A Fast Segmentation Algorithm Revisited

Sharat Chandran
Computer Science and Engg. Dept.
IIT, Powai
Mumbai, INDIA 400076
sharat@iitb.ac.in

Kamlesh K. Madheshiya
Computer Science and Engg. Dept.
IT BHU
Varanasi, INDIA 221005
kamleshitbhu@rediffmail.com

Abstract

Image and, more recently, video segmentation form an important part of image understanding. Several algorithms presuppose that this step is efficiently and correctly completed by an oracle.

A recent bottom up image segmentation algorithm incorporates both local and global information in an image and runs fast in practice. This algorithm is loosely based on Kruskal's algorithm on minimum spanning trees. We build on this work and use the classical Prim's algorithm which is a theoretically superior algorithm. Our experimental result indicates that the quality of our segmentation does not suffer, yet the speed does improve. We also prove our algorithm is good in the same sense as the original work.

1. Introduction

Image segmentation is one of the best known problems in computer vision, and has been vigorously addressed for the last 30 years. Graph based methods were earlier considered to be too inefficient in practice. Recent advances in technology and algorithms ([3, 4]) have negated this assumption.

This work is directly based on the minimum spanning tree based segmentation scheme in [3]. Two properties are deemed important for producing regions:

1. The algorithm must reflect global aspects of the image.

2. The algorithm must be highly efficient.

At a higher level, our point of departure from the algorithm in [3], denoted as Algorithm K in this paper, is the second step. Our goal is to produce an algorithm that retains the advantages of Algorithm K, but runs faster. The results in Figure 1 show that our algorithm, Algorithm P2, is invariably faster (especially for larger sized images). We also prove that Algorithm P2 produces a *good* segmentation. (One of the contributions in [3] is the formal defintion of an intuitive 'good' segmentation.)

Algorithm K starts off with a connected graph representation of an image that has n vertices and $O(n)$ edges. It partitions the image (graph) into regions (components). It measures the evidence for a boundary between two regions by comparing two parametric measures: one based on intensity differences across the boundary, and the other based on intensity differences between neighbouring pixels within each region. The algorithm starts with as many regions as the vertices, and merges regions.

Our algorithm shares many important characteristics with Algorithm K (notably, use of the framework, and proof of being 'good'). The important differences are

1. Our algorithm uses a notion of a seed point and grows a region based on the seed. The seed is normally automatically chosen; however, when necessary, it supports segmenting only a part of a large image unlike Algorithm K.

2. Our algorithm uses identical parameters to those in Al-

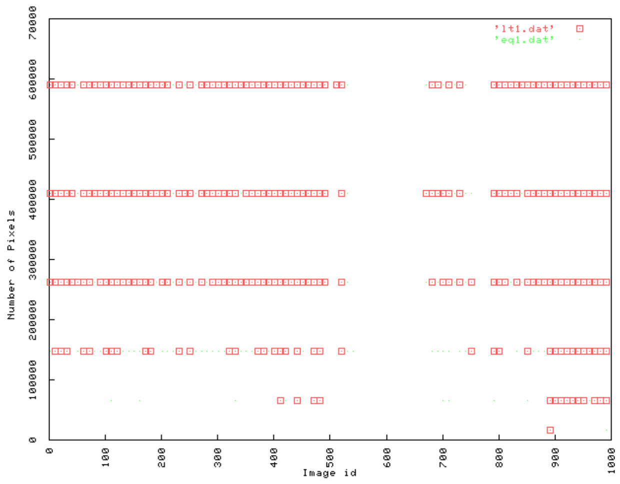


Figure 1: The ratio of time taken by Algorithm P2 to algorithm K for 100 images (with random images on the x-axis) of increasing sizes (y-axis). A box indicates that P2 is faster, a dot without a box indicates that P2 takes about the same time.

gorithm K. Since regions are grown sequentially, 'what if' analysis by varying the parameters is easier, and a segmentation can be abandoned earlier.

3. Algorithm K runs in $O(E \log E)$ time if there are E edges. In modeling non-grid graphs, the algorithm requires E to be $O(n)$ so that the overall algorithm runs in "almost" linear time. By using the Prim variation on MST and Fibonacci heaps, our algorithm has a theoretical running time of $O(E + n \log n)$ time. Therefore, there is no linearity requirement if the segmentation is to be performed in feature space.

4. Even without using Fibonacci heaps, our implementation shows a faster running time in 82 out of 100 cases in images of size 768×768.

A variation of our algorithm retains all the advantages mentioned above and is always faster. However, we cannot formally prove that the algorithm is *good*.

The rest of this paper is as follows. After a discussion of the previous work in Section 2, we give details of our method in Section 4. In Section 5 we give the results of our approach. We conclude in the final section.

2. Previous work

There is a large literature on segementation and clustering and we refer the interested reader to textbooks on the same. Our brief consideration is related to graph-based approaches.

As in the work of [4], we believe it is important that segementation algorithms take into account non-local properties of the image, because local properties fail to capture perceptually important differences in images.

Early graph based approaches do not appear to satisfy the non-local property requirement of region formations. [7] breaks large edges in a minimum spanning tree of a subgraph. Performance of [7] is based on certain heuristics and is difficult to characterize. The algorithm proposed in [5] uses a measure of local variability to decide which edges to break in partitioning of graph. The measure is based only on the nearest neighbors of each point. When applied to image segementation problems, the nearest neighbors alone are not enough to get a reasonable measure of image variability. A problem with clustering methods [2] that are based on finding compact (small radius) clusters in some intensity-based feature space is that they often [3] produce nearly constant intensity regions.

More recent graph based approach are efficient and produce good quality. The spectal partitioning technique [4], [6] operate by finding minimum cuts in the graph, where the cut criterion is designed to minimize the similarity between regions that are being split. Efficient approximation to the use of sophisticated models based on Markov Random Fields are reported in [1].

3. Algorithm K

In this section, we present only the salient parts of the algorithm in [3] as it relates to our work and skip many details.

Given a set V of elements (e.g. image pixels) to be segmented, the goal is to find a partition, or *segmentation* $S = \{C_1, C_2, \ldots, C_p\}$. We denote by $D(C_i, C_j)$ a pairwise region comparison Boolean function that judges whether or not there is evidencefor a boundary between two components C_i and C_j. S is said to be *too fine* when there is some pair of regions C_1 and C_2 for which $D(C_1, C_2)$ is false. Given two segmentations S and T of V, T is said to be a proper refinement of S when $\forall C \in T$, $\exists C' \in S$ such that $C \subset C'$. S is said to be *too coarse* when there exists a proper refinement of S that is not too fine.

A segmentation is *good* if it is neither too fine nor too coarse. There are several (different) good segmentations for the same image, and the nature of D dictates some of them.

In a weighted graph based image segmentation scheme, pixels correspond to vertices, adjacent pixels contribute edges, and the difference in image properties (such as intensity, color, motion) contribute to weights. A segmentation is obtained by discarding some of the edges, and the resulting connectivity among the vertices results in components. By defining the internal variation of a component to be the largest weight in the minimum spanning tree of the component, and the external variation to be the minimum weight edge connecting two components, and D to be true if the external variation is larger than the two internal variations, we get a *good* segmentation. The icing on the cake is that using Kruskal's algorithm a good segmentation is obtained in $n \log n$ time. (In practice, instead of D to be defined as above, a threshold τ is used to control the degree of difference).

4. Algorithm P2

The input to our algorithm is a graph $G = (V, E)$ with n vertices and m edges. The output is a segmentation that is neither too fine nor too coarse. Our algorithm uses two priority queues Q_1 and Q_2.

Inspecting algorithm P2, we observe that we essentially have two different ways to grow a component. Algorithm K worked by ordering the edges, and inserting them one by one into a component taking care never to introduce a cycle, or never to introduce a boundary edge. In contrast, the `grow()` method in Algorithm P2 builds a component starting from a source, and by adding 'leaves' one at a time to the current component. When a component cannot be grown further, a new source needs to be seleted – which is available in Q_2. The choice of the new source is central

```
overall () {
  init$Q_2$();
  for v ∈ V do {
    key[v] = $\infty$;
    insert$Q_1$(v, key[v]);
  }
  i = 0;
  while ($Q_2 \neq$ { }) {
    s = findMin ($Q_2$);
    $Q_1$.dec(s, 0);
    grow (s, i);
    i = i+1;
  }
}
```

```
init$Q_2$ () {
  for v ∈ V do {
    x = minAdjacent(v);
    insert$Q_2$ (v,x);
  }
}
```

```
grow (s, i) {
  done = false;
  $C_i$ = makeSet();
  while not done do {
    u = findMin($Q_1$);
    if (causesMerge(u, i)) {
      $C_i$ = $C_i \bigcup$ u;
      updateAdj(u);
      delete($Q_1$,u);
      delete($Q_2$,u);
    }
    else done = true;
  }
}
```

```
causesMerge(u, i) {
  if (key[u]<int($C_i$) + $\tau$)
    return TRUE;
  else return FALSE;
}
```

```
updateAdj(u) {
  for each v ∈ adj(u) {
    if (v ∈ $Q_1$ and
      w(u,v) < key[v]) {
      key[v] = w(u,v);
      $Q_1$.dec(v, key[v]);
    }
  }
}
```

Figure 2: Algorithm P2

to obtaining a good segmentation, as the following result shows.

Lemma 4.1 *The segmentation S produced by Algorithm P2 is not too fine.*

Proof: In order for S to be too fine, there is some edge between component C_i and $V - C_i$ for which D returns false. This means there were some edge e such that $w(e) < Int(C_i) + \tau$. But in this case, `causesMerge()` would have succeeded contradicting the non existence of edge e in C_i. □

Lemma 4.2 *The segmentation S produced by Algorithm P2 is not too coarse.*

Proof: In order for S to be too coarse, there must be a proper refinement T that is not too fine. Thus some component of S must be split into two or more components in T. Consider the minimum weight edge e that is internal to a component $C \in S$ but connects distinct components A and B, both $\in S$. Since T is not too fine, let $w(e) > Int(A) + \tau(A)$ without loss of generality. By construction, any edge connecting A to another subcomponent of C must have weight as large as $w(e)$ which implies that the weights of edges in A is smaller than that of e. This implies that source must have been selected from A in the method `overall()`. Hence the algorithm must have formed A before forming C. But the existence of e would then have prevented the growth of A into C which happened, contradicting the assumption that C is too coarse. □
.

The running time of the algorithm depends on how we implement the priority queue. If we use a normal binary heap (implemented as an array), it takes $O(\log n)$ time to retrieve a new node. For each incident edge we spend potentially $O(\log n)$ time decreasing the key of the neighbouring vertex. Thus the time is $O(\log n +$degree(u)$\log n)$. The computation of internal variation is done in constant time within the loop in grow(). As a result the entire algorithm takes $O((n + E) \log n)$. However, if Fibonacci heaps are used, the decrease key is done in $O(1)$ amortized time. Therefore the algorithm runs in time $O(E + n \log n)$.

4.1 Algorithm P1

As a variation, we disbanded with the second priority queue, and used only one as in the traditional Prim algorithm. When a component stops growing, the new source is set to the vertex with minimum key in the priority queue Q_1. This variation makes the proof of Lemma 4.2 invalid. Nevertheless, as we see in Section 5, the quality of segmentation is good, and the running time is excellent.

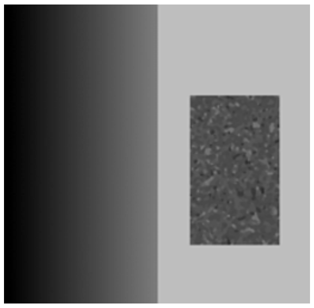
5. Results

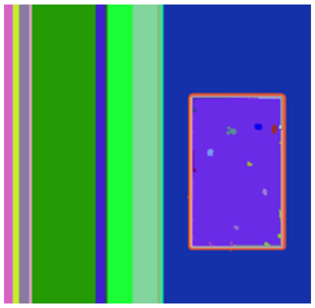
Two types of results are shown in this section. We first show evidence that the qualitative performance of Algorithm P2 and Algorithm P1 is acceptable. Later we show that the time taken by our algorithms is often less.

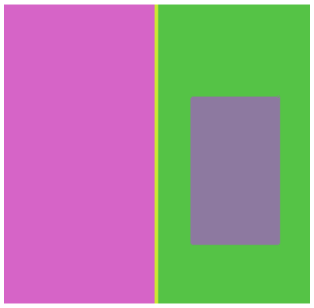
5.1 Qualitative Results

Figure 3(a) is an interesting synthetic image with three perceptually distinct regions on which Algorithm K works very well. Algorithms P2 and P1 pass this test very well as seen in Figure 3(b), Figure 3(c) and Figure 3(d) respectively.

Figure 4 and Figure 5 show the results of segmentation on other images. In all these cases, the the first image of these figures is the original image. The second, third, and fourth sub-figures show the segmented results of Algorithms K, P2, and P1 respectively. Fig. 4(a) shows the original dinosaurs image. Note that P2 (Fig. 4(c)) produces only


(a) A well known synthetic image (size 448x438).


(b) Algorithm K (4.98 seconds, k=300, σ=0.8)


(c) Algorithm P2 (7.86 seconds, k=80000, σ=2.75)

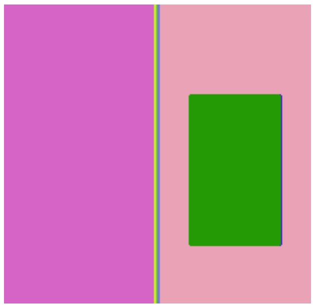
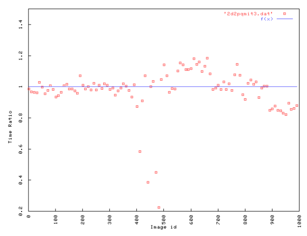

(d) Algorithm P1 (4.95 seconds, k=2000, σ=1.75)

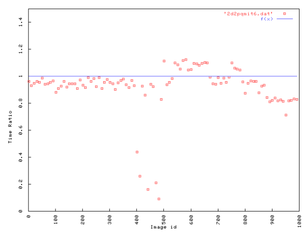
Figure 3: Segmenting a synthetic image. The goal is to obtain only 3 components.

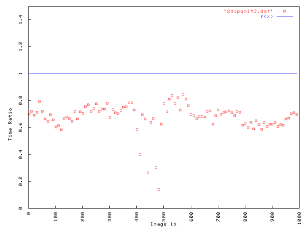
1 segment for the background. Figure 4(e) shows a plane in the sky with perceptually two distinct regions. The segmentation output (Fig. 4(f), Figure 4(g) and Figure 4(h)) are nearly the same in segmentation quality, but the proposed methods take only half as much time. Figure 4(i) shows as the original image a synthetic horse. The proposed methods (Figure 4(k) and Figure 4(l)) appear to have about the same quality as that of Algorithm K (Figure 4(j)). The running time is lower. Figure 5(a) shows an image of a boy carrying a bucket. The P2 method (Figure 5(c)) produces only one major segment for the background, whereas the other two methods (Figure 5(b) and Figure 5(d)) produce many small segments. Figure 4(q) shows a street scene. The results of all the three methods are comparable.

5.2 Running Time

Figure 6 shows the ratio of the running time of our algorithm to that of Algorithm K. The x and y axes correspond to the image ID and the ratio respectively. The points lying below the line $y = 1$ show that our proposed methods (P2 or P1) is better than algorithm K.


(a) Algorithm P2 (Image size 384x384.)


(b) Algorithm P2 (Image size 768x768.)


(c) Algorithm P1 (Image size 384x384.)

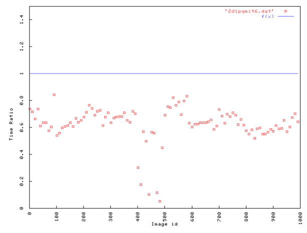

(d) Algorithm P1 (Image size 768x768.)

Figure 6: Ratio of running times. See text.

The results in Figure 7 show that our algorithm, Algorithm P1, is invariably faster (especially for larger sized images) and also produces a *good* segmentation.

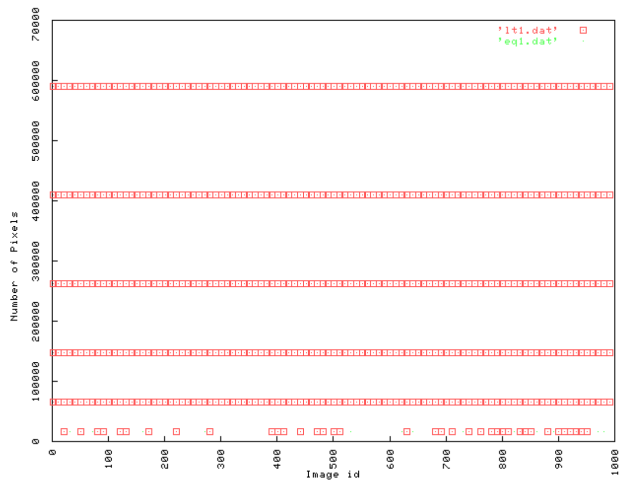
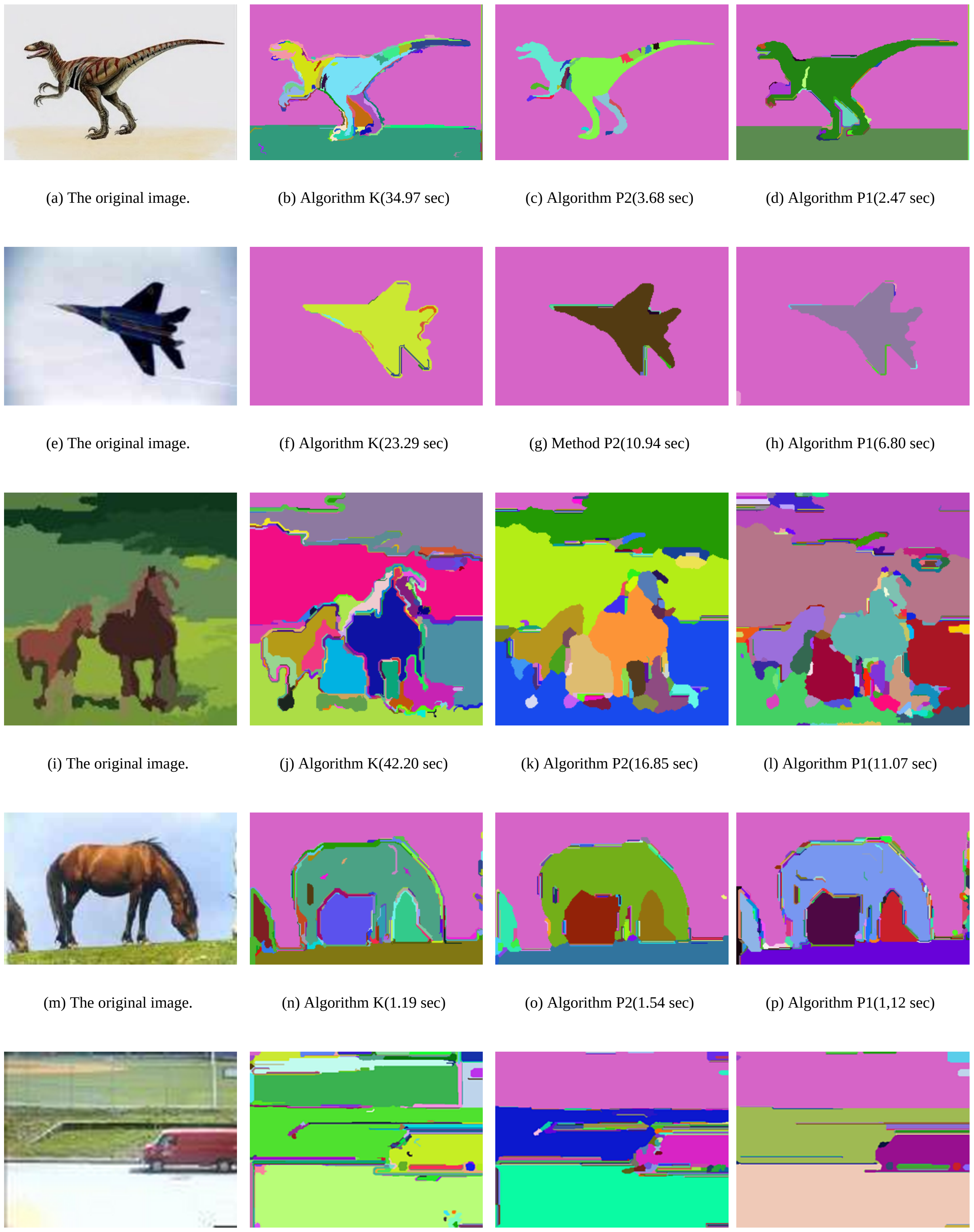


Figure 7: The ratio of time taken by Algorithm P1 to algorithm K for 100 images (with random images on the x-axis) of increasing sizes (y-axis). A box indicates that P1 is faster, a dot without a box indicates that P1 takes about the same time.



(a) The original image.
(b) Algorithm K(34.97 sec)
(c) Algorithm P2(3.68 sec)
(d) Algorithm P1(2.47 sec)

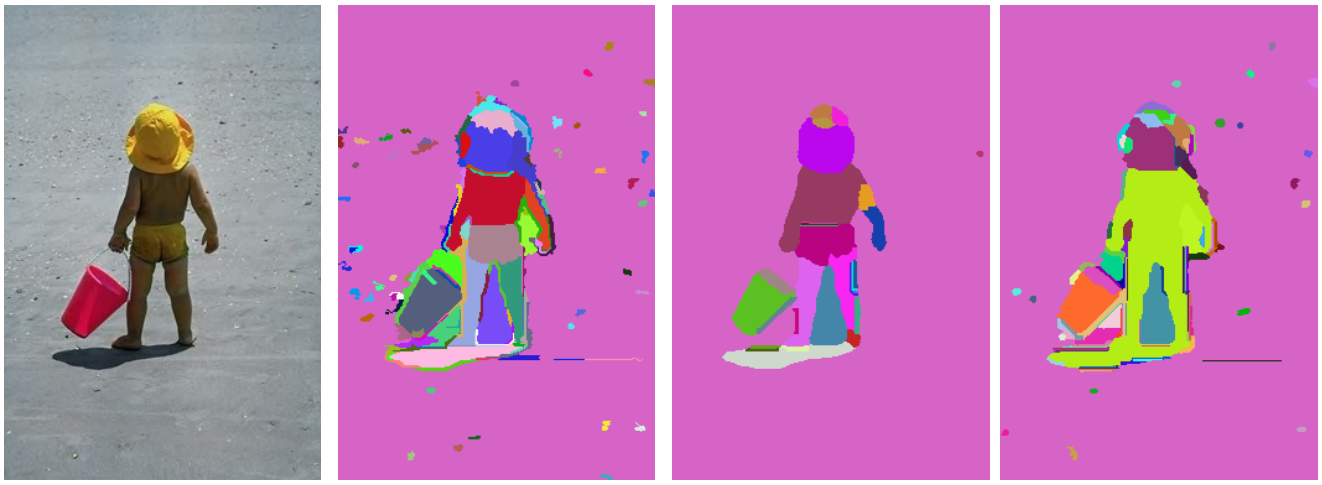
(e) The original image.
(f) Algorithm K(23.29 sec)
(g) Method P2(10.94 sec)
(h) Algorithm P1(6.80 sec)

(i) The original image.
(j) Algorithm K(42.20 sec)
(k) Algorithm P2(16.85 sec)
(l) Algorithm P1(11.07 sec)

(m) The original image.
(n) Algorithm K(1.19 sec)
(o) Algorithm P2(1.54 sec)
(p) Algorithm P1(1,12 sec)

(q) The original image.
(r) Algorithm K(5.14 sec)
(s) Algorithm P2(5.12 sec)
(t) Algorithm P1(3.61 sec)

Figure 4: Results of various segmentation algorithms best viewed in color.



(a) The original image. (b) Algorithm K(3.62 sec) (c) Algorithm P2(3.84 sec) (d) Algorithm P1(2.53 sec)

Figure 5: Results of various segmentation algorithms best viewed in color.

6. Concluding remarks

In this paper we have used the framework for image segmentation developed in [3] to come up with a faster algorithm that is also proved to be *good*.

The original result used the union-find data structure and a particular function to determine when components ought to be merged. It was also observed that tweaking this function resulted in a NP-hard problem. Given this background, we did not change the nature of function. Instead, we observe that a source based simplification is possible using the Prim variation on minimum weight spanning tree. Our implementation using two priority queues was almost always faster than the original implementation.

A weaker version of our algorithm may result in a segmentation that is too coarse. We have found this algorithm useful in content based image retrieval where the fine details might not always be relevant. This version works even faster than our dual queue version.

Acknowledgements

We thank the authors of [3] for providing the source of the original algorithm which enabled us to perform exhasutive testing. We also thank Mayur Srivastava for help with production of images.

References

[1] Y. Boykov, O. Veksler, and R. Zabih. Markov Random Fields with efficient approximations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–655, 1998.

[2] D. Comaniciu and P. Meer. Robust analysis of feature spaces: Color image segmentation, 1997.

[3] P. Felzenszwalb and D. Huttenlocher. Image segmentation using local variation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 98–104, 1998.

[4] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[5] R. Urquhart. Graph theoretical clustering based on limited neighbourhood sets. *Pattern Recognition*, 15:173–187, 1982.

[6] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clusterin:theory and its application to image segmentation. *IEEE Transactions on PAMI*, 11:1101–1113, 1993.

[7] C. Zahn. Graph-theoretic methods for detection and describing Gestalt clusters. *IEEE Transactions on Computers*, 20:68–86, 1971.