^i_λ, and all others are n-edges, whose weights represent the pairwise V_{uv}. Min-cut produces two disjoint sets S and T, where node $u \in S$ iff $x_u = 1$, and $u \in T$ iff $x_u = 0$. The cut is defined by the sum of edge weights from S to T, which can be verified to equal (1).

Distinct segments are produced by enumerating foreground seeds at different locations of an image. Each node in our formulation represents a superpixel in the image, as described in §4.1. The set $\mathcal{E}$ consists of all superpixel pairs which share a boundary. The term, V_{uv}, is the pairwise potential (see §4.1) between a pair of superpixels in $\mathcal{E}$. V_{uv} and $\mathcal{E}$ do not change with the seed. Each seed i comprises of a set of superpixels $\mathbf{S}_i$ (note, $\mathbf{S}_i \subset \mathcal{V}$), such that any superpixel $x_u \in \mathbf{S}_i$ has a unary potential $D^i_\lambda(x_u) = \infty$ if $x_u = 0$. In other words, all superpixels belonging to foreground seed i have an infinite cost if assigned to the background label 0. For the remaining superpixels, the unaries are set parametrically, $D^i_\lambda(x_u) = f_i(x_u) + \lambda$, where different $f_i(\cdot)$ define uniform and color graph types. Increasing the parameter λ returns larger segments.

For each foreground seed i, unary costs are computed (Sec. 4) and the resulting energy function is minimized by obtaining the MAP solution via graph min-cut. As mentioned in the introduction, we seek to reuse computations across all seed graphs. This is possible in our case because all pairwise potentials are guaranteed to be the same regardless of the choice of seed. If we visualize parametric min-cuts for all seed graphs, one notices that some $(u, v) \in \mathcal{E}$ never belong to the cut set, i.e. u and v always belong to the same segment. Fig. 2 shows an example, where each edge in the graph is colored by the frequency with which it appears in a cut set. An edge is unlikely to be in any cut set if it aligns with a weak boundary, regardless of the choice of seed. Our experiments show that around 45% of edges are never in the cut. We have developed a computationally efficient scheme to capture such information, so it can be recycled for computing min-cuts on different seed graphs.

We utilize the reparametrization technique described in [4, 20], which demonstrated that if only unaries change, one can reparameterize a graph so that the min-cut does not change and only the flow value changes. This is desirable since we are mostly interested in the cut (segment) and not the flow value itself. Our reuse scheme first generates a *precomputation graph* which consists of one tree for each foreground seed plus a sink tree, as an extension to the source and sink trees used in the Boykov-Kolmogorov algorithm (BK) [5] (summarized in §3.1).

Given this precomputation graph, the final two stages run independently, in parallel, for each seed graph. The second stage reparametrizes the precomputation graph into a graph suitable for the current seed (§3.2.2). In the final stage, the trees in the precomputation graph are transformed to compute multiple parametric min-cuts (§3.2.3). Since these processes only involve reparameterizations, the cuts for each seed graph $\mathcal{G}_i$ do not change.

3.1. Boykov-Kolmogorov Max-flow

Augmenting path algorithms find max-flow by discovering unsaturated paths from $\mathbf{s}$ to $\mathbf{t}$. Flow is pushed/*augmented* along these paths, until no more augmenting paths remain [12]. A path p between nodes $u, v \in \mathcal{V}$ is denoted by $u \rightsquigarrow v$. At termination, only nodes which have an unsaturated path $p \equiv \mathbf{s} \rightsquigarrow u$, are in S. All other nodes belong to T.

Typical augmenting path algorithms use breadth-first search (BFS) trees to find shortest augmenting paths in a graph. Unlike algorithms like blocking flow [7] which performs BFS afresh for exploring every new depth level, BK maintains two search trees, $\mathcal{S}$ (originating from $\mathbf{s}$) and $\mathcal{T}$ (from $\mathbf{t}$). If $u, v \in \mathcal{S}$ and u is the parent of v, then (u, v) is unsaturated. In tree $\mathcal{T}$, the opposite holds true i.e. if v is the child of u, (v, u) is unsaturated. An augmenting path can be found through a path in these trees because they consist only of unsaturated edges. The algorithm iterates over three steps until no unsaturated paths from $\mathbf{s}$ to $\mathbf{t}$ exist:

I. Growth Stage: In this stage, both $\mathcal{S}$ and $\mathcal{T}$ trees greedily acquire nodes which do not belong to any tree. When we find an unsaturated edge (u, v), where $u \in \mathcal{S}$ and $v \in \mathcal{T}$, we advance to the next stage. The algorithm terminates if no such edges are found after completely growing both trees.

II. Augmentation Stage: Flow is augmented over the path found in the previous stage. This saturates one or more

edges (u, v) in the path, which disconnects u or v from $\mathcal{S}$ or $\mathcal{T}$ respectively, forming its own tree. At this point, the disconnected node is called an *orphan*.

III. Adoption Stage: Disconnected orphans are adopted back by their parent tree through an unsaturated edge, or set *free* if no such edge exists. In the latter case, the children of the free node are marked as orphans. Note, all free nodes can be later taken by $\mathcal{S}$ or $\mathcal{T}$ during the growth stage.

3.2. Graph Cuts Over Multiple Seed Enumerations

In order to reuse the $\mathcal{S}, \mathcal{T}$ trees from BK, [20] runs an adoption like stage from BK to reflect the change to a new graph. Motivated by [20], we propose a precomputation step which builds a tree from $\mathbf{s}$ for each foreground seed $\mathbf{S}_i$. These trees are then transformed to solve the min-cut on each seed graph. Reusing these multiple source trees is possible due to the structure of our problem.

To illustrate our technique, let us consider two foreground seeds $\mathbf{S}_1$ and $\mathbf{S}_2$, which induce seed graphs $\mathcal{G}_1$ and $\mathcal{G}_2$. One could run BK separately for each of these graphs, producing trees $\mathcal{S}_1, \mathcal{T}_1$ and $\mathcal{S}_2, \mathcal{T}_2$ respectively. Suppose a node $s_1 \in \mathbf{S}_1$ has an augmenting path $s_1 \rightsquigarrow t$ in graph $\mathcal{G}_1$. Similarly, $s_2 \in \mathbf{S}_2$ has an augmenting path in $\mathcal{G}_2$, $s_2 \rightsquigarrow t$. If there exists a node u such that $u \rightsquigarrow t$ is common to both augmenting paths ($s_1/s_2 \rightsquigarrow u \rightsquigarrow t$), then the flow can be recycled from one graph to the other. Conceptually, these paths might be discovered by finding the common sections in trees $\mathcal{S}_1$ and $\mathcal{S}_2$ (and similarly in $\mathcal{T}_1$ and $\mathcal{T}_2$). But finding such common paths can be hard in itself, since BK trees are not unique, and, even if they were, finding common paths will add unnecessary complexity to the problem.

3.2.1 Precomputation Graph: Instead of building trees in isolated seed graphs, we first create a precomputation graph $\mathcal{G}^p$, which contains N seed trees $\mathcal{S}_i^p$, for each of the N seed enumerations. An additional tree $\mathcal{T}^p$ is from the sink, as before. To reiterate, this is possible in our formulation because all seed graphs only differ in their unary capacities.

To construct $\mathcal{G}^p$, we start with any seed graph, let's say $\mathcal{G}_1$, which would have infinite unary capacities from $\mathbf{s}$ to nodes in $\mathbf{S}_1$. Fig. 3a would be equivalent to $\mathcal{G}_1$ if we replace the infinite capacity $(\mathbf{s}_2, 11)$ green edge with an $(11, \mathbf{t})$ edge. Next, in this graph we reparameterize all nodes belonging to seeds $\{\mathbf{S}_i\}_{i=2}^N$. This reparameterization would result in a graph which has infinite capacity from $\mathbf{s}$ to all seeds $\{\mathbf{S}_i\}_{i=1}^N$. These seed node unaries are reparameterized in the same way as explained in [20, 4]. Initially, each source tree $\mathcal{S}_i^p$ consists only of nodes in $\mathbf{S}_i$. This graph construction is illustrated in Fig. 3a.

We now use BK to grow trees in this graph. We need to introduce some changes to BK to handle multiple trees from $\mathbf{s}$. The first critical change is in the growth stage, where free nodes would be absorbed by $\mathcal{S}_i^p$ and $\mathcal{T}^p$ trees until an augmenting path is found. Like before, we can find unsaturated edges from any tree $\mathcal{S}_i^p$ to $\mathcal{T}^p$. On the other hand, we disallow finding augmenting paths between any two seed trees $\mathcal{S}_i^p$ and $\mathcal{S}_j^p$. This is enforced because both trees originate from $\mathbf{s}$, making flow augmentation impossible over such paths. This process necessitates that each node, in any source tree, stores the index i of its tree. Moreover, since source trees need to be disjoint, no two seeds can have any common superpixels, i.e. $\mathbf{S}_i \cap \mathbf{S}_j = \emptyset$, $\forall i \neq j$.

The second difference is in the adoption stage. When a node in $\mathcal{S}_i^p$ becomes an orphan due to the augmentation stage, it can only be adopted back by $\mathcal{S}_i^p$, otherwise it is set free. The reason that we do not allow adoption by any other seed tree $\mathcal{S}_{j\neq i}^p$ is to prevent adoption by a tree which itself is rooted at an orphan. This is similar to the condition in BK, where an orphan node from $\mathcal{S}$ cannot be directly adopted by $\mathcal{T}$, and vice versa. Running BK with these changes results in the precomputation graph $\mathcal{G}^p$ with multiple seed trees $\mathcal{S}_i^p$ and $\mathcal{T}^p$ (Fig. 3b). In the figure, each seed tree has its own source $\mathbf{s}_i$ for a simpler presentation. This is equivalent to using multiple sources in flow networks.

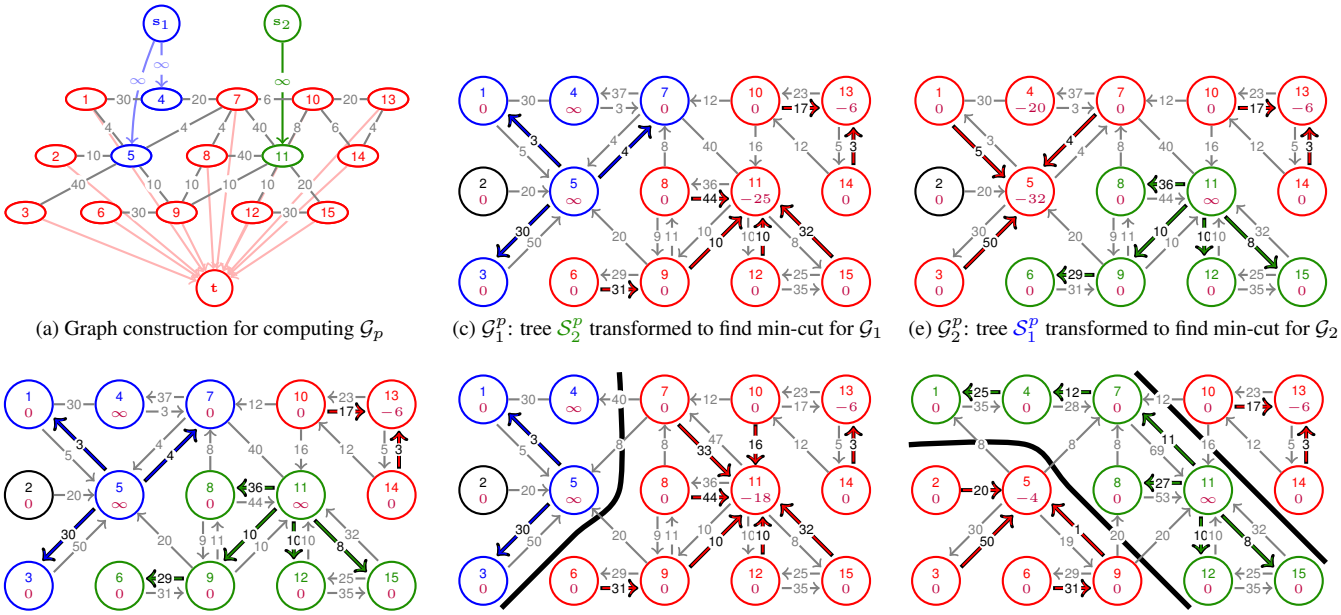
Once $\mathcal{S}_i^p$ and $\mathcal{T}^p$ have fully grown in $\mathcal{G}^p$, they can be used for computing min-cut in any seed graph $\mathcal{G}_i$. This is accomplished through the reparameterization and transformation steps detailed in the next two subsections.

Notation: On a directed edge (u, v), the capacity is c_{uv}; the flow is f_{uv}; and the residual capacity is $r_{uv} = c_{uv} - f_{uv} + f_{vu}$. When $r_{uv} > 0$, the edge is unsaturated, i.e. more flow can be pushed through it. The unary capacity c_u is defined as $c_u = c_{\mathbf{s}u} - c_{u\mathbf{t}}$, the unary flow as $f_u = f_{\mathbf{s}u} - f_{u\mathbf{t}}$, while the unary residual capacity is $r_u = c_u - f_u$.

3.2.2 Reparameterizations: Boykov and Jolly [4] demonstrated that if the capacity difference, $c_u = c_{\mathbf{s}u} - c_{u\mathbf{t}}$, remains unchanged in a new graph, the min-cut would be the same as in the original graph, and only the flow value would change. We use this idea to reparameterize all nodes whose unary capacities change from $\mathcal{G}^p$ to $\mathcal{G}_i$, resulting in a graph $\mathcal{G}_i^p$. For each node where $c_u^p \neq c_u^i$ (the superscript denotes that the quantity c_u^p is from $\mathcal{G}^p$, c_u^i is from $\mathcal{G}_i$, the same convention will be used throughout the paragraph), we would like to make sure that the flow at u remains the same as in the precomputation graph, while the unaries change from c_u^p to c_u^i. In order to achieve this, we increase the capacities of both t-edges by the same amount, so that $c_u^i = c_{\mathbf{s}u}^i - c_{u\mathbf{t}}^i$, $c_{\mathbf{s}u}^i \geq f_{\mathbf{s}u}^p$ and $c_{u\mathbf{t}}^i \geq f_{u\mathbf{t}}^p$ (there is enough capacity to hold flow in the precomputation graph). After this, the new residual capacity r_u^i is computed as:

$$r_u^i \;=\; r_u^p + (c_u^i - c_u^p) \;=\; c_u^i - f_u^p \,. \tag{2}$$

Note when $r_u^i > 0$, then $(\mathbf{s}, u)$ is unsaturated, and if $r_u^i < 0$, then $(u, \mathbf{t})$ is unsaturated.



(a) Graph construction for computing $\mathcal{G}_p$

(c) $\mathcal{G}_1^p$: tree $\mathcal{S}_2^p$ transformed to find min-cut for $\mathcal{G}_1$

(e) $\mathcal{G}_2^p$: tree $\mathcal{S}_1^p$ transformed to find min-cut for $\mathcal{G}_2$

(b) $\mathcal{G}_p$ at the end of precomputation stage, containing trees $\mathcal{S}_1^p$, $\mathcal{S}_2^p$, and $\mathcal{T}^p$ fully grown

(d) Min-cut for $\mathcal{G}_1$, computed from Fig. 3c

(f) Min-cut for $\mathcal{G}_2$, computed from Fig. 3e

Figure 3: This illustrates min-cut for $\mathcal{G}_1$ and $\mathcal{G}_2$, using the precomputation graph $\mathcal{G}_p$, composed of trees $\mathcal{S}_1^p$, $\mathcal{S}_2^p$ and $\mathcal{T}^p$. The value within each node is the unary residual capacity r_u. The value on each n-edge is its residual capacity r_{uv}. Gray edges, without an arrow, stand-in for bi-directional n-edges. **(a)** shows the precomputation graph where $c_{\mathbf{s}4} = c_{\mathbf{s}5} = c_{\mathbf{s}11} = \infty$, and other nodes are connected to the sink $\mathbf{t}$. The color of the node indicates which tree it belongs to. **(b)** shows the result once the precomputation graph $\mathcal{G}_p$ is built with $\mathcal{S}_1^p$, $\mathcal{S}_2^p$ and $\mathcal{T}^p$. Unsaturated tree edges are given in the color of the tree. Node 2 is free because it cannot be grown into $\mathcal{S}_1^p$. **(c)** shows the transformation of $\mathcal{G}_p$ into graph $\mathcal{G}_1^p$, which is valid for finding a cut for $\mathcal{G}_1$. **(d)** shows the resulting cut after running BK on the graph in (c). Similarly, **(e)** shows the transformed graph $\mathcal{G}_2^p$. **(f)** shows the final cut. Notice that nodes 10, 13, 14 remain in the sink all the time.

As an example, let us reparameterize node 5 from $\mathcal{G}^p$ (Fig. 3b) to $\mathcal{G}_2^p$ (Fig. 3e). The capacity of node 5 in the original $\mathcal{G}_2$ was $c_5^2 = -1$ (not visible in the figure). When building $\mathcal{G}^p$, a flow of $f_5^p = 31$ was pushed from $\mathbf{s}$ to node 5 (all the gray links from node 5 in Fig. 3a minus the residual capacities in the blue arrows in Fig. 3b). Hence, we know that the residual capacity is $r_5^2 = -32$ for the reparameterized $\mathcal{G}_2^p$ (see Fig. 3e).

3.2.3 Transforming the Precomputation Graph: The reparameterized $\mathcal{G}_i^p$ will contain all $N+1$ trees from $\mathcal{G}^p$, which now need to be transformed to a single source and sink tree. Let us first consider uniform unaries, where $f_i(\cdot)$ is constant, which means unary capacities only change at seeds $u \in \mathbf{S}_{j \neq i}$. In this case, to find the min-cut for $\mathcal{G}_1$, for instance, the precomputation tree $\mathcal{S}_1^p$ can be directly used as $\mathcal{S}_1$, and $\mathcal{T}^p$ can be used as $\mathcal{T}_1$. All other trees $\mathcal{S}_{i\neq 1}^p$, need to be converted to either $\mathcal{S}_1$, $\mathcal{T}_1$ or set free.

These trees are converted as follows: after reparameterization, if $r_u^i > 0$, i.e. u has a residual capacity from $\mathbf{s}$, the tree $\mathcal{S}_j^p$ rooted at node u can directly become a part of $\mathcal{S}_1$. On the other hand, if $r_u^i < 0$, $\mathcal{S}_j^p$ will need to be converted to become part of the sink tree $\mathcal{T}_1$. The key idea here is to use the existing tree structure $\mathcal{S}_j^p$ rooted at u, by searching for unsaturated edges in the reverse direction. Starting at u, we already know that all its children v have unsaturated edges (u, v). What we need to find is whether (v, u) is also an unsaturated edge. If so, node v is added to $\mathcal{T}_1$ using this edge. For instance in Fig. 3b, since all reverse edges for $\mathcal{S}_2^p$, rooted at 11, were also unsaturated, the whole tree is converted to $\mathcal{T}_1$, as shown in Fig. 3c. The chance of finding an unsaturated edge in the reverse direction is high, because the graph has the same capacity n-edges in both directions. If any child node is not reachable by an unsaturated edge, it is set free along with its children.

With non-uniform unaries, whole trees cannot be transformed directly, because non-seed nodes can also change between seed graphs. We run down every tree, $\mathcal{S}_i^p$ and $\mathcal{T}^p$, from root and check each node to see whether it has residual capacity. If it does, then we verify if it belongs to tree $\mathcal{S}_i$ or $\mathcal{T}_i$, according to the sign of its residual. Any node can be broken off from its previous parent in $\mathcal{S}_i^p$ or $\mathcal{T}^p$, as needed. Since this transformation process visits each tree node once, its computational complexity is $O(|\mathcal{V}|)$. One can see two benefits of the precomputation graph: (1) reusing tree structure is faster than building new trees, as also demonstrated by [20]; and (2) some nodes are never revisited after precomputation (nodes 10, 13, 14 in Fig. 3).

4. Implementation Details

4.1. The Superpixel Graph

We adopted the superpixel graph from [9], which is created by computing the watershed image from soft edge potentials before non-maximum suppression [2]. This gives a superpixel graph of roughly $500 - 3000$ superpixels per image and we found it to be excellent in terms of boundary recall ($> 97\%$ across the PASCAL VOC dataset). Computing min-cuts on the superpixels is significantly faster than the pixel-level graph used in [6] and loses little accuracy.

Foreground seeds are sampled from a regularly spaced grid. For the uniform unary potential, we set $f_i(x_u) = 0$ so that the potential is only dependent on λ. For the color potential, we fix a Gaussian using the colors within the seed region and compute the Bhattacharya distance from this color prior to all the superpixels as the unary.

Pairwise potentials between superpixels are learned using LAD-boosted regression trees (Sec. 4.2). The features are computed from the distribution of edge potentials along a superpixel boundary. For each superpixel boundary, the edge potentials from all pixel boundaries are collected and the percentiles (10th, 20th, ..., 100th percentile in our experiments) of the set are used as features, along with the mean edge potential and number of pixels along the edge. This reflects a much simpler potential than the one in [18] which utilized complex geometric context features. We have tried to add simple appearance difference features (e.g. [26]) but they did not help. We hypothesize that the appearance information has already been well-encoded in the edge detection algorithm. The learning of the pairwise potential is done on the training set of the BSDS-500 boundary detection dataset. One training example is generated for each superpixel pair in each image. Then all boundaries between all pairs of superpixels are compared against the human-annotated ground truth in BSDS, and the percentage of ground truth matches along each superpixel boundary is taken as the regression output. pairs within an image.

Instead of LAD regression, one could use a simpler alternative, such as the average of all edge potentials along the superpixel boundary to represent the same metric (percentage of edge matches). However, empirically we found that the alternative was not robust enough, since along a superpixel boundary there is often a significant percentage of missed edge detections or false positives, which can negatively bias the superpixel boundary potential. The learned edge potentials perform significantly better than this naive choice, especially for certain boundary detection algorithms, see Sec. 5.1.

4.2. LAD-Boosting on Regression Trees

We use a customized piecewise-linear regression tree that trains a linear regressor via least squares at each split, and decides the split based on the sum of the L_2 errors of both splits. In order to speed-up the training and avoid evaluating many features, only features that have been used in previous splits are considered in the regressor. At the root, this means that only a single dimension will be used to train each regressor. Then, suppose the tree splits using the 3^{rd} dimension, its child nodes could use both the 3^{rd} dimension and another one to train regressors and locate splits. Since we add only one dimension at each split, we were able to derive an analytic formula for splitting. Thus the training is almost as fast as a traditional decision tree with piecewise-constant splits.

Piecewise linear decision trees have been proposed before. Our main purpose is to use this decision tree in boosting in order to reduce the number of trees in a forest regressor. Previous research has shown that a boosted regressor with a few deep decision trees could achieve similar performance as one with many shallow trees while being faster to evaluate [14]. Our customized decision tree is designed to be a stronger regressor for the same purpose. For boosting a regressor, we used LAD-boosting [13] which is found empirically to be more robust than L_2 or Huber loss functions. Because of the strong tree regressor, we are able to obtain excellent results with only $5 - 50$ trees of depth $5 - 7$, as compared with the several hundreds to thousands of trees in conventional boosting and random forests.

5. Experiments

The main experiments were conducted on the validation set of the PASCAL VOC 2012 segmentation challenge. Additional experimental results can be found on the project website.[1] There are 1,449 images in the VOC 2012 validation dataset with pixel-level ground truth annotations for each object. The scores we report are average object overlap of the best segment in the pool (mean best overlap):

$$Ov(\mathbf{S}, GT) = \max_i \frac{|S_i \cap GT|}{|S_i \cup GT|}$$

where segment S_i is in the segment pool $\mathbf{S} = \{S_1, \ldots, S_n\}$, GT is the ground truth object and $|S|$ denotes the number of pixels in the segment. The spatial overlap treats objects of all sizes equally, but in the PASCAL challenge, many small objects are labeled, and segmentation would not be very valuable in those cases. Therefore, as is common in other papers, we also report the mean best covering:

$$Cov(\mathbf{S}, \mathbf{GT}) = \frac{\sum_j |GT_j| Ov(\mathbf{S}, GT_j)}{\sum_j |GT_j|}$$

where $\mathbf{GT} = \{GT_1, \ldots, GT_k\}$ denotes multiple ground truth objects in the image. Covering measures the ability to extract larger segments and explain the scene as a whole.

[1] http://cpl.cc.gatech.edu/projects/RIGOR/

Method		Mean Best Overlap	Mean Best Covering	Run Time (s)	Number of Segments
CPMC		70.67	82.24	34.01	624.1
Object Proposals		71.48	80.98	126.46	1544.1
Shape Sharing		67.82	82.71	410.31	1115.4
RIGOR	GB-25	68.04	79.83	4.62	808.7
	SketchTokens-25	67.33	78.94	2.75	839.1
	StructEdges-25	68.85	79.89	**2.16**	741.9
	GB-64	72.83	82.55	6.99	1490.3
	SketchTokens-64	72.62	82.05	4.84	1630.5
	StructEdges-64	73.64	82.84	4.71	1462.8
	GB-100	74.22	83.25	9.26	1781.9
	SketchTokens-100	73.78	82.61	6.80	1995.7
	StructEdges-100	**75.19**	**83.52**	6.84	1828.7

Table 1: VOC performance and timing results. Our approach is comparable to others in covering, better in overlap, and more than an order of magnitude faster. The number after GB, SketchTokens, and StructEdges indicates the amount of seeds used.

RIGOR is implemented in MATLAB with many crucial functions written in C++. All timings were generated on a 3.2GHz Intel i7-3930K 8 cores machine, using a set of 100 PASCAL VOC images. We combine our algorithm with three recent approaches for fast boundary detection: GB [25], SketchTokens [28] and StructEdges [8]. Separate pairwise potentials are learnt for each boundary detector.

5.1. Segmentation Performance

The segmentation performance on the PASCAL VOC 2012 validation set is shown in Table 1. Our algorithm is compared against CPMC [6], Object Proposals [9], and Shape Sharing [19]. All timings are for the full pipelines, which includes time taken for boundary computation. GB is used for computing CPMC boundaries. Note that CPMC results are about 1% better if the GlobalPB [2] boundaries are used, as in the public version, however, GlobalPB takes much longer to compute and therefore for the timing comparison it puts CPMC in a worse position.

It can be seen that our method has better performance if more seeds are used, and the number of segments is also larger. To illustrate the effect of the number of seeds we have plotted the overlap and covering scores, as well as the number of segments for different numbers of seeds in Fig. 4. Note that SketchTokens results have not been included since they are consistently worse than the two other boundary detectors. GB and StructEdges perform similarly in covering, but StructEdges is slightly better in overlap, potentially due to better boundary recall than GB. Interestingly, even with only 9 seeds the algorithm already has a mean best overlap of about 60% and a covering more than 75%, finding most foreground objects in the scene. This is suggestive of future real-time applications of the segmentation algorithm.

Table 2 presents results for learning the pairwise potentials. It can be seen that improvements are significant for StructEdges, which has higher recall but more false positives. For GB, the improvements are not significant since the precision is already high.

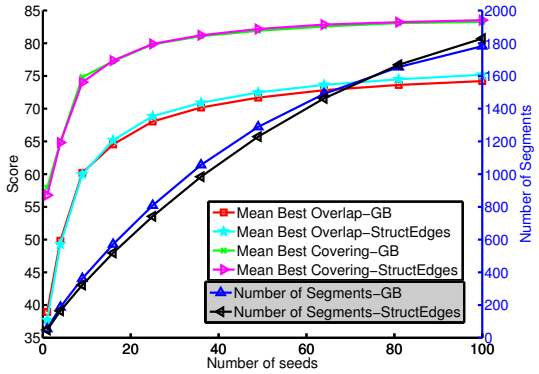


Figure 4: The number of seeds and algorithm performance.

Method	Mean Best Overlap	Mean Best Covering
StructEdges-Median	67.59	80.27
StructEdges-Mean	69.96	81.82
StructEdges-Learned	**73.64**	**82.84**
GB-Median	71.38	82.42
GB-Mean	72.67	82.39
GB-Learned	**72.83**	**82.55**

Table 2: Effects of learning pairwise potentials, on 64 seeds per image.

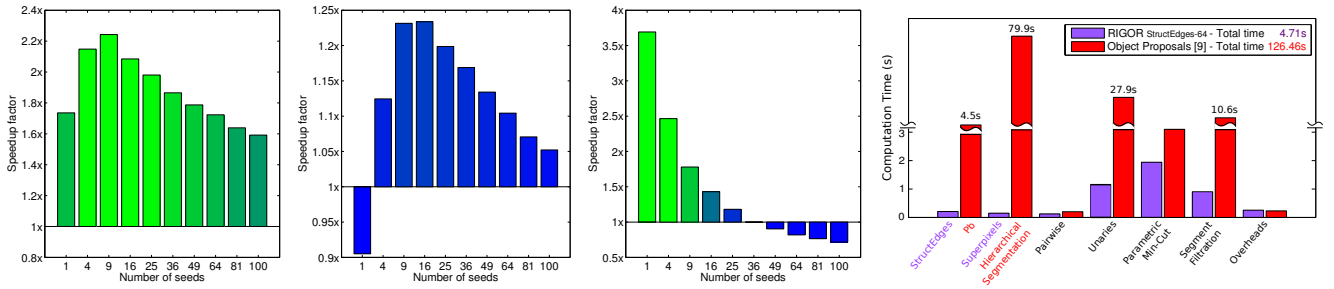
5.2. Detailed Timing Results

We quantitatively verified our use of the precomputation graph in comparison to other methods. We used the implementation provided by authors of BK [5] and PF [17]. Kohli and Torr [20]'s implementation comes packaged with BK.

We first generated precomputation graphs for each graph type (uniform and color unaries) in parallel, which are then reparameterized (§3.2.2) and transformed for each seed graph (§3.2.3), and solved in parallel. All parallelization for our method and others is done using Intel TBB. The comparative performance with different numbers of seeds is shown in Table 3 and Fig. 5. The results suggest that it is advantageous to use a precomputation graph when using 4-25 seeds. Fig. 5a-5c shows that the benefit of having a single precomputation graph tapers off when the number of seeds becomes higher. This indicates that having more than 25 trees in a single precomputation graph is less beneficial, because each tree now only contains a small set of variables. In this case, the cost of generating, reparameterizing and transforming such a graph diminishes its benefits. In future, we plan to generate multiple precomputation graphs, where each accommodates a small set of seeds with similar unaries. Parallelizing this process will harness the power of our concept further.

Time (ms)	[5]	[20]	[17]	Ours
RIGOR StructEdges-9	632.8	347.5	502.2	**282.1**
RIGOR StructEdges-25	1,363.8	825,4	813.5	**688.5**
RIGOR StructEdges-64	3,181.3	2,038.8	**1,511.8**	1,846.2

Table 3: Max-flow/Min-cut timing comparison



(a) Comparison to BK [5] (b) Comparison to Kohli & Torr [20] (c) Comparison to PF [17] (d) Pipeline timing comparison to Object Proposals [9]

Figure 5: Timing comparisons. **(a)**, **(b)**, and **(c)** give speedup factors for our parametric min-cut when compared to [5], [20], and [17], with increasing number of seeds. Here, $2\times$ means twice as fast. Note that the scale is different across graphs, but coloring is consistent; a more saturated green bar is better. **(d)** compares RIGOR StructEdges-64 timing of each component in the pipeline to its counterpart in [9].

6. Conclusion

RIGOR is the first approach to compute high-quality figure-ground segmentation proposals from a single image in just a few seconds. Our speed-up over previous methods hinges on two contributions. The first is a formulation that reuses graph computations by building a precomputation graph that considers graphs with different seeds and unary potentials simultaneously. Second, we made several design choices that avoid extensive computations. The resulting algorithm can compute a pool of segments with similar accuracy as previous methods, but an order of magnitude faster. We plan to implement our algorithm on the GPU to achieve real-time segment proposal pool generation.

Acknowledgments: This work was supported in part by NSF grants IIS 1016772 and IIS 1320348.

References

[1] P. Arbelaez, B. Hariharan, C. Gu, S. Gupta, L. Bourdev, and J. Malik. Semantic segmentation using regions and parts. In *CVPR*, 2012. 1, 2

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 33(5):898–916, 2011. 6, 7

[3] R. Benenson, M. Mathias, R. Timofte, and L. V. Gool. Pedestrian detection at 100 frames per second. In *CVPR*, 2012. 1

[4] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *CVPR*, pages 105–112 vol.1, 2001. 2, 3, 4

[5] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–1137, 2004. 2, 3, 7, 8

[6] J. Carreira and C. Sminchisescu. CPMC: Automatic Object Segmentation Using Constrained Parametric Min-Cuts. *PAMI*, 34(7):1312–1328, 2012. 1, 2, 6, 7

[7] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl*, 11(5):1277–1280, 1970. 3

[8] P. Dollar and C. Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013. 7

[9] I. Endres and D. Hoiem. Category independent object proposals. In *ECCV*, pages 575–588, 2010. 1, 2, 6, 7, 8

[10] P. F. Felzenszwalb, D. A. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008. 1

[11] S. Fidler, R. Mottaghi, A. Yuille, and R. Urtasun. Bottom-up segmentation for top-down detection. In *CVPR*, 2013. 1, 2

[12] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956. 3

[13] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001. 6

[14] R. Girshick, J. Shotton, P. Kohli, A. Criminsi, and A. Fitzgibbon. Efficient regression of general-activity human poses from depth images. In *ICCV*, 2011. 6

[15] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988. 2

[16] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B. Methodological*, 51(2):271–279, 1989. 2

[17] D. S. Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations research*, 56(4):992–1009, 2008. 2, 7, 8

[18] D. Hoiem, A. S. Stein, A. A. Efros, and M. Hebert. Recovering occlusion boundaries from a single image. In *ICCV*, 2007. 6

[19] J. Kim and K. Grauman. Shape sharing for object segmentation. In *ECCV*, 2012. 1, 2, 7

[20] P. Kohli and P. H. Torr. Dynamic graph cuts for efficient inference in markov random fields. *PAMI*, 29(12):2079–2088, 2007. 2, 3, 4, 5, 7, 8

[21] V. Kolmogorov, Y. Boykov, and C. Rother. Applications of parametric maxflow in computer vision. In *ICCV*, 2007. 2

[22] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *PAMI*, 26:147–159, 2004. 2

[23] C. Lampert, M. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, 2008. 1

[24] Y. J. Lee, J. Kim, and K. Grauman. Key-segments for video object segmentation. In *ICCV*, 2011. 1, 2

[25] M. Leordeanu, R. Sukthankar, and C. Sminchisescu. Efficient closed-form solution to generalized boundary detection. In *ECCV*, 2012. 7

[26] A. Levinshtein, C. Sminchisescu, and S. Dickinson. Optimal contour closure by superpixel grouping. In *ECCV*, pages 480–493, 2010. 2, 6

[27] F. Li, J. Carreira, and C. Sminchisescu. Object recognition as ranking holistic figure-ground hypotheses. In *CVPR*, 2010. 1, 2

[28] J. Lim, C. Zitnick, and P. Dollar. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, 2013. 7

[29] M. A. Sadeghi and D. Forsyth. Fast template evaluation with vector quantization. In *NIPS*, pages 2949–2957, 2013. 1

[30] J. Shi and J. Malik. Normalized cuts and image segmentation. *PAMI*, 22(8):888–905, 2000. 2

[31] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *IJCV*, 104:154–171, 2013. 1

[32] T. Verma and D. Batra. Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems. In *BMVC*, 2012. 2

[33] S. Wang and J. Siskind. Image segmentation with ratio cut. *PAMI*, 25(6):675–690, 2003. 2

[34] D. Weiss and B. Taskar. Scalpel: Segmentation cascasdes with localized priors and efficient learning. In *CVPR*, 2013. 1, 2

[35] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *PAMI*, 15(11):1101–1113, 1993. 2