Random Forest Clustering and Application to Video Segmentation

Frank Perbet
http://frank.perbet.org
Björn Stenger
http://mi.eng.cam.ac.uk/~bdrs2
Atsuto Maki
atsuto.maki@crl.toshiba.co.uk

Toshiba Research Europe
Cambridge Research Laboratory
208 Science Park, Cambridge CB4 0GZ, UK
http://www.toshiba-europe.com/research/crl/cvg/

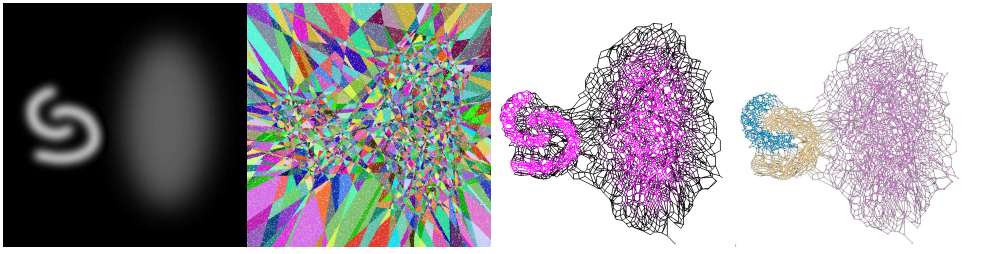
Abstract

This paper considers the problem of clustering large data sets in a high-dimensional space. Using a random forest, we first generate multiple partitions of the same input space, one per tree. The partitions from all trees are merged by intersecting them, resulting in a partition of higher resolution. A graph is then constructed by assigning a node to each region and linking adjacent nodes. This *Graph of Superimposed Partitions (GSP)* represents a remapped space of the input data where regions of high density are mapped to a larger number of nodes. Generating such a graph turns the clustering problem in the feature space into a graph clustering task which we solve with the Markov cluster algorithm (MCL). The proposed algorithm is able to capture non-convex structure while being computationally efficient, capable of dealing with large data sets. We show the clustering performance on synthetic data and apply the method to the task of video segmentation.

1 Introduction

Clustering is the task of partitioning a data set into subsets so that the data points in each subset are more similar to each other, according to some distance measure, than those from different subsets [7, 13]. It is a fundamental technique in data analysis and has many applications in computer vision, including image and video segmentation [1, 6, 15, 22]. This paper mainly considers the problem of clustering data on non-convex manifolds in high-dimensional spaces.

Many existing algorithms for clustering directly use the distances between input data in an iterative process. Given the number of clusters K as input, the K-means algorithm alternates the computation of cluster centres and cluster membership. Mean shift clustering requires a kernel bandwidth parameter and performs hill climbing in the data density space, assigning each point to the mode it converges to [4, 5, 12]. Spectral clustering is based on pairwise similarities and a variety of methods have been proposed with different definitions of affinity functions [18, 21, 23]. Recent work introduces some scale invariance by computing similarity values depending on the local neighbourhood of each point [26].



(a) Input data density (b) Random forest partitions (c) Graph structure (d) Clustering result

Figure 1: **Overview of the proposed clustering algorithm.** *(a) PDF of a 2D synthetic data set where brighter regions correspond to higher density. (b) Compact partitions (intersections of multiple partitions from decision forest). (c) The edges of the Graph of Superimposed Partitions (GSP), brighter colour corresponds to larger edge weight. (d) The nodes of the GSP coloured by the final cluster ID.*

In this paper we present a two-stage clustering algorithm that allows structure recovery of clusters of different sizes as well as non-convex shaped clusters. We first construct multiple partitions of the data using a random forest [3]. This process spreads the density of data samples evenly, similar to histogram equalization, but applied to a high dimensional space. The partitions of each tree are then merged by intersecting them, resulting in a partition of higher resolution.

We subsequently construct a graph by assigning a node to each region and connect neighbouring nodes. Graph-based clustering methods use connectedness between points, allowing to cluster non-convex and elongated structures [10, 14, 24, 25]. We employ the Markov Cluster algorithm (MCL) which is based on stochastic flow simulation [20]. The main parameter of this algorithm is the resolution which controls the size of the generated clusters. A lower resolution parameter results in fewer clusters. See Figure 1 for an example with a synthetic data set involving non-convex manifolds and varying sample densities.

The two steps of the method presented here, remapping the space and clustering the resulting graph, are independent and alternative methods could be used for each step. Our motivation to use a random forest in the first step is that it is fast to train and extremely fast to evaluate new data points. The overlapping nature of the trees can be efficiently used to link the regions without using an explicit notion of distance, which can lead to incorrect connections in the case of thin, elongated manifolds. For the second step, the Markov cluster algorithm (MCL) [20] is employed owing to its scalability to large data sets.

Related work can be found in the literature on clustering using trees. Decision trees have been used for clustering, for example by assuming the existence of a distance measure [2] or by assuming a second, uniformly distributed, background class [16]. Decision forests have also recently been applied to visual codebook construction where leaf indices from multiple trees are stacked into a single vector [17, 19]. The resulting partitions are useful for the task of codebook generation, but do not necessarily capture the underlying structure of the data. Random projections of the data have been shown to reduce the dimensionality of the data while sufficiently preserving distances. They have been applied to clustering high dimensional data in combination with EM clustering [9]. Recently, random projection trees have been used to learn the structure of manifolds, outperforming k-d trees in terms of quantisation error [11].

With the goal of maintaining the benefits of tree-based approaches, the proposed algorithm employs a random forest to create multiple partitions of the input space. The subse-

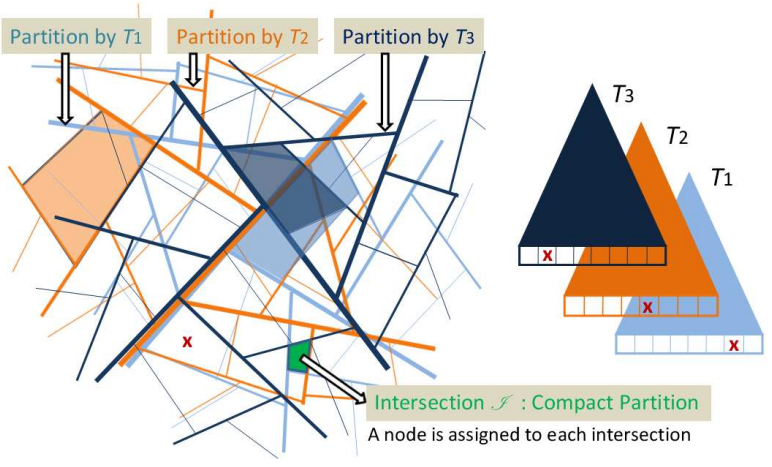


Figure 2: **Feature Space Partitioning Using a Random Forest.** *Sketch of a random forest with three trees splitting 2D input feature space into different partitions. An example of a region is filled with a colour for each partition. The identical input region in the feature space marked by 'x'(left), will have a different partition index in each tree (right). An example of intersection (compact partition) is shown in green.*

quent graph construction allows the application of efficient graph clustering methods, which in the case of video segmentation needs to handle millions of data points. The algorithm does not require the number of clusters, nor kernel radius as input, however the number of trees, the tree depth and the resolution parameter in the graph clustering step need to be determined.

2 Clustering Algorithm

Given is a set of N data points, $\{\mathbf{x}_i\}$, where $\mathbf{x}_i \in \mathbf{R}^D$ for $i = 1,\ldots,N$, our task is to assign a *cluster index*, $c \in (1,...,C)$, to every $\mathbf{x}_i$ as well as to find a meaningful number of clusters C. We would also like to acquire a function that returns a cluster index for any new query data. The strategy is to use a random forest [3] for constructing a Graph of Superimposed Partitions (GSP) that maps the high dimensional data to a lower dimensional space. We subsequently aggregate the regions represented by the GSP nodes into clusters by employing MCL.

2.1 Data Partitioning

Partition Defined by a Single Tree. We first partition the input data with a forest, i.e. an ensemble of F trees, $\{T_j\}, j = 1,\ldots,F$. Starting at the root node, the input data is recursively split at each node when a certain number of data points (typically one thousand) has reached this node. At each node, we compute a random split function for a data point $\mathbf{x}_i$ as the inner product with a vector $\mathbf{f} \in \mathbf{R}^D$. The vector $\mathbf{f}$ is a random unit vector generated using a normal distribution for each element, followed by normalisation. The resulting value $\mathbf{x}_i^T\mathbf{f}$ can also be interpreted as a projection onto a random direction. The values are histogrammed and the median taken as the split threshold in order to evenly split the input data. An initial sampling

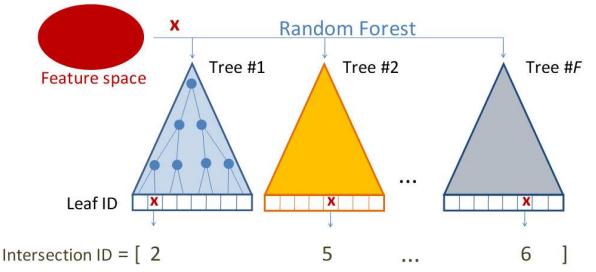


Figure 3: **Random Forest Mapping.** *Intersection index vectors have leaf indices across the trees as their elements, and represent a compact partition.*

step allows the adaptation of the histogram range. The stopping criterion is defined by the depth of the tree.

The n_j leaf nodes of tree T_j represent the regions of the induced partition. An input data point $\mathbf{x}_i$ is assigned to one of these leaves to obtain a unique index $l_j \in \{1,\dots,n_j\}$, called the *partition index* of $\mathbf{x}_i$ in the j-th tree. All data points assigned to the same leaf node of T_j belong to the same l_j-region of the input data space. We denote the partition of the feature space induced by T_j as $\mathscr{P}_j$.

Multiple Partitions Defined by a Forest. Multiple random trees result in different partitions of the input space, see Figure 2. It shows that the identical input will not necessarily have the same partition index in each tree. Thus, there is no consistency between the partition indices obtained from different trees. By concatenating the partition indices of $\mathbf{x}_i$ from all F trees, we define the *intersection index vector* as

$$\mathbf{l}(\mathbf{x}_i) = (l_1,\dots,l_F)^{\mathrm{T}} \in \mathbf{N}_+^{\mathbf{F}} \quad ,$$

where each element l_j of the vector represents the leaf index in T_j, see Figure 3. We now define the *intersections*, $\mathscr{I}(\mathbf{l})$, of the output of F trees according to the *intersection index vectors* and merge all partitions $\mathscr{P}_j, j = 1,\dots,F$ by intersection, i.e. by subdividing every region such that each $\mathscr{I}(\mathbf{l})$ consists of inputs with only a single $\mathbf{l}$. As these intersections form compact partition regions, we also refer to $\mathscr{I}(\mathbf{l})$ as *compact partitions*. For each compact region with a sufficiently large number of sample points we compute the first two moments of its data points to compute volume and density estimates.

2.2 GSP: Graph of Superimposed Partitions

The input data is represented by a graph $\mathscr{G}(V,E)$, consisting of a set of nodes V and edges E, which we call the Graph of Superimposed Partitions (GSP).

Assignment of a Node to Each Compact Partition. Let us consider a certain compact partition $\mathscr{I}(\mathbf{l})$ and call it $\mathscr{I}_\alpha$ for now. We assign a node V_α to $\mathscr{I}_\alpha$ as a representative of the group of inputs, $\mathbf{x}_i \in \mathscr{I}_\alpha$. All the input data of $\mathscr{I}_\alpha$ is therefore associated with V_α. We characterise V_α by its mean, μ_α, and the sample density of input instances, λ_α, which we define as

$$\mu_\alpha = \frac{1}{N(\mathscr{I}_\alpha)} \sum_{\mathbf{x}_i \in \mathscr{I}_\alpha} \mathbf{x}_i \quad \text{and} \tag{1}$$

$$\lambda_\alpha = \frac{N(\mathscr{I}_\alpha)}{S(\mathscr{I}_\alpha)} \ , \tag{2}$$

respectively, where $N(\mathscr{I}_\alpha)$ is the number of input instances in $\mathscr{I}_\alpha$ and $S(\mathscr{I}_\alpha)$ an approximation to its volume. We set $S(\mathscr{I}_\alpha) = \prod_{d=1}^{D} \sigma_d^2$, where σ_d^2 is the variance in the d-th coordinate direction. Note that a node V_α has by definition a common intersection index vector and we call it $\mathbf{l}_\alpha$ for convenience.

Edges between adjacent nodes. We now connect *adjacent* pairs of nodes by edges. Two nodes representing two compact partitions, $\mathscr{I}_\alpha$ and $\mathscr{I}_\beta$, are defined as adjacent when the intersection of their index vectors $\mathbf{l}_\alpha$ and $\mathbf{l}_\beta$ are different only by one element. That is, we define an edge $E_{\alpha\beta}$ between two nodes V_α and V_β if

$$H_{dist}(\mathbf{l}_\alpha, \mathbf{l}_\beta) = 1, \tag{3}$$

where $H_{dist}(\cdot,\cdot)$ stands for the Hamming distance of the two vectors. We associate $E_{\alpha\beta}$ with a weight $w_{\alpha\beta}$, which we define as

$$w_{\alpha\beta} = \frac{N(\mathscr{I}_\alpha) + N(\mathscr{I}_\beta)}{S(\mathscr{I}_\alpha) + S(\mathscr{I}_\beta)} \ . \tag{4}$$

Algorithm 4 describes the process of generating the GSP.

2.3 Graph Clustering Using the Markov Cluster Algorithm

In order to cluster the GSP, we use the Markov Cluster (MCL) algorithm [20]. The main idea of the algorithm is to simulate flow within the graph, increasing weights where it is strong and decreasing weights when it is weak. After convergence regions of constant flow remain which are separated by edges with zero flow, defining a clustering of the nodes.

This idea is formulated as a random walk within the graph. First the graph is transformed to a Markov graph, i.e. a graph where for all nodes the weights of outgoing edges sum to one. Flow is simulated by computing powers of the Markov matrix, corresponding to flow expansion. An additional *inflation* operator is inserted to allow weighting of the flow. The MCL process consists of alternately applying expansion and inflation steps to the same stochastic matrix until convergence. The contraction and expansion parameters of the MCL process influence the resolution of the output. An implementation of the algorithm that exploits the sparsity of the graph lends has shown to be scalable to very large numbers of nodes. Given certain sparseness conditions, th algorithm has a complexity of $O(Nk^2)$, where N is the number of nodes, and k is the average of neighbours of nodes in the graph [20].

3 Experiments

This section illustrates the performance of the proposed clustering algorithm. We also apply the method to the task of automatic video segmentation.

```
ADDALLEDGES()
1  for treeId in Trees
2      for nodeClusterId in Graph
3          Intersections = allIntersection(treeId, nodeClusterId[treeId])
4          for otherClusterId in intersections
5              if hammingDistance(otherClusterId, nodeClusterId) == 1
6                  addEdge(nodeClusterId, otherClusterId)

ALLINTERSECTION(treeId, partitionId)
1  intersections = {}
2  for nodeClusterId in graph
3      if nodeClusterId[treeId] == partitionId
4          intersections.add(nodeClusterId)
5  return intersections
```

Figure 4: **Algorithm for GSP Generation**

3.1 Synthetic data

Figure 1 (a) shows the density of a 2D synthetic data set that involves non-convex manifolds and varying sample densities. Figure 8 shows that the proposed algorithm successfully identifies three clusters and assigns the cluster index to each node of the GSP using appropriate parameters. In the following we examine the effect of different parameter settings on the clustering result on the same data set.

In the stage of partitioning data by a random forest, the tree depth determines the resolution of the quantisation. When the tree depth is small, clusters cannot be distinguished from each other, see Figure 5. Currently the optimal tree depth is found on a validation set and set to a value of 8 in our experiments.

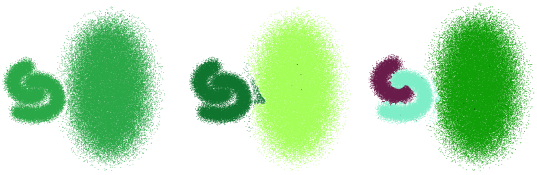
The most influential parameter of the MCL algorithm is the resolution (or *inflation*) term. A high inflation parameter leads to smaller clusters. This parameter allows a smooth control between compact partitions and clustering, see Figure 6. We consistently set the parameter value to 1.1 in our experiments to capture the structure of the data. Note that a property of MCL is that it converges more rapidly for larger values of the inflation parameter.

We further investigate the effect of the *pre-inflation parameter* π in MCL. It is the exponent applied to the edge weight, $w_{new} = w^{\pi}$, allowing modulation of the edge contrast: a high *pre-inflation* parameter will lead to a larger variation among the edge weights, resulting in a higher resolution clustering, see Figure 7. Good results have been obtained with values in the range from 1 to 4.

The three parameters, tree depth, *inflation* and *pre-inflation* all contribute to the resolution of the final clustering. They are currently found by testing them on a validation set for each application.

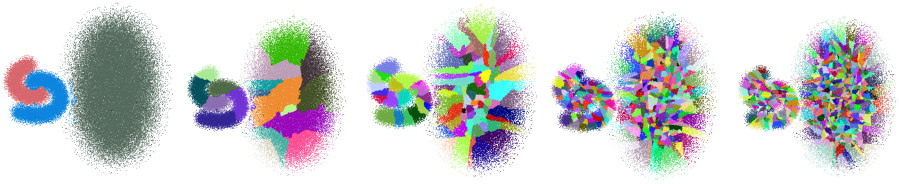
3.2 Comparisons

We compare our method against K-means [8], Mean Shift clustering [12] and Spectral Clustering [26] using publicly available code. None of these methods is able to correctly separate the correct C-shaped clusters, see Figure 9. The timing results shows that only K-means is



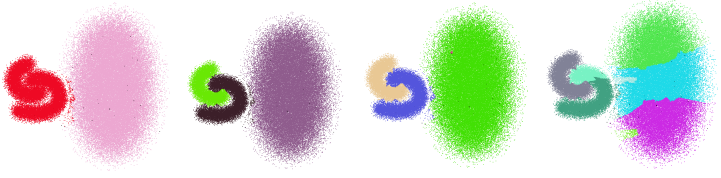
(a) depth=4 (b) depth=6 (c) depth=8

Figure 5: **Effect of tree depth.**



(a) inflation=1.1 (b) inflation=1.2 (c) inflation=1.3 (d) inflation=1.7 (e) inflation=2.0

Figure 6: **Effect of the MCL inflation term.**



(a) exponent=1 (b) exponent=2 (c) exponent=4 (d) exponent=6

Figure 7: **Effect of changing the power exponent for edge weights in MCL.**

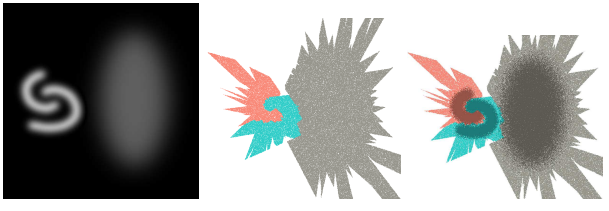
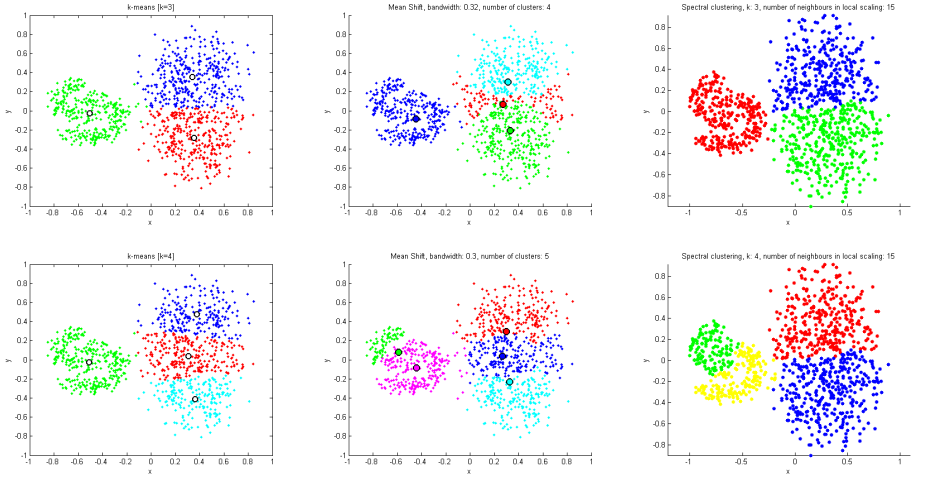


Figure 8: **Clustering Result.** *The original PDF is shown on the left. The result of the proposed clustering method is a function which, for any new data point, returns a cluster index or an 'unknown' state. This function is shown in the centre. On the right the same function is shown along with the data points. As the GSP focuses on regions of high density, the output becomes less precise when the function is computed outside of these.*

comparably efficient and only for a small (K=3) number of clusters, while Mean shift and spectral clustering are significantly slower, see Table 1.

3.3 Video Segmentation

As an example application, we apply the GSP algorithm to the video segmentation problem on two public test sequences. The data space is six-dimensional and includes the pixel coordinates, time index and the colour values: $\mathbf{x} = [x, y, t, r, g, b]^{\mathrm{T}}$, each coordinate scaled



(a) *K*-Means (b) Mean Shift (c) Spectral Clustering

Figure 9: **Comparison of Clustering Algorithms.** *Results on synthetic data set sampled from the distribution in Figure 1(a). For K-means, Mean Shift and Spectral Clustering results using two different parameter settings are shown. None of the algorithms captures the correct C-shaped clusters.*

Method	No. samples $\times 10^3$	Time [s]	Time/10^6 samples [s]
K-means, $k = 3$	1000	10	10
K-means, $k = 4$	1000	29	29
Mean Shift, $bandwidth = 0.30$	200	127	635
Mean Shift, $bandwidth = 0.32$	200	191	955
Spectral Clustering	3	180	60000
GSP	5000	112	22

Table 1: **Computation time comparison.** This table shows the computation times (in seconds) of various clustering methods applied to the synthetic 2D data set in Figure 8. The last column shows the computation time taken per millions of sample points. The proposed method is comparable with an efficient *K*-means implementation and significantly faster that Mean Shift and Spectral Clustering.

to the range $[0,1]$. The clustering algorithm is run on the complete space-time volume and therefore insures that segmented regions are consistent over time.

The algorithm is run with an increasing number of tree depth, where the clustering resolution increases with tree depth, see Figures 10 and 11. The segmentation results show plausible segmentations at multiple resolutions of regions of similar colour. These may be used to generate layered image representations or as input for further processing.

The computation time for segmenting a sequence of 30 frames of size 200×130 at the highest resolution setting was approximately 10 minutes on an Intel Xeon 3.2 GHz machine.

4 Discussion

This paper introduced a novel clustering algorithm based on a Graph of Superimposed Partitions (GSP) generated with a random forest. The resulting graph clustering problem is solved using the Markov Clustering algorithm. The method is able to recover non-convex structure

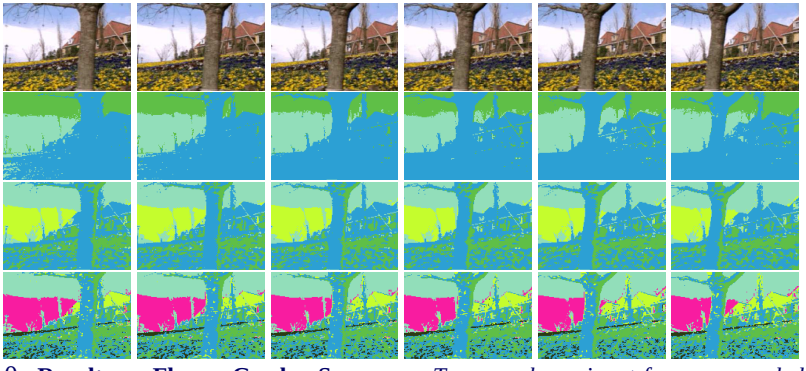


Figure 10: **Results on Flower Garden Sequence.** *Top row shows input frames, rows below show segmentation results with increasing cluster resolution parameter.*

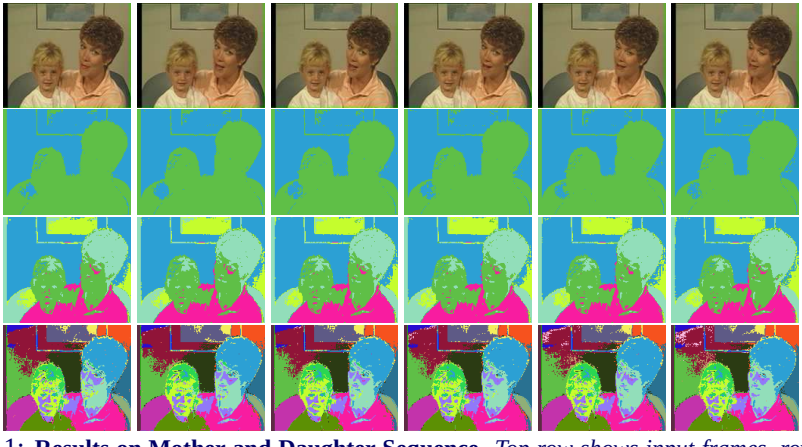


Figure 11: **Results on Mother and Daughter Sequence.** *Top row shows input frames, rows below show segmentation results with increasing cluster resolution parameter.*

and can efficiently handle large data sets. This scalability is required in computationally expensive applications such as video segmentation. We have investigated the influence of some parameters and compared the performance with three existing clustering algorithms on synthetic data. The algorithm was also applied to video segmentation of standard test sequences.

Some compact partitions are not represented in the GSP due to a lack of samples; extending the graph by including those extra node remain an open question. Since the GSP performs a remapping of input-space distances, one direction for future research includes the comparison to other graph-based methods for dimensionality reduction such as Locally Linear Embedding (LLE) and Isomap.

Acknowledgments The authors would like to thank Roberto Cipolla for valuable discussions and the reviewers for constructive comments.

References

[1] E. H. Adelson and J. Y. A. Wang. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3:625–638, 1993.

[2] H. Blockeel, L. D. Raedt, and J. Ramong. Top-down induction of clustering trees. In *ICML*, pages 55–63, 1998.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):790–799, 1995.

[5] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE-PAMI*, 24: 603–619, 2002.

[6] D. DeMenthon. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *Statistical Methods in Video Processing Workshop*, 2002.

[7] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[8] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, pages 624–630, 2003.

[9] X. Z. Fern and C. E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *ICML*, 2003.

[10] B. Fischer, V. Roth, and J. M. Buhmann. Clustering with the connectivity kernel. In *NIPS*, 2004.

[11] Y. Freund, S. Dasgupta, M. Kabra, and N. Verma. Learning the structure of manifolds using random projections. In *NIPS*, 2007.

[12] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975.

[13] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[14] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[15] S. Khan and M. Shah. Object based segmentation of video using color, motion and spatial information. In *CVPR (2)*, pages 746–751, 2001.

[16] B. Liu, Y. Xia, and P. S. Yu. Clustering through decision tree construction. In *CIKM*, pages 20–29, 2000.

[17] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *NIPS*, pages 985–992, 2006.

[18] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE-PAMI*, 22:888–905, 2000.

[19] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, 2008.

[20] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[21] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[22] J. Wang, B. Thiesson, Y. Xu, and M. F. Cohen. Image and video segmentation by anisotropic kernel mean shift. In *ECCV (2)*, pages 238–249, 2004.

[23] Y. Weiss. Segmentation using eigenvectors: A unifying view. In *ICCV*, pages 975–982, 1999.

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

[25] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *Trans. Computers*, C-20 (1):68–86, January 1971.

[26] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *NIPS*, 2004.