SCALABLE SEMIDEFINITE MANIFOLD LEARNING

Nikolaos Vasiloglou, Alexander G. Gray, David V. Anderson

Georgia Institute of Technology
Atlanta GA 30332
nvasil@ieee.org, agray@cc.gatech.edu, dva@ece.gatech.edu

ABSTRACT

Maximum Variance Unfolding (MVU) is among the state of the art Manifold Learning (ML) algorithms and experimentally proven to be the best method to unfold a manifold to its intrinsic dimension. Unfortunately it doesn't scale for more than a few hundred points. A non convex formulation of MVU made it possible to scale up to a few thousand points with the risk of getting trapped in local minima. In this paper we demonstrate techniques based on the dual-tree algorithm and L-BFGS that allow MVU to scale up to 100,000 points. We also present a new variant called Maximum Furthest Neighbor Unfolding (MFNU) which performs even better than MVU in terms of avoiding local minima.

1. INTRODUCTION

Maximum Variance Unfolding (MVU) is one of the state of the art Manifold Learning (ML) algorithms [**?**]. In his original paper Weinberger showed that MVU is empirically the best unfolding method that recovers the intrinsic dimension of a manifold compared to other known ML algorithms. Unfortunately MVU is scalable up to a few hundred points because it is cast as a Semidefinite Program (SDP) [**?**] and its complexity is cubic. In [**?**] a non convex formulation of MVU known as NCMVU was presented based on the non-convex SDP framework developed in [**?**]. This method [**?**] has linear complexity per iteration based on L-BFGS algorithm [**?**], but nothing can be said about the overall complexity. Experiments showed that it speeds MVU significantly with the disadvantage of getting non-optimal solutions that correspond to local minima. Despite the speedup the method could scale up to a few thousand of points. This is mainly because MVU like any other ML technique requires the computation of all k-nearest neighbors which has quadratic complexity. Another problem is auto-tuning the optimization parameters that can distort the optimal solution with local optima.

For the first problem we considered this paper with the dual-tree algorithm [**?**] that has empirically linear complexity and speeds up the computation of neighborhoods significantly. For the second problem we introduced a different objective function for unfolding that is based on the distances of the furthest neighbors, Maximum Furthest Neighbor Unfolding MFNU. In the experiments we show that it behaves better than NCMVU in terms of local optima. We also derived an upper bound for NCMVU and MFNU that helps in auto-tuning the optimization parameters. In our experiments we were able to unfold 10dimensional 100K point datasets in 5 hours. This is the largest dataset to be unfolded based on MVU or its variants. The largest set ever processed with ML was done in [**?**], that involved 18M images, but different techniques were used.

The rest of the paper is organized as follows. In section 2 and 3 the convex and non-convex MVU algorithms are outlined. A description of all k-nearest neighbors is given in section 4. MFNU is presented in section 5 and in the end (sections 6, 6.2) implementational issues and experiments are discussed.

2. MAXIMUM VARIANCE UNFOLDING, THE CONVEX SDP CASE

Weinberger formulated the problem of isometric unfolding as a Semidefinite Programming algorithm [**?**]. According to his experiments MVU has the best performance compared to the other state of the art Manifold Learning methods.

Given a set of data $X \in \Re^{N\times d}$, where N is the number of points and d is the dimensionality. The dot product or Gramm matrix is defined as $G = XX^T$. The goal is to find a new Gramm matrix K such that $rank(K) < rank(G)$ in other words $K = \hat{X}\hat{X}^T$ where $\hat{X} \in \Re^{N\times d'}$ and $d' < d$. Now the dataset is represented by $\hat{X}$ which has fewer dimensions that X. The requirement of isometric unfolding is that the euclidian distances in the $\Re^{d'}$ for a given neighborhood around every point have to be the same as in the $\Re^d$. This is expressed in:

$$K_{ii} + K_{jj} - K_{ij} - K_{ji} = G_{ii} + G_{jj} - G_{ij} - G_{ji}, \forall i, j \in I_i \quad (1)$$

where I_i is the set of the indices of the neighbors of the ith

This work was sponsored by Google grants

point. From all the K matrices MVU chooses the one with the maximum variance. So the algorithm is presented as an SDP:

$$\begin{aligned} \max\ & \text{Trace}(K) \\ & \text{subject to} \\ Trace(A_{ij}K) &= d_{ij} \quad \forall i,j \in I_i \\ Trace(\mathbf{1}K_{ij}) &= 0 \end{aligned}$$

where A_{ij} has the following form:

$$\begin{bmatrix} 1 & 0 & \dots & -1 & \dots & 0 \\ 0 & \ddots & 0 & \dots & 0 & 0 \\ \vdots & 0 & \ddots & 0 & \dots & 0 \\ -1 & \dots & 0 & 1 & \dots & 0 \\ \vdots & 0 & \dots & 0 & \dots & 0 \\ 0 & \dots & \dots & 0 & \dots & 0 \end{bmatrix} \tag{2}$$

$$\mathbf{1} = \begin{bmatrix} 1 & \dots & \dots & 1 \\ \vdots & \ddots & \dots & 1 \\ \vdots & \dots & \ddots & 1 \\ 1 & \dots & \dots & 1 \end{bmatrix} \tag{3}$$

and

$$d_{ij} = G_{ii} + G_{jj} - G_{ij} - G_{ji} \tag{4}$$

The last condition is just a centering constraint for the covariance matrix. The new dimensions $\hat{X}$ are the eigenvectors of K. In general MVU gives Gram matrices that have compact spectrum at least more compact than traditional linear Principal Component Analysis (PCA). Unfortunately this method can handle datasets of no more than hundreds of points because of its complexity.

3. THE NON CONVEX MAXIMUM VARIANCE UNFOLDING

By replacing the constraint $K \succeq 0$ [**?**] with an explicit rank constraint $K = RR^T$ the problem becomes non-convex and it is reformulated to

$$\max RR^T \tag{5}$$

$$A_{ij} \bullet RR^T = d_{ij} \tag{6}$$

$$\mathbf{1} \bullet RR^T = 0 \tag{7}$$

The above problem can be solved with the augmented Lagrangian method [**?**]. Gradient descent is a possible way to solve the minimization of the Lagrangian, but it is rather slow. The Newton method is also prohibitive. The Hessian of this problem is a sparse matrix although the cost of the inversion might be high it is worth investigating. In our experiments we used the limited memory BFGS (L-BFGS) method [**?**, **?**] that is known to give a good rate for convergence.

4. COMPUTING THE NEIGHBORHOODS

As already discussed in previous section MVU and its variants require the computation of all-nearest neighbors. The all-nearest neighbor problem is a special case of a more general class of problems called N-body problems [**?**]. In the following sections we give a sort description of the nearest neighbor computation. The actual algorithm is a four-way recursion. More details can be found in [**?**].

4.1. Kd-tree

The kd-tree fig 1a is a hierarchical partitioning structure for fast nearest neighbor search [**?**]. Every node is recursively partitioned in two nodes until the points contained are less than a fixed number. This is a leaf. Nearest neighbor search is based on a top down recursion until the query point finds the closest leaf. When the recursion hits a leaf then it searches locally for a candidate nearest neighbor fig 1bc. At this point we have an upper bound for the nearest neighbor distance, meaning that the true neighbor will be at most as far away as the candidate one. As the recursion backtracks it eliminates (prunes) nodes that there are further away than the candidate neighbor. Kd-trees provide on the average nearest neighbor search in $O(\log N)$ time, although for pathological cases the kd-tree performance can asymptotically have linear complexity like the naive method.

4.2. The Dual Tree Algorithm

In the single tree algorithm the reference points are ordered on a kd-tree. Every nearest neighbor computation requires $O(\log(N))$ computations. Since there are N query points the total cost is $O(N \log(N))$. The dual-tree algorithm [**?**] orders the query points on a tree too. If the query set and the reference set are the same then they can share the same tree. Instead of querying a single point at a time the dual-tree algorithm always queries a group of points that live in the same node. So instead of doing the top-down recursion individually for every point it does it for the whole group at once. Moreover instead of computing distances between points and nodes it computes distances between nodes. This is the reason why most of the times the dual-tree algorithm can prune larger portions of the tree than the single tree algorithm. The complexity of the dual-tree algorithm is empirically $O(N)$. If the dataset is pathological then the algorithm can be of quadratic complexity too.

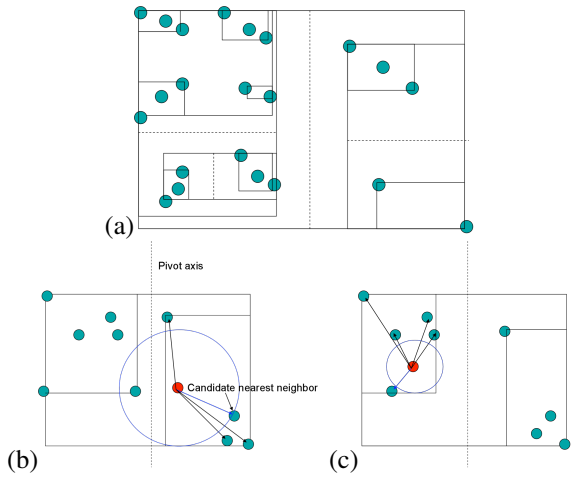

Fig. 1. a)Two dimensional kd tree. The solid lines indicate the bounding boxes, while the dashed lines indicate the splitting dimension on the node b) The red point is the query point. The single tree algorithm recurses to the closest leaf following always the closest node. Then it finds localy the candidate neighbor. In this case the algorithm has to search the left leaf too. c)In this case the candidate nearest neighbor is the true too. The sphere around the query point doesn't intersect the other leaf

5. MAXIMUM FURTHEST NEIGHBOR UNFOLDING

Weinberger presented a physical explanation of MVU by simulating every point with a metallic ball connected to its neighbors with a rod. Every rod can freely rotate around the ball. Every ball tries to move far away from the origin bound to its neighbors. Mathematically this is expressed as variance maximization.

Instead of maximizing the variance which is equivalent to maximizing the distance of every point from the origin we propose maximization of the distance between furthest neighbors. So the objective function becomes

$$\max \sum_{i=1}^{N} C_i \bullet RR^T \tag{8}$$

where C_i selects the pair of furthest neighbor, it has similar structure with (2). This formulation as we will see later leads to better unfolding of the manifold, bypassing local minima. Computing the furthest neighbors is an N-body problem too, meaning that the naive method would be of $O(N^2)$ complexity. It turns out that the dual-tree and single tree algorithms can efficiently compute the furthest neighbors too. The only difference is that in the top down recursion they have to choose the furthest node instead of the nearest.

6. IMPLEMENTATION AND EXPERIMENTS

Our experiments target datasets from 5K up to a 100K points. All the algorithms were developed in C++ as part of the Fastlib library [**?**], that uses data structures based on BLAS and LAPACK that are known to be optimal for linear algebra operations. Moreover Fastlib has several other optimizations ideal for machine learning algorithms. Fastlib also contains algorithms for kd-trees and fast all-nearest neighbor implementation. All the experiments were ran on identical dual Xeon 3.00GHz, 64-bit processors with 8 GB RAM and with the hyperthreading off. The following objective functions were tested:

1. Maximum Variance Unfolding
2. Maximum Furthest Neighbors Unfolding

6.1. Implementation Issues of the Augmented Lagrangian and L-BFGS

Before presenting the results the authors think it is necessary to mention some details on the parameters of the optimization methods. As mentioned above, the augmented Lagrangian method has some parameters that need to be tuned. First of all the memory of the L-BFGS method was chosen to be 50.

The sigma (penalty parameter) is the most critical one since if it is very small then the method does not converge. The solution is moving away from the feasible domain. If sigma is very high then the method moves very quickly to feasible domain without giving the opportunity for the variance to be maximized. Our strategy was to always start from low sigmas and if the objective function exceeds an upper bound sigma is increased. Eventually sigma will get the right value. It turns out that a reasonable upper bound for the objective (distance of furthest neighbors) in the optimization problem is the following:

$$B = d_{max} * N^2 \tag{9}$$

where d_{max} is the maximum nearest neighbor distance in the set. The geodesic distance between two furthest neighbors cannot be greater than Nd_{max}. This upper bound is also valid for the maximum variance case since the variance is the distance from the axis origin which is always less than the distance of the furthest neighbors.

Another parameter of the method is the k-neighborhood. This can be set ad-hoc or it can be tuned. In the experiments, the technique of leave-one-out cross validation was used described in [**?**]. As it will be shown in the following sections, bad choice of the k-parameter can give wrong results.

The only parameter that still remains ad-hoc is the norm gradient accuracy for optimization for a fixed penalty parameter (sigma). As a heuristic we found out that when

the norm gradient is less than sigma the inner optimization should terminate, although we noticed that it could have been set in a higher value. The whole optimization algorithm terminates when the feasibility error is below a certain value.

6.2. Datasets

For our experiments we used the following datasets:

1. A three dimensional Swiss roll ranging from 1000 points up to 100K
2. Speech features from the TIMIT database
3. The Corel image features

Detailed descriptions of the datasets can be found at [**?**], while a short descriptions can be found in table 6.2 The goal

dataset	points	dimension
Swiss roll	up to 100,000	3
TIMIT MFCC features	100,000	39
Corel color histogram	68,040	32
Corel color moments	68,040	9
Corel textures	68,040	16

Fig. 2. Dataset description

of the experiments was to visualize the datasets so we tried low dimensional embedding to 2, 3 dimensions or up to the dimension for which the optimization method would give a satisfactory feasibility error.

Swiss roll experiments. Swiss roll has been a benchmark for manifold learning, but the experiments have been limited to few hundreds of points. In these experiments the goal is to investigate the scalability of the MVU and MFNU and the quality of the results too. In fig. 7 we see the tremendous difference between convex MVU and NCMFNU, as it is 6 orders of magnitude slower for 100K points.

Maximum Furthest Neighbors Unfolding. In fig. 7.a we see that the algorithm scales in a quasi-linear way. The jump between 50 and 70 thousand points is because we had to increase the number of neighbors. In fig. 7.c we see that the number of iterations has a linear trend which means that the whole complexity of the algorithm has to be quadratic asymptotically. The reason why we don't notice clear quadratic behavior is because all linear operations of LBFGS don't scale linearly because of the BLAS implementation. BLAS has almost constant behavior for small vectors and asymptotically linear. In fig.7.c we see that the objective function increases linearly, which is something expected and a good way to verify that the algorithm works, although the results were also visually inspected in order to verify convergence of the algorithm. In fig. 7.d the number of constraints versus the number of points is plotted. When the k-neighborhoods are computed it is obvious that some of them will be duplicated. In this picture we see that the necessary constraints also grow linearly but they are always less than the imposed ones (solid line).

Maximum Furthest Neighbors Unfolding with auto-tuning. In fig. 7 we see the results. It is noteworthy that in the middle column the unfolded Swiss roll seems very well unfolded. The scaling seems to be be almost linear until 45000 points.

Maximum Variance Unfolding. In fig. 7 we see the scaling of MVU. By comparing fig. 7.a and 7.a we can see that MVU is faster than MFNU. Unfortunately in fig 7 we see that the MVU gives very poor results and gets trapped into local minima more easily.

Corel dataset. In fig. 7 the results from MFNU are depicted for the color moments and color histograms. We tested different k-neighborhoods and dimensions. It turns out that both datasets can be embedded in 3 dimensions. All the experiments run in reasonable time from a few minutes up to 2 hours.

TIMIT dataset. The TIMIT dataset is a benchmark speech dataset. After sampling we extracted 100,000 39-dim points that they correspond to the Mel-frequency cepstrum coefficients of 25msec frames with 0.125msec overlap. The dimension was reduced with PCA and the first 10 principal components that correspond to 96% of the total sum of eigenvalues. It took about 5 hours to unfold it in 5 dimensions. Optimization is the most intensive part as it took 95% of the time. The part of computing the nearest neighbors took only 15 minutes with the dual tree algorithm. Just as a comparison if we we using the naive method it would take 14hours just to compute the neighbors and the whole algorithm 14+5=19hours.

Although the dataset was unfolded in 5 dimensions, it turns out that 3 are the dominant ones. This is a very useful result since it shows clearly that there are many redundant dimensions on the initial 10 dimensional dataset. Due to space limitations the plots are not shown.

7. SUMMARY

In this paper we presented implementational issues of Maximum Variance Unfolding and tested it over medium size to large datasets. The contribution of this paper is mainly on the modification of the objective function from Max Variance to Max Furthest Neighbor distance, that turned out to have better performance in terms of overcoming local optima. We also presented some heuristics for auto-tuning of the augmented Lagrangian, by estimating an upper bound for the objective function. Or experiments showed that it is not yet clear if the method can scale linearly, as it depends on the dataset pathology, and on the number of the

necessary constraints. This is the first time that MVU is applied on medium to large size datasets, revealing some dimensional aspects. As a future direction we believe that analytical computation of the Hessian and investigation of the right preconditioner for the Newton Method should be considered. The authors would like to thank Professors Nemirovski and Shapiro for their help in semidefinite programming.

8. REFERENCES

[1] K.Q. Weinberger and L.K. Saul, "An introduction to nonlinear dimensionality reduction by maximum variance unfolding," *Proceedings of the Twenty First National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[2] L. Vandenberghe and S. Boyd, "Semidefinite Programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.

[3] B. Kulis, A.C. Surendran, and J.C. Platt, "Fast Low-Rank Semidefinite Programming for Embedding and Clustering," *Proc. 11th Intl. AISTATS Conference*, 2007.

[4] S. Burer and R.D.C. Monteiro, "A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization," *Mathematical Programming*, vol. 95, no. 2, pp. 329–357, 2003.

[5] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer, 1999.

[6] A. Gray and A.W. Moore, "N-Body problems in statistical learning," *Advances in Neural Information Processing Systems*, vol. 13, 2001.

[7] S. Kumar A. Talwalkar and H. Rowley, "Large-Scale Manifold Learning," *IEEE International Conference on Vision and Pattern Recognition*, 2008.

[8] D.C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989.

[9] J.H. Friedman, J.L. Bentley, and R.A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[10] Garry Boyer, Ryan Riegel, Nikolaos Vasiloglou, and Alexander Gray, "FASTlib Design and Development Manual, Version 0.1," Tech. Rep., Georgia Institute of Technology, 2008.

[11] N. Vasiloglou, A.G Gray, and DV Anderson, "Parameter Estimation for Manifold Learning, Through Density Estimation," *Machine Learning for Signal Processing, 2006. Proceedings of the 2006 16th IEEE Signal Processing Society Workshop on*, pp. 211–216, 2006.

[12] http://archive.ics.uci.edu/ml/datasets.html.

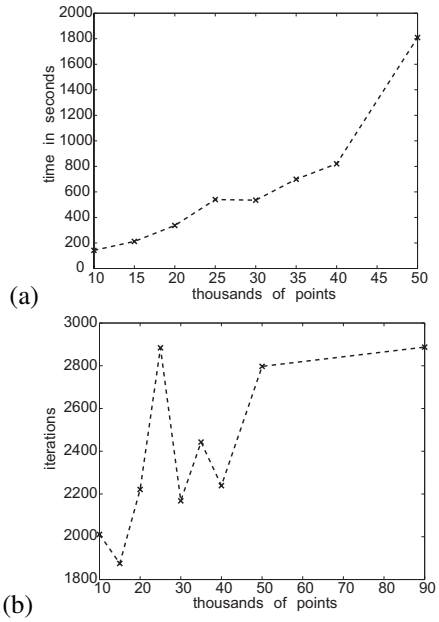

Fig. 3. The classic MVU algorithm a)Scaling performance b)Iterations required for the optimization

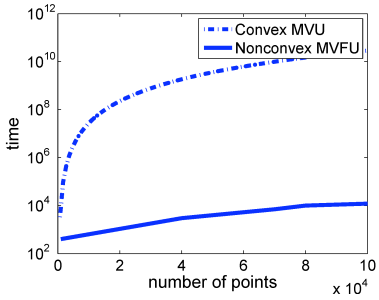

Fig. 4. Convex MVU vs non-convex MFNU. For the convex MVU we run experiments up to 600 points and then extrapolated

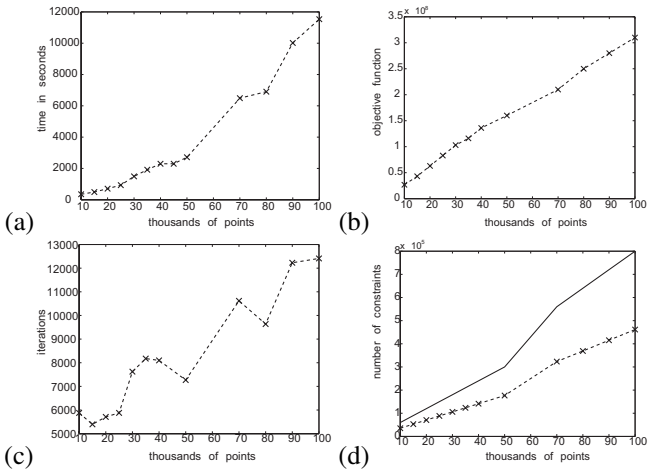

Fig. 5. The MFNU performance, a)Scaling of the MFNU b)Maximization results of the Maximum Furthest Neighbors objective c)Iterations required for the optimization d)Number of constrained kN (solid line), consolidated constraints (dashed line)

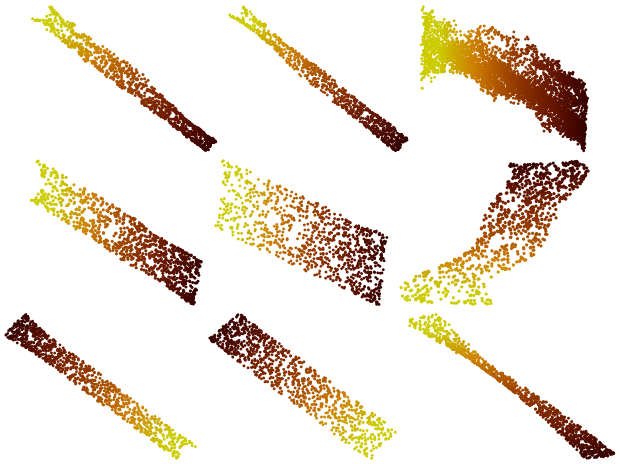

Fig. 7. Unfolded Swiss rolls 10K, 20K, 40K (top to bottom row), (left column) MFNU, (center column)MFNU with auto-tuning for k-neighborhoods, (right column)MVU. All images have been sampled showing only 4000 points, because eps files get too big

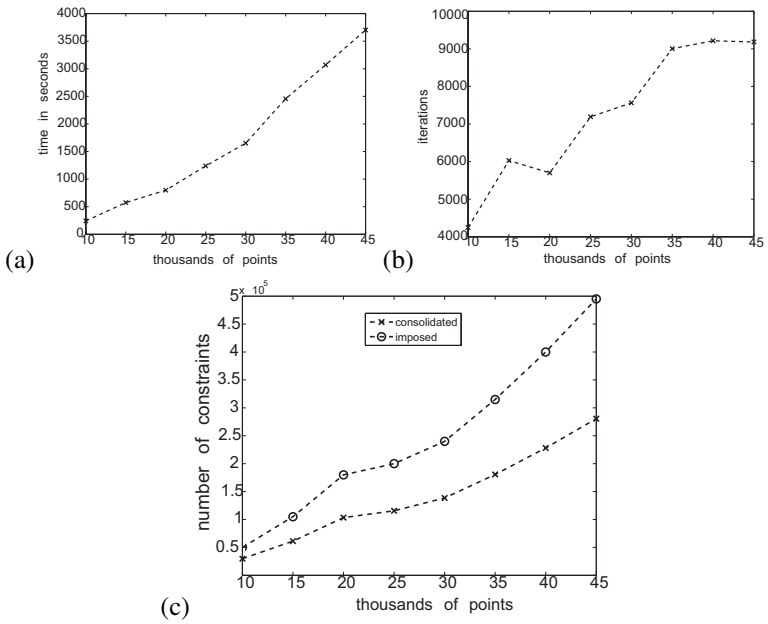

Fig. 6. The MFNU algorithm with auto-tuning of k-neighborhoods, a)Scaling of the algorithm b)Iterations required for the optimization c)Number of constrained kN (solid line), consolidated constraints (dashed line)

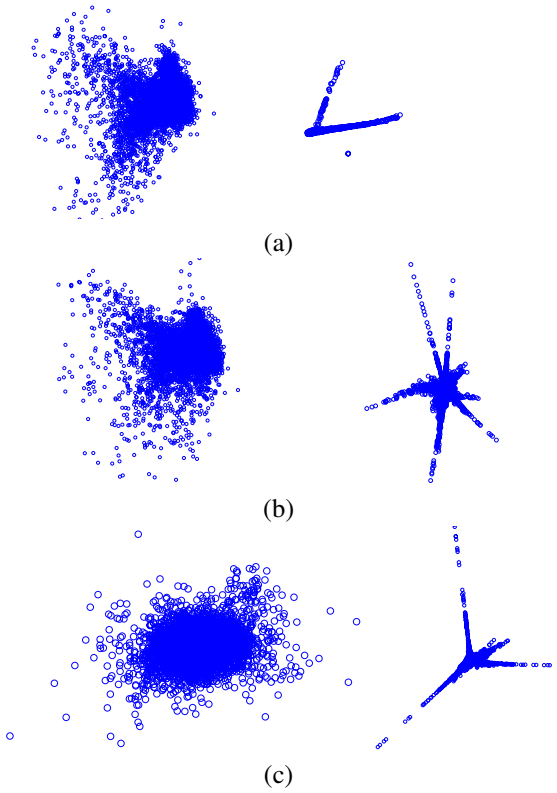

Fig. 8. (left column) corel color moments, (right column) corel color histogram, (a)4-point neighborhood, (b)5-point neighborhood, (c)7-point neighborhood