Efficient Graph-Based Image Segmentation

Pedro F. Felzenszwalb
Artificial Intelligence Lab, Massachusetts Institute of Technology
pff@ai.mit.edu

Daniel P. Huttenlocher
Computer Science Department, Cornell University
dph@cs.cornell.edu

Abstract

This paper addresses the problem of segmenting an image into regions. We define a predicate for measuring the evidence for a boundary between two regions using a graph-based representation of the image. We then develop an efficient segmentation algorithm based on this predicate, and show that although this algorithm makes greedy decisions it produces segmentations that satisfy global properties. We apply the algorithm to image segmentation using two different kinds of local neighborhoods in constructing the graph, and illustrate the results with both real and synthetic images. The algorithm runs in time nearly linear in the number of graph edges and is also fast in practice. An important characteristic of the method is its ability to preserve detail in low-variability image regions while ignoring detail in high-variability regions.

Keywords: image segmentation, clustering, perceptual organization, graph algorithm.

1 Introduction

The problems of image segmentation and grouping remain great challenges for computer vision. Since the time of the Gestalt movement in psychology (e.g., [17]), it has been known that perceptual grouping plays a powerful role in human visual per-

ception. A wide range of computational vision problems could in principle make good use of segmented images, were such segmentations reliably and efficiently computable. For instance intermediate-level vision problems such as stereo and motion estimation require an appropriate region of support for correspondence operations. Spatially non-uniform regions of support can be identified using segmentation techniques. Higher-level problems such as recognition and image indexing can also make use of segmentation results in matching, to address problems such as figure-ground separation and recognition by parts.

Our goal is to develop computational approaches to image segmentation that are broadly useful, much in the way that other low-level techniques such as edge detection are used in a wide range of computer vision tasks. In order to achieve such broad utility, we believe it is important that a segmentation method have the following properties:

1. *Capture perceptually important groupings or regions, which often reflect global aspects of the image.* Two central issues are to provide precise characterizations of what is perceptually important, and to be able to specify what a given segmentation technique does. We believe that there should be precise definitions of the properties of a resulting segmentation, in order to better understand the method as well as to facilitate the comparison of different approaches.

2. *Be highly efficient, running in time nearly linear in the number of image pixels.* In order to be of practical use, we believe that segmentation methods should run at speeds similar to edge detection or other low-level visual processing techniques, meaning nearly linear time and with low constant factors. For example, a segmentation technique that runs at several frames per second can be used in video processing applications.

While the past few years have seen considerable progress in eigenvector-based methods of image segmentation (e.g., [14, 16]), these methods are too slow to be practical for many applications. In contrast, the method described in this paper has been used in large-scale image database applications as described in [13]. While there are other approaches to image segmentation that are highly efficient, these methods generally fail to capture perceptually important non-local properties of an image as discussed below. The segmentation technique developed here both captures certain perceptually important non-local image characteristics and is computationally

efficient – running in $O(n \log n)$ time for n image pixels and with low constant factors, and can run in practice at video rates.

As with certain classical clustering methods [15, 19], our method is based on selecting edges from a graph, where each pixel corresponds to a node in the graph, and certain neighboring pixels are connected by undirected edges. Weights on each edge measure the dissimilarity between pixels. However, unlike the classical methods, our technique adaptively adjusts the segmentation criterion based on the degree of variability in neighboring regions of the image. This results in a method that, while making greedy decisions, can be shown to obey certain non-obvious global properties. We also show that other adaptive criteria, closely related to the one developed here, result in problems that are computationally difficult (NP hard).

We now turn to a simple synthetic example illustrating some of the non-local image characteristics captured by our segmentation method. Consider the image shown in the top left of Figure 1. Most people will say that this image has three distinct regions: a rectangular-shaped intensity ramp in the left half, a constant intensity region with a hole on the right half, and a high-variability rectangular region inside the constant region. This example illustrates some perceptually important properties that we believe should be captured by a segmentation algorithm. First, widely varying intensities should not alone be judged as evidence for multiple regions. Such wide variation in intensities occurs both in the ramp on the left and in the high variability region on the right. Thus it is not adequate to assume that regions have nearly constant or slowly varying intensities.

A second perceptually important aspect of the example in Figure 1 is that the three meaningful regions cannot be obtained using purely local decision criteria. This is because the intensity difference across the boundary between the ramp and the constant region is actually smaller than many of the intensity differences within the high variability region. Thus, in order to segment such an image, some kind of adaptive or non-local criterion must be used.

The method that we introduce in Section 3.1 measures the evidence for a boundary between two regions by comparing two quantities: one based on intensity differences across the boundary, and the other based on intensity differences between neighboring pixels within each region. Intuitively, the intensity differences across the boundary of two regions are perceptually important if they are large relative to the intensity differences inside at least one of the regions. We develop a simple algorithm which

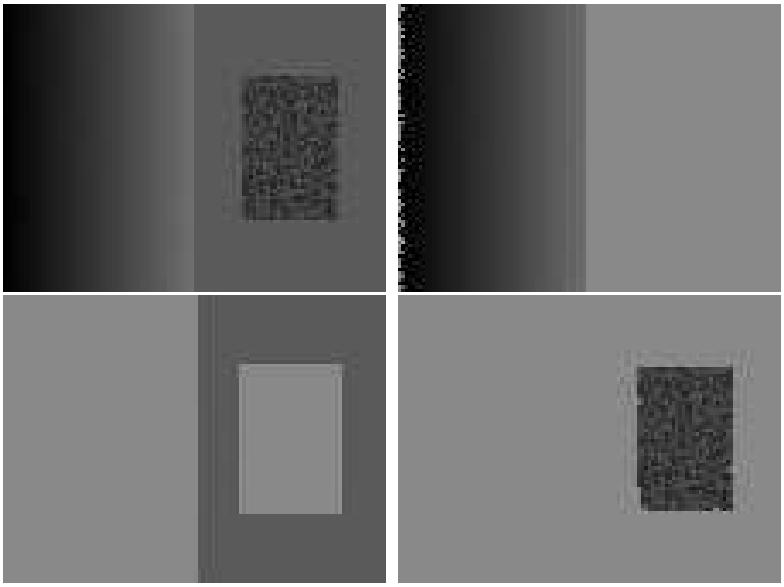

Figure 1: A synthetic image with three perceptually distinct regions, and the three largest regions found by our segmentation method (image 320×240 pixels; algorithm parameters $\sigma = 0.8$, $k = 300$, see Section 5 for an explanation of the parameters).

computes segmentations using this idea. The remaining parts of Figure 1 show the three largest regions found by our algorithm. Although this method makes greedy decisions, it produces results that capture certain global properties which are derived below and whose consequences are illustrated by the example in Figure 1. The method also runs in a small fraction of a second for the 320×240 image in the example.

The organization of this paper is as follows. In the next Section we discuss some related work, including both classical formulations of segmentation and recent graph-based methods. In Section 3 we consider a particular graph-based formulation of the segmentation problem and define a pairwise region comparison predicate. Then in Section 4 we present an algorithm for efficiently segmenting an image using this predicate, and derive some global properties that it obeys even though it is a greedy algorithm. In Section 5 we show results for a number of images using the image grid to construct a graph-based representation of the image data. Then in Section 6 we illustrate the method using more general graphs, but where the number of edges is still linear in the number of pixels. Using this latter approach yields results that capture high-level scene properties such as extracting a flower bed as a single region, while still preserving fine detail in other portions of the image. In the Appendix we show that a straightforward generalization of the region comparison predicate presented in

Section 3 makes the problem of finding a good segmentation NP-hard.

2 Related Work

There is a large literature on segmentation and clustering, dating back over 30 years, with applications in many areas other than computer vision (cf. [9]). In this section we briefly consider some of the related work that is most relevant to our approach: early graph-based methods (e.g., [15, 19]), region merging techniques (e.g., [5, 11]), techniques based on mapping image pixels to some feature space (e.g., [3, 4]) and more recent formulations in terms of graph cuts (e.g., [14, 18]) and spectral methods (e.g., [16]).

Graph-based image segmentation techniques generally represent the problem in terms of a graph $G = (V, E)$ where each node $v_i \in V$ corresponds to a pixel in the image, and the edges in E connect certain pairs of neighboring pixels. A weight is associated with each edge based on some property of the pixels that it connects, such as their image intensities. Depending on the method, there may or may not be an edge connecting each pair of vertices. The earliest graph-based methods use fixed thresholds and local measures in computing a segmentation. The work of Zahn [19] presents a segmentation method based on the minimum spanning tree (MST) of the graph. This method has been applied both to point clustering and to image segmentation. For image segmentation the edge weights in the graph are based on the differences between pixel intensities, whereas for point clustering the weights are based on distances between points.

The segmentation criterion in Zahn's method is to break MST edges with large weights. The inadequacy of simply breaking large edges, however, is illustrated by the example in Figure 1. As mentioned in the introduction, differences between pixels within the high variability region can be larger than those between the ramp and the constant region. Thus, depending on the threshold, simply breaking large weight edges would either result in the high variability region being split into multiple regions, or would merge the ramp and the constant region together. The algorithm proposed by Urquhart [15] attempts to address this shortcoming by normalizing the weight of an edge using the smallest weight incident on the vertices touching that edge. When applied to image segmentation problems, however, this is not enough to provide a reasonable adaptive segmentation criterion. For example, many pixels in

the high variability region of Figure 1 have some neighbor that is highly similar.

Another early approach to image segmentation is that of splitting and merging regions according to how well each region fits some uniformity criterion (e.g., [5, 11]). Generally these uniformity criteria obey a subset property, such that when a uniformity predicate $U(A)$ is true for some region A then $U(B)$ is also true for any $B \subset A$. Usually such criteria are aimed at finding either uniform intensity or uniform gradient regions. No region uniformity criterion that has been proposed to date could be used to correctly segment the example in Figure 1, due to the high variation region. Either this region would be split into pieces, or it would be merged with the surrounding area.

A number of approaches to segmentation are based on finding compact clusters in some feature space (cf. [3, 9]). These approaches generally assume that the image is piecewise constant, because searching for pixels that are all close together in some feature space implicitly requires that the pixels be alike (e.g., similar color). A recent technique using feature space clustering [4] first transforms the data by smoothing it in a way that preserves boundaries between regions. This smoothing operation has the overall effect of bringing points in a cluster closer together. The method then finds clusters by dilating each point with a hypersphere of some fixed radius, and finding connected components of the dilated points. This technique for finding clusters does not require all the points in a cluster to lie within any fixed distance. The technique is actually closely related to the region comparison predicate that we introduce in Section 3.1, which can be viewed as an adaptive way of selecting an appropriate dilation radius. We return to this issue in Section 6.

Finally we briefly consider a class of segmentation methods based on finding minimum cuts in a graph, where the cut criterion is designed in order to minimize the similarity between pixels that are being split. Work by Wu and Leahy [18] introduced such a cut criterion, but it was biased toward finding small components. This bias was addressed with the *normalized cut* criterion developed by Shi and Malik [14], which takes into account self-similarity of regions. These cut-based approaches to segmentation capture non-local properties of the image, in contrast with the early graph-based methods. However, they provide only a characterization of each cut rather than of the final segmentation.

The normalized cut criterion provides a significant advance over the previous work in [18], both from a theoretical and practical point of view (the resulting segmenta-

tions capture intuitively salient parts of an image). However, the normalized cut criterion also yields an NP-hard computational problem. While Shi and Malik develop approximation methods for computing the minimum normalized cut, the error in these approximations is not well understood. In practice these approximations are still fairly hard to compute, limiting the method to relatively small images or requiring computation times of several minutes. Recently Weiss [16] has shown how the eigenvector-based approximations developed by Shi and Malik relate to more standard spectral partitioning methods on graphs. However, all such methods are too slow for many practical applications.

An alternative to the graph cut approach is to look for cycles in a graph embedded in the image plane. For example in [10] the quality of each cycle is normalized in a way that is closely related to the normalized cuts approach.

3 Graph-Based Segmentation

We take a graph-based approach to segmentation. Let $G = (V, E)$ be an undirected graph with vertices $v_i \in V$, the set of elements to be segmented, and edges $(v_i, v_j) \in E$ corresponding to pairs of neighboring vertices. Each edge $(v_i, v_j) \in E$ has a corresponding weight $w((v_i, v_j))$, which is a non-negative measure of the dissimilarity between neighboring elements v_i and v_j. In the case of image segmentation, the elements in V are pixels and the weight of an edge is some measure of the dissimilarity between the two pixels connected by that edge (e.g., the difference in intensity, color, motion, location or some other local attribute). In Sections 5 and 6 we consider particular edge sets and weight functions for image segmentation. However, the formulation here is independent of these definitions.

In the graph-based approach, a segmentation S is a partition of V into components such that each component (or region) $C \in S$ corresponds to a connected component in a graph $G' = (V, E')$, where $E' \subseteq E$. In other words, any segmentation is induced by a subset of the edges in E. There are different ways to measure the quality of a segmentation but in general we want the elements in a component to be similar, and elements in different components to be dissimilar. This means that edges between two vertices in the same component should have relatively low weights, and edges between vertices in different components should have higher weights.

3.1 Pairwise Region Comparison Predicate

In this section we define a predicate, D, for evaluating whether or not there is evidence for a boundary between two components in a segmentation (two regions of an image). This predicate is based on measuring the dissimilarity between elements along the boundary of the two components relative to a measure of the dissimilarity among neighboring elements within each of the two components. The resulting predicate compares the inter-component differences to the within component differences and is thereby adaptive with respect to the local characteristics of the data.

We define the *internal difference* of a component $C \subseteq V$ to be the largest weight in the minimum spanning tree of the component, $MST(C, E)$. That is,

$$Int(C) = \max_{e \in MST(C,E)} w(e) \ . \tag{1}$$

One intuition underlying this measure is that a given component C only remains connected when edges of weight at least $Int(C)$ are considered.

We define the *difference between* two components $C_1, C_2 \subseteq V$ to be the minimum weight edge connecting the two components. That is,

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j)) \ . \tag{2}$$

If there is no edge connecting C_1 and C_2 we let $Dif(C_1, C_2) = \infty$. This measure of difference could in principle be problematic, because it reflects only the *smallest* edge weight between two components. In practice we have found that the measure works quite well in spite of this apparent limitation. Moreover, changing the definition to use the median weight, or some other quantile, in order to make it more robust to outliers, makes the problem of finding a good segmentation NP-hard, as discussed in the Appendix. Thus a small change to the segmentation criterion vastly changes the difficulty of the problem.

The region comparison predicate evaluates if there is evidence for a boundary between a pair or components by checking if the difference between the components, $Dif(C_1, C_2)$, is large relative to the internal difference within at least one of the components, $Int(C_1)$ and $Int(C_2)$. A threshold function is used to control the degree to which the difference between components must be larger than minimum internal difference. We define the pairwise comparison predicate as,

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases} \tag{3}$$

where the minimum internal difference, $MInt$, is defined as,

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)). \tag{4}$$

The threshold function τ controls the degree to which the difference between two components must be greater than their internal differences in order for there to be evidence of a boundary between them (D to be true). For small components, $Int(C)$ is not a good estimate of the local characteristics of the data. In the extreme case, when $|C| = 1$, $Int(C) = 0$. Therefore, we use a threshold function based on the size of the component,

$$\tau(C) = k/|C| \tag{5}$$

where $|C|$ denotes the size of C, and k is some constant parameter. That is, for small components we require stronger evidence for a boundary. In practice k sets a scale of observation, in that a larger k causes a preference for larger components. Note, however, that k is not a minimum component size. Smaller components are allowed when there is a sufficiently large difference between neighboring components.

Any non-negative function of a single component can be used for τ without changing the algorithmic results in Section 4. For instance, it is possible to have the segmentation method prefer components of certain shapes, by defining a τ which is large for components that do not fit some desired shape and small for ones that do. This would cause the segmentation algorithm to aggressively merge components that are not of the desired shape. Such a shape preference could be as weak as preferring components that are not long and thin (e.g., using a ratio of perimeter to area) or as strong as preferring components that match a particular shape model. Note that the result of this would not solely be components of the desired shape, however for any two neighboring components one of them would be of the desired shape.

4 The Algorithm and Its Properties

In this section we describe and analyze an algorithm for producing a segmentation using the decision criterion D introduced above. We will show that a segmentation produced by this algorithm obeys the properties of being neither *too coarse* nor *too fine*, according to the following definitions.

Definition 1 *A segmentation S is* too fine *if there is some pair of regions $C_1, C_2 \in S$ for which there is no evidence for a boundary between them.*

In order to define the complementary notion of what it means for a segmentation to be too coarse (to have too few components), we first introduce the notion of a *refinement* of a segmentation. Given two segmentations S and T of the same base set, we say that T is a refinement of S when each component of T is contained in (or equal to) some component of S. In addition, we say that T is a *proper* refinement of S when $T \neq S$. Note that if T is a proper refinement of S, then T can be obtained by splitting one or more regions of S. When T is a proper refinement of S we say that T is finer than S and that S is coarser than T.

Definition 2 *A segmentation S is* too coarse *when there exists a proper refinement of S that is not too fine.*

This captures the intuitive notion that if regions of a segmentation can be split and yield a segmentation where there is evidence for a boundary between all pairs of neighboring regions, then the initial segmentation has too few regions.

Two natural questions arise about segmentations that are neither too coarse nor too fine, namely whether or not one always exists, and if so whether or not it is unique. First we note that in general there can be more than one segmentation that is neither too coarse nor too fine, so such a segmentation is not unique. On the question of existence, there is always some segmentation that is both not too coarse and not too fine, as we now establish.

Property 1 *For any (finite) graph $G = (V, E)$ there exists some segmentation S that is neither too coarse nor too fine.*

It is easy to see why this property holds. Consider the segmentation where all the elements are in a single component. Clearly this segmentation is not too fine, because there is only one component. If the segmentation is also not too coarse we are done. Otherwise, by the definition of what it means to be too coarse there is a proper refinement that is not too fine. Pick one of those refinements and keep repeating this procedure until we obtain a segmentation that is not too coarse. The procedure can only go on for $n-1$ steps because whenever we pick a proper refinement we increase the number of components in the segmentation by at least one, and the finest segmentation we can get is the one where every element is in its own component.

We now turn to the segmentation algorithm, which is closely related to Kruskal's algorithm for constructing a minimum spanning tree of a graph (cf. [6]). It can be implemented to run in $O(m \log m)$ time, where m is the number of edges in the graph.

Algorithm 1 *Segmentation algorithm.*

The input is a graph $G = (V, E)$, with n vertices and m edges. The output is a segmentation of V into components $S = (C_1, \ldots, C_r)$.

0. Sort E into $\pi = (o_1, \ldots, o_m)$, by non-decreasing edge weight.
1. Start with a segmentation S^0, where each vertex v_i is in its own component.
2. Repeat step 3 for $q = 1, \ldots, m$.
3. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing. More formally, let C_i^{q-1} be the component of S^{q-1} containing v_i and C_j^{q-1} the component containing v_j. If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ then S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise $S^q = S^{q-1}$.
4. Return $S = S^m$.

We now establish that a segmentation S produced by Algorithm 1 obeys the global properties of being neither too fine nor too coarse when using the region comparison predicate D defined in (3). That is, although the algorithm makes only greedy decisions it produces a segmentation that satisfies these global properties. Moreover, we show that any of the possible non-decreasing weight edge orderings that could be picked in Step 0 of the algorithm produce the same segmentation.

Lemma 1 *In Step 3 of the algorithm, when considering edge o_q, if two distinct components are considered and not merged then one of these two components will be in the final segmentation. Let C_i^{q-1} and C_j^{q-1} denote the two components connected by edge $o_q = (v_i, v_j)$ when this edge is considered by the algorithm. Then either $C_i = C_i^{q-1}$ or $C_j = C_j^{q-1}$, where C_i is the component containing v_i and C_j is the component containing v_j in the final segmentation S.*

Proof. There are two cases that would result in a merge not happening. Say that it is due to $w(o_q) > Int(C_i^{q-1}) + \tau(C_i^{q-1})$. Since edges are considered in non-decreasing weight order, $w(o_k) \geq w(o_q)$, for all $k \geq q+1$. Thus no additional merges will happen to this component, i.e., $C_i = C_i^{q-1}$. The case for $w(o_q) > Int(C_j^{q-1}) + \tau(C_j^{q-1})$ is analogous. ∎

Note that Lemma 1 implies that the edge causing the merge of two components is exactly the minimum weight edge between the components. Thus the edges causing merges are exactly the edges that would be selected by Kruskal's algorithm for constructing the minimum spanning tree (MST) of each component.

Theorem 1 *The segmentation S produced by Algorithm 1 is not too fine according to Definition 1, using the region comparison predicate D defined in (3).*

Proof. By definition, in order for S to be too fine there is some pair of components for which D does not hold. There must be at least one edge between such a pair of components that was considered in Step 3 and did not cause a merge. Let $o_q = (v_i, v_j)$ be the first such edge in the ordering. In this case the algorithm decided not to merge C_i^{q-1} with C_j^{q-1} which implies $w(o_q) > MInt(C_i^{q-1}, C_j^{q-1})$. By Lemma 1 we know that either $C_i = C_i^{q-1}$ or $C_j = C_j^{q-1}$. Either way we see that $w(o_q) > MInt(C_i, C_j)$ implying D holds for C_i and C_j, which is a contradiction. ∎

Theorem 2 *The segmentation S produced by Algorithm 1 is not too coarse according to Definition 2, using the region comparison predicate D defined in (3).*

Proof. In order for S to be too coarse there must be some proper refinement, T, that is not too fine. Consider the minimum weight edge e that is internal to a component $C \in S$ but connects distinct components $A, B \in T$. Note that by the definition of refinement $A \subset C$ and $B \subset C$.

Since T is not too fine, either $w(e) > Int(A) + \tau(A)$ or $w(e) > Int(B) + \tau(B)$. Without loss of generality, say the former is true. By construction any edge connecting A to another sub-component of C has weight at least as large as $w(e)$, which is in turn larger than the maximum weight edge in $MST(A, E)$ because $w(e) > Int(A)$. Thus the algorithm, which considers edges in non-decreasing weight order, must consider all the edges in $MST(A, E)$ before considering any edge from A to other parts of C. So the algorithm must have formed A before forming C, and in forming C it must have merged A with some other sub-component of C. The weight of the edge that caused this merge must be least as large as $w(e)$. However, the algorithm would not have merged A in this case because $w(e) > Int(A) + \tau(A)$, which is a contradiction. ∎

Theorem 3 *The segmentation produced by Algorithm 1 does not depend on which non-decreasing weight order of the edges is used.*

Proof. Any ordering can be changed into another one by only swapping adjacent elements. Thus it is sufficient to show that swapping the order of two adjacent edges of the same weight in the non-decreasing weight ordering does not change the result produced by Algorithm 1.

Let e_1 and e_2 be two edges of the same weight that are adjacent in some non-decreasing weight ordering. Clearly if when the algorithm considers the first of these two edges they connect disjoint pairs of components or exactly the same pair of components, then the order in which the two are considered does not matter. The only case we need to check is when e_1 is between two components A and B and e_2 is between one of these components, say B, and some other component C.

Now we show that e_1 causes a merge when considered after e_2 exactly when it would cause a merge if considered before e_2. First, suppose that e_1 causes a merge when considered before e_2. This implies $w(e_1) \leq MInt(A, B)$. If e_2 were instead considered before e_1, either e_2 would not cause a merge and trivially e_1 would still cause a merge, or e_2 would cause a merge in which case the new component $B \cup C$ would have $Int(B \cup C) = w(e_2) = w(e_1)$. So we know $w(e_1) \leq MInt(A, B \cup C)$ which implies e_1 will still cause a merge. On the other hand, suppose that e_1 does not cause a merge when considered before e_2. This implies $w(e_1) > MInt(A, B)$. Then either $w(e_1) > Int(A) + \tau(A)$, in which case this would still be true if e_2 were considered first (because e_2 does not touch A), or $w(e_1) > Int(B) + \tau(B)$. In this second case, if e_2 were considered first it could not cause a merge since $w(e_2) = w(e_1)$ and so $w(e_2) > MInt(B, C)$. Thus when considering e_1 after e_2 we still have $w(e_1) > MInt(A, B)$ and e_1 does not cause a merge. ∎

4.1 Implementation Issues and Running Time

Our implementation maintains the segmentation S using a disjoint-set forest with union by rank and path compression (cf. [6]). The running time of the algorithm can be factored into two parts. First in Step 0, it is necessary to sort the weights into non-decreasing order. For integer weights this can be done in linear time using counting sort, and in general it can be done in $O(m \log m)$ time using any one of several sorting methods.

Steps 1-3 of the algorithm take $O(m\alpha(m))$ time, where α is the very slow-growing inverse Ackerman's function. In order to check whether two vertices are in the same component we use set-find on each vertex, and in order to merge two components we

use use set-union. Thus there are at most three disjoint-set operations per edge. The computation of $MInt$ can be done in constant time per edge if we know Int and the size of each component. Maintaining Int for a component can be done in constant time for each merge, as the maximum weight edge in the MST of a component is simply the edge causing the merge. This is because Lemma 1 implies that the edge causing the merge is the minimum weight edge between the two components being merged. The size of a component after a merge is simply the sum of the sizes of the two components being merged.

5 Results for Grid Graphs

First we consider the case of monochrome (intensity) images. Color images are handled as three separate monochrome images, as discussed below. As in other graph-based approaches to image segmentation (e.g., [14, 18, 19]) we define an undirected graph $G = (V, E)$, where each image pixel p_i has a corresponding vertex $v_i \in V$. The edge set E is constructed by connecting pairs of pixels that are neighbors in an 8-connected sense (any other local neighborhood could be used). This yields a graph where $m = O(n)$, so the running time of the segmentation algorithm is $O(n \log n)$ for n image pixels. We use an edge weight function based on the absolute intensity difference between the pixels connected by an edge,

$$w((v_i, v_j)) = |I(p_i) - I(p_j)|$$

where $I(p_i)$ is the intensity of the pixel p_i. In general we use a Gaussian filter to smooth the image slightly before computing the edge weights, in order to compensate for digitization artifacts. We always use a Gaussian with $\sigma = 0.8$, which does not produce any visible change to the image but helps remove artifacts.

For color images we run the algorithm three times, once for each of the red, green and blue color planes, and then intersect these three sets of components. Specifically, we put two neighboring pixels in the same component when they appear in the same component in all three of the color plane segmentations. Alternatively one could run the algorithm just once on a graph where the edge weights measure the distance between pixels in some color space, however experimentally we obtained better results by intersecting the segmentations for each color plane in the manner just described.

There is one runtime parameter for the algorithm, which is the value of k that is used to compute the threshold function τ. Recall we use the function $\tau(C) =$

$k/|C|$ where $|C|$ is the number of elements in C. Thus k effectively sets a scale of observation, in that a larger k causes a preference for larger components. We use two different parameter settings for the examples in this section (and throughout the paper), depending on the resolution of the image and the degree to which fine detail is important in the scene. For instance, in the 128×128 images of the COIL database of objects we use $k = 150$. In the 320×240 or larger images, such as the street scene and the baseball player, we use $k = 300$.

The first image in Figure 2 shows a street scene. Note that there is considerable variation in the grassy slope leading up to the fence. It is this kind of variability that our algorithm is designed to handle (recall the high variability region in the synthetic example in Figure 1). The second image shows the segmentation, where each region is assigned a random color. The six largest components found by the algorithm are: three of the grassy areas behind the fence, the grassy slope, the van, and the roadway. The missing part of the roadway at the lower left is a visibly distinct region in the color image from which this segmentation was computed (a spot due to an imaging artifact). Note that the van is also not uniform in color, due to specular reflections, but these are diffuse enough that they are treated as internal variation and incorporated into a single region.

The first image in Figure 3 shows two baseball players (from [14]). As in the previous example, there is a grassy region with considerable variation. The uniforms of the players also have substantial variation due to folds in the cloth. The second image shows the segmentation. The six largest components found by the algorithm are: the back wall, the Mets emblem, a large grassy region (including part of the wall under the top player), each of the two players' uniforms, and a small grassy patch under the second player. The large grassy region includes part of the wall due to the relatively high variation in the region, and the fact that there is a long slow change in intensity (not strong evidence for a boundary) between the grass and the wall. This "boundary" is similar in magnitude to those within the player uniforms due to folds in the cloth.

Figure 4 shows the results of the algorithm for an image of an indoor scene, where both fine detail and larger structures are perceptually important. Note that the segmentation preserves small regions such as the name tags the people are wearing and things behind the windows, while creating single larger regions for high variability areas such as the air conditioning duct near the top of the image, the clothing and the

furniture. This image also shows that sometimes small "boundary regions" are found, for example at the edge of the jacket or shirt. Such narrow regions occur because there is a one or two pixel wide area that is halfway between the two neighboring regions in color and intensity. This is common in any segmentation method based on grid graphs. Such regions can be eliminated if desired, by removing long thin regions whose color or intensity is close to the average of neighboring regions.

Figure 5 shows three simple objects from the Columbia COIL image database. Shown for each is the largest region found by our algorithm that is not part of the black background. Note that each of these objects has a substantial intensity gradient across the face of the object, yet the regions are correctly segmented. This illustrates another situation that the algorithm was designed to handle, slow changes in intensity due to lighting.

6 Results for Nearest Neighbor Graphs

One common approach to image segmentation is based on mapping each pixel to a point in some feature space, and then finding clusters of similar points (e.g., [3, 4, 9]). In this section we investigate using the graph-based segmentation algorithm from Section 4 in order to find such clusters of similar points. In this case, the graph $G = (V, E)$ has a vertex corresponding to each feature point (each pixel) and there is an edge (v_i, v_j) connecting pairs of feature points v_i and v_j that are nearby in the feature space, rather than using neighboring pixels in the image grid. There are several possible ways of determining which feature points to connect by edges. We connect each point to a fixed number of nearest neighbors. Another possibility is to use all the neighbors within some fixed distance d. In any event, it is desirable to avoid considering all $O(n^2)$ pairs of feature points.

The weight $w((v_i, v_j))$ of an edge is the distance between the two corresponding points in feature space. For the experiments shown here we map each pixel to the feature point (x, y, r, g, b), where (x, y) is the location of the pixel in the image and (r, g, b) is the color value of the pixel. We use the L_2 (Euclidean) distance between points as the edge weights, although other distance functions are possible.

The internal difference measure, $Int(C)$, has a relatively simple underlying intuition for points in feature space. It specifies the minimum radius of dilation necessary to connect the set of feature points contained in C together into a single volume in

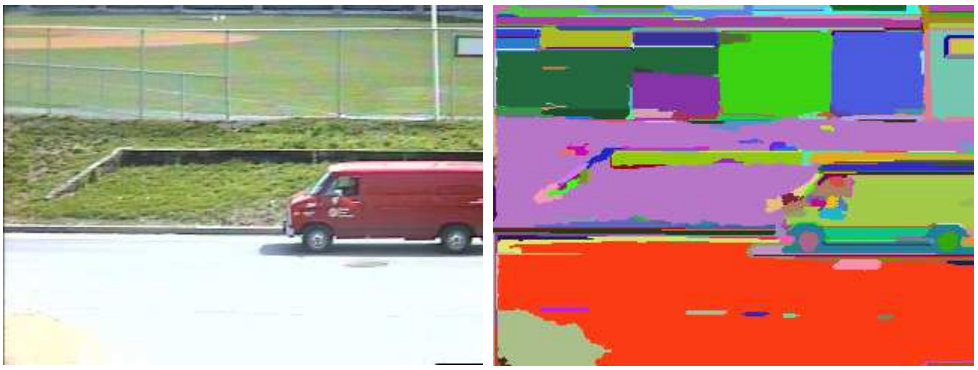

Figure 2: A street scene (320×240 color image), and the segmentation results produced by our algorithm ($\sigma = 0.8$, $k = 300$).

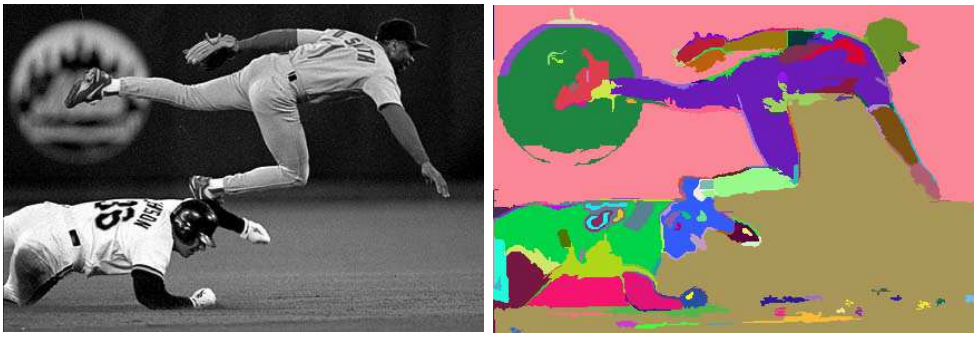

Figure 3: A baseball scene (432×294 grey image), and the segmentation results produced by our algorithm ($\sigma = 0.8$, $k = 300$).

Figure 4: An indoor scene (image 320×240, color), and the segmentation results produced by our algorithm ($\sigma = 0.8$, $k = 300$).

Figure 5: Three images from the COIL database, , and the largest non-background component found in each image (128×128 color images; algorithm parameters $\sigma = 0.8$, $k = 150$).

feature space. Consider replacing each feature point by a ball with radius r. From the definition of the MST it can be seen that the union of these balls will form one single connected volume only when $r \geq Int(C)/2$. The difference between components, $Dif(C_1, C_2)$, also has a simple underlying intuition. It specifies the minimum radius of dilation necessary to connect at least one point of C_1 to a point of C_2. Our segmentation technique is thus closely related to the work of [4], which similarly takes an approach to clustering based on dilating points in a parameter space (however they first use a novel transformation of the data that we do not perform, and then use a fixed dilation radius rather than the variable one that we use).

Rather than constructing the complete graph, where all points are neighbors of one another, we find a small fixed number of neighbors for each point. This results in a graph with $O(n)$ edges for n image pixels, and an overall running time of the segmentation method of $O(n \log n)$ time. There are many possible ways of picking a small fixed number of neighbors for each point. We use the ANN algorithm [1] to find the nearest neighbors for each point. This algorithm is quite fast in practice, given a 5-dimensional feature space with several hundred thousand points. The ANN method can also find approximate nearest neighbors, which runs more quickly than finding the actual nearest neighbors. For the examples reported here we use ten nearest neighbors of each pixel to generate the edges of the graph.

One of the key differences from the previous section, where the image grid was used to define the graph, is that the nearest neighbors in feature space capture more

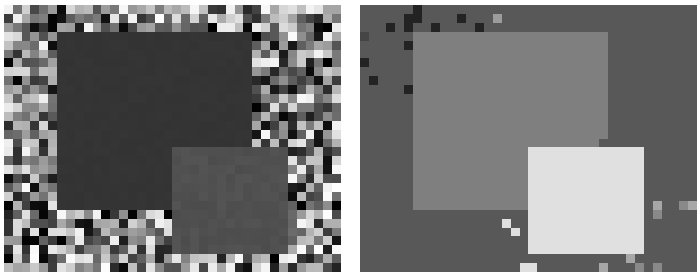

Figure 6: A synthetic image (40×32 grey image) and the segmentation using the nearest neighbor graph ($\sigma = 0$, $k = 150$).

spatially non-local properties of the image. In the grid-graph case, all of the neighbors in the graph are neighbors in the image. Here, points can be far apart in the image and still be among a handful of nearest neighbors (if their color is highly similar and intervening image pixels are of dissimilar color). For instance, this can result segmentations with regions that are disconnected in the image, which did not happen in the grid-graph case.

Figure 6 shows a synthetic image from [12] and [8] and its segmentation, using $k = 150$ and with no smoothing ($\sigma = 0$). In this example the spatially disconnected regions do not reflect interesting scene structures, but we will see examples below which do.

For the remaining examples in this section, we use $k = 300$ and $\sigma = 0.8$, as in the previous section. First, we note that the nearest neighbor graph produces similar results to the grid graph for images in which the perceptually salient regions are spatially connected. For instance, the street scene and baseball player scene considered in the previous section yield very similar segmentations using either the nearest neighbor graph or the grid graph, as can be seen by comparing the results in Figure 7 with those in Figure 2 and Figure 3.

Figure 8 shows two additional examples using the nearest neighbor graph. These results are not possible to achieve with the grid graph approach because certain interesting regions are not spatially connected. The first example shows a flower garden, where the red flowers are spatially disjoint in the foreground of the image, and then merge together in the background. Most of these flowers are merged into a single region, which would not be possible with the grid-graph method. The second example in Figure 8 shows the Eiffel tower at night. The bright yellow light forms a spatially

disconnected region. These examples show that the segmentation method, coupled with the use of a nearest neighbor graph, can capture very high level properties of images, while preserving perceptually important region boundaries.

7 Summary and Conclusions

In this paper we have introduced a new method for image segmentation based on pairwise region comparison. We have shown that the notions of a segmentation being too coarse or too fine can be defined in terms of a function which measures the evidence for a boundary between a pair of regions. Our segmentation algorithm makes simple greedy decisions, and yet produces segmentations that obey the global properties of being not too coarse and not too fine according to a particular region comparison function. The method runs in $O(m \log m)$ time for m graph edges and is also fast in practice, generally running in a fraction of a second.

The pairwise region comparison predicate we use considers the minimum weight edge between two regions in measuring the difference between them. Thus our algorithm will merge two regions even if there is a single low weight edge between them. This is not as much of a problem as it might first appear, in part because this edge weight is compared only to the minimum spanning tree edges of each component. For instance, the examples considered in Sections 5 and 6 illustrate that the method finds segmentations that capture many perceptually important aspects of complex imagery. Nonetheless, one can envision measures that require more than a single cheap connection before deciding that there is no evidence for a boundary between two regions. One natural way of addressing this issue is to use a quantile rather than the minimum edge weight. However, in this case finding a segmentation that is neither too coarse nor too fine is an NP-hard problem (as shown in the Appendix). Our algorithm is unique, in that it is both highly efficient and yet captures non-local properties of images.

We have illustrated our image segmentation algorithm with two different kinds of graphs. The first of these uses the image grid to define a local neighborhood between image pixels, and measures the difference in intensity (or color) between each pair of neighbors. The second of these maps the image pixels to points in a feature space that combines the (x, y) location and (r, g, b) color value. Edges in the graph connect points that are close together in this feature space. The algorithm yields good results

using both kinds of graphs, but the latter type of graph captures more perceptually global aspects of the image.

Image segmentation remains a challenging problem, however we are beginning to make substantial progress through the introduction of graph-based algorithms that both help refine our understanding of the problem and provide useful computational tools. The work reported here and the normalized cuts approach [14] are just a few illustrations of these recent advances.

Appendix: NP-Hardness of D with Quantiles

Intuitively the region comparison predicate D defined in Section 3.1 could be made more robust by changing the definition of the difference between two regions to reflect a quantile rather than the minimum weight edge between them. We show that with this modification the problem of finding a segmentation that is neither too coarse nor too fine becomes NP-hard.

The only difference between the new problem and the old one is the definition of the difference between two regions $C_1, C_2 \in S$ in equation (2), which becomes

$$Dif(C_1, C_2) = K_{th}\ w((v_i, v_j)) \tag{6}$$

where K_{th} selects the Kth quantile of its arguments (K should be between zero and one). For example with $K = 0.5$ the difference becomes the median edge weight between the two components. This quantile is computed over all edges (v_i, v_j) such that $v_i \in C_1$ and $v_j \in C_2$.

We reduce the min ratio cut problem with uniform capacities and demands to the problem of finding a good segmentation. The min ratio cut problem with uniform capacities and demands is the following: we are given a graph $G = (V, E)$ and another set of edges F. Each edge in E indicates a unit capacity between a pair of nodes and each edge in F indicates a unit demand between a pair of nodes. The value of a cut (A, B) is the ratio of the total capacity and the total demand between the sets A and B. So it is the ratio of the number edges in E crossing the cut and the number of edges in F crossing the cut. Finding the value of the minimum ratio cut is an NP-hard problem (cf. [2]).

First we show how to transform an instance of this problem to one where the sets E and F are disjoint, without modifying the value of the minimum cut. For every

edge $(a, b) \in E \cap F$ we create a new node ab, and exchange the edge in E with the edges (a, ab) and (b, ab). For a cut with a and b in the same side, its always better to keep ab in that side too and the value of the cut is the same as in the original graph. For a cut with a and b in different sides the node ab can be in either side and there will be one capacity and one demand edge crossing the cut and the value of cut is again the same as in the original graph.

Now we show how to decide if the modified instance of the min ratio cut problem has a cut with value at most v by solving a segmentation problem. Let c be the number of edges from E crossing a cut (A, B) and similarly d is the number of edges from F crossing (A, B). It's easy to show that the cut value is small exactly when the fraction of edges crossing the cut that come from F is large,

$$\frac{c}{d} \leq v \Leftrightarrow \frac{d}{c+d} \geq \frac{1}{v+1} \tag{7}$$

Define $G' = (V, E')$ where $E' = E \cup F$. We let the edges from E have weight zero and the edges from F have weight one.

Lemma 2 *The graph G has a cut with value at most v if and only if a segmentation of G' is not one single component, where Dif is defined in Equation 6, $K = 1-1/(v+1)$ and $\tau(C) = 0$ for all C.*

Proof. First we show that if G has a cut (A, B) with value at most v there exists $C \subseteq A$ such that the segmentation $\{C, \bar{C}\}$ is not too fine. We just need to find C such that $Int(C) = 0$ and $Dif(C, \bar{C}) = 1$. If G has a cut (A, B) with value at most v, than Equation 7 tells us that $d/(c + d) \geq 1/(v + 1)$. Remember that there are d edges of weight one and c edges of weight zero crossing the cut. So the fraction of weight one edges crossing the cut is at least $1/(v + 1)$. Look at the connected components of A using only edges of weight zero. Clearly $Int(C) = 0$ for all such components. Let C be the component with largest fraction of weight one edges going to B. This fraction must be at least $1/(v + 1)$. Moreover, the fraction of weight one edges between C and $\bar{C} = V - C$ is at least as large since $\bar{C} = B \cup (\bar{C} \cap A)$ and the there are only weight one edges between C and $\bar{C} \cap A$. This implies the fraction of weight zero edges between C and $\bar{C}$ is less than $1 - 1/(v+1) = K$. So the Kth quantile weight between C and $\bar{C}$ is one. Thus $Dif(C, \bar{C}) = 1$ and the segmentation $S = \{C, \bar{C}\}$ of G' is not too fine. Hence the segmentation of G' as a single component is too coarse.

If G' has a segmentation that is not a single component $S = \{C_1, \ldots, C_l\}$ then the Kth quantile edge weight between every pair of components C_i and C_j is one (or else

the segmentation would be too fine). Thus the Kth quantile edge weight between C_1 and $\bar{C_1} = C_2 \cup \cdots \cup C_l$ is one. So the fraction of weight one edges between C_1 and $\bar{C_1}$ is at least $1/(v+1)$. Equation 7 implies that value of the cut $(C_1, \bar{C_1})$ is at most v. ∎

It is straightforward to see that the transformation of the min ratio cut problem to the problem of finding a segmentation presented above can be done in polynomial time. This is sufficient to show the hardness of the segmentation problem.

Theorem 4 *The problem of finding a segmentation that is neither too coarse nor too fine using Dif as defined in Equation 6 is NP-hard.*

Acknowledgments

This work was supported in part by gifts from Intel, Microsoft and Xerox corporations, in part by DARPA under contract DAAL01-97-K-0104, and in part by NSF Research Infrastructure award CDA-9703470. We would like to thank Shree Nayar, Jianbo Shi and Daphna Weinshall for use of their images. We would also like to thank Jon Kleinberg, Eva Tardos and Dan Ramras for discussions about the algorithm and the NP hardness result.

References

[1] S. Arya and D. M. Mount. Approximate nearest neighbor searching. *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 271-280, 1993.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti Spaccamela and M. Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*, to appear, Springer-Verlag, Berlin.

[3] D. Comaniciu and P. Meer. Robust analysis of feature spaces: color image segmentation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 750-755, 1997.

[4] D. Comaniciu and P. Meer. Mean shift analysis and applications. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1197-1203, 1999.

[5] M.C. Cooper. The tractability of segmentation and scene analysis. *International Journal of Computer Vision*, vol 30, no 1, pages 27-42, October 1998.

[6] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, McGraw-Hill Book Company, 1990.

[7] P. Felzenszwalb and D. Huttenlocher. Image segmentation using local variation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 98-104, 1998.

[8] Y. Gdalyahu, D. Weinshall and M. Werman. Stochastic clustering by typical cuts. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2596-2601, 1999.

[9] A.K. Jain and R.C. Dubes. *Algorithms for clustering data*. Prentice Hall, 1988.

[10] I. Jermyn and H. Ishikawa. Globally Optimal Regions and Boundaries as Minimum Ratio Weight Cycles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 23, pages 1075-1088, October 2001.

[11] T. Pavlidas. *Structural Pattern Recognition*. Springer-Verlag, 1977.

[12] P. Perona and W. Freeman. A factorization approach to grouping. *Proceedings of the European Conference on Computer Vision*, pages 655-670, 1998.

[13] A.L. Ratan, O. Maron, W.E.L. Grimson and T. Lozano-Perez. A framework for learning query concepts in image classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 423-431, 1999.

[14] J. Shi and J. Malik. Normalized cuts and image segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 731-737, 1997.

[15] R. Urquhart. Graph theoretical clustering based on limited neighborhood sets. *Pattern Recognition*, vol 15:3, pages 173-187, 1982.

[16] Y. Weiss. Segmentation using Eigenvectors: A Unifying View. *Proceedings of the International Conference on Computer Vision* (2), pages 975-982, 1999.

[17] M. Wertheimer. Laws of organization in perceptual forms (partial translation). W. B. Ellis, editor, *A Sourcebook of Gestalt Psychology*, pages 71-88. Harcourt, Brace and Company, 1938.

[18] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 11, pages 1101-1113, November 1993.

[19] C.T. Zahn. Graph-theoretic methods for detecting and describing gestalt clusters. *IEEE Transactions on Computing*, vol 20, pages 68-86, 1971.

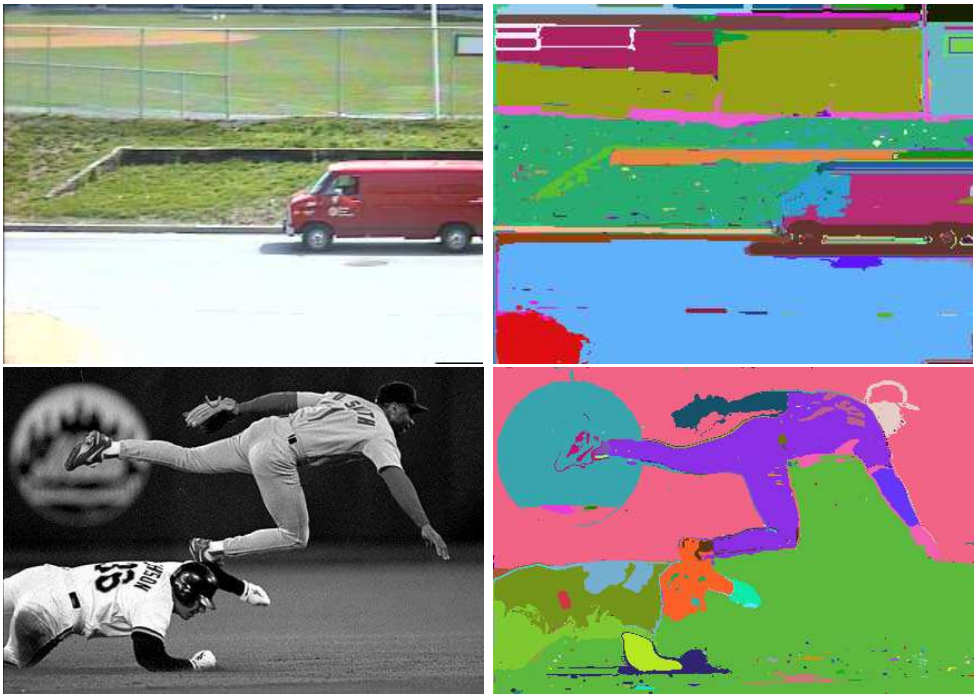

Figure 7: Segmentation of the street and baseball player scenes from the previous section, using the nearest neighbor graph rather than the grid graph ($\sigma = 0.8$, $k = 300$).

Figure 8: Segmentation using the nearest neighbor graph can capture spatially non-local regions ($\sigma = 0.8$, $k = 300$).