Clustering with Hypergraphs: The Case for Large Hyperedges

Pulak Purkait[a], Tat-Jun Chin[a], Hanno Ackermann[b] and David Suter[a]

[a]The University of Adelaide, [b] Leibniz Universität Hannover

Abstract. The extension of conventional clustering to *hypergraph clustering*, which involves higher order similarities instead of pairwise similarities, is increasingly gaining attention in computer vision. This is due to the fact that many grouping problems require an affinity measure that must involve a subset of data of size more than two, i.e., a *hyperedge*. Almost all previous works, however, have considered the smallest possible hyperedge size, due to a lack of study into the potential benefits of large hyperedges and effective algorithms to generate them. In this paper, we show that large hyperedges are better from both theoretical and empirical standpoints. We then propose a novel guided sampling strategy for large hyperedges, based on the concept of *random cluster models*. Our method can generate pure large hyperedges that significantly improve grouping accuracy without exponential increases in sampling costs. In the important applications of face clustering and motion segmentation, our method demonstrates substantially better accuracy and efficiency.

Keywords: Hypergraph clustering, model fitting, guided sampling.

1 Introduction

We follow the nomenclature of [1] in describing hypergraph clustering. A hypergraph $\mathcal{H} = (V, E)$ consists of a set of vertices V and hyperedges E. A hyperedge $e \in E$ is a subset of the vertices, i.e., $e \subseteq V$, and e is said to be incident with a vertex v if $v \in e$. For a weighted hypergraph $\mathcal{H}$, a weight $w(e)$ is associated with each hyperedge. The degree $\delta(e)$ of a hyperedge e is the number of vertices in e, i.e., $\delta(e) = |e|$. The degree $d(v)$ of a vertex v is the sum of all the weights of the hyperedges incident with v, i.e., $d(v) = \sum_{e \in E | v \in e} w(e)$. If all hyperedges have the same degree r, the hypergraph is said to be *r-uniform*. Clearly 2-uniform hypergraphs are normal graphs where an edge connects only two vertices.

The concept of hypergraph clustering generalises the traditional notion of clustering, whereby the affinity measure is now defined over more than a pair of vertices. In the graph partitioning view, a partitioning of $\mathcal{H}$ separates the vertices V into mutually exclusive clusters. In particular, a 2-way partitioning results in (V_1, V_2), where $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. The "goodness" of the partitioning is inversely proportional to the cost of the cut that separates the vertices, which in turn is a function of the weights of the hyperedges with vertices in both clusters. Methods have been proposed to find the best cut [1], given arbitrary $\mathcal{H}$. Below, we give examples of grouping problems based on hypergraph clustering.

Multiple line fitting. Consider a set of points V in 2D which are known to lie on a set of K lines. Our goal is to cluster V into K groups based on their line membership. The traditional concept of clustering cannot be applied here, since any two points lie exactly on a line and thus have infinite affinity. To construct useful similarity measures, we must consider more than two points. Given a subset $e \subset V$ with $|e| > 2$, we can fit a line through e (e.g., via least squares) and convert the error of the fit d into an affinity measure $w(e) = \exp(-d^2/\sigma^2)$. A hyperedge e thus conveys the existence of a line with evidence $w(e)$.

Subspace clustering. Consider a set of vectors V in $\mathbb{R}^D$, e.g., face images or feature trajectories, where the vectors are known to lie on K p-dimensional subspaces. To cluster V into K groups, we must consider affinity measures defined over more than p vectors. Given a hyperedge $e \subset V$ with $|e| > p$, a subspace can be fitted (e.g., via SVD) and the fitting error converted into a weight as above.

Whilst the above examples relate to linear models, hypergraphs can also be defined using nonlinear models, e.g., projective entities like homography. The key requirement is a geometric model that describes the "shape" of the underlying clusters. The order p of the model is the minimum number of data to instantiate the model, and a valid hyperedge must thus be larger than p. Theoretically, the total number of hyperedges is $\mathcal{O}(2^{|V|})$, which is gargantuan for even small $|V|$.

In practice, therefore, approximations are necessary. An overwhelming majority (if not all) of the previous works that utilised the hypergraph formalism [2–8] limited the hyperedges to size $p+1$, i.e., the smallest possible. The resulting hypergraph is thus $(p+1)$-uniform. Even with this limit, the number of possible hyperedges $\binom{|V|}{p+1}$ is too large for exhaustive listing. Previous works thus *sample* the set of hyperedges to construct "sparse" hypergraphs. Concurrently, limiting to the smallest size $p+1$ also maximises the chance of recovering *pure* hyperedges, i.e., those containing data that are likely from the same clusters.

Whilst computational feasibility was the overriding factor in imposing the size constraint, we nevertheless ask in this paper, *is there any benefit in using large hyperedges (size $> p+1$) for higher-order grouping?* This is a natural question since the hypergraph formalism theoretically allows hyperedges of arbitrary size, and using the smallest allowable size seems to limit the potential. We will answer in the affirmative and provide theoretical and empirical justifications.

Secondly, *how can we sample large hyperedges without expending too much effort?* Previous guided sampling approaches for hyperedges select data one-by-one [7, 4]. Extending them to large hyperedges will inevitably expose them to the effects of exponentially decreasing probability of sampling pure hyperedges. We propose a novel guided sampling strategy to sample large hyperedges based on *random cluster models* [9, 10]. Our method generates large pure hyperedges accurately without exponential increases in computational effort.

In practical applications, our approach outperforms previous hypergraph clustering methods on face clustering and motion segmentation. Notwithstanding the usage of large hyperedges, our guided sampling strategy enables our technique to be orders of magnitude more efficient (in terms of number of hyperedges and actual time required) than previous hypergraph clustering algorithms.

1.1 Previous work

Hypergraph clustering has a long history in VLSI [11]. There, vertices correspond to circuit elements and hyperedges correspond to wiring that may connect more than two elements. Finding the minimum cost cuts allows to divide the elements into modules with minimum interconnections. Unlike in computer vision, the hyperedges arise from the circuit design and need not be sampled.

The introduction of hypergraph clustering to computer vision and machine learning is relatively recent [3, 2]. Zhou et al. [12] generalised the popular Normalised Cut (NCut) algorithm [13] to the hypergraph setting. A more theoretical work by Agarwal et al. [1] analysed different existing hypergraph methods and showed that they can all be expressed as different clique projection techniques onto ordinary graphs. More recent works have largely followed the basic concepts, with various algorithmic extensions [4–8, 14]. Applications include face image clustering [3], motion segmentation [2, 4, 7], grouping of categorical data [12], and plane segmentation in RGBD data [8] and two-view images [6].

As mentioned earlier, for feasibility the methods [2–8] considered only $(p+1)$-uniform hypergraphs. Moreover, most of the methods either use random sampling [2, 6] or spatial proximity sampling [7] to generate hyperedges. Chen and Lerman [4] presented an incremental clustering and sampling technique which is more accurate than random or proximity sampling. However, a naive extension to large hyperedges is unworkable, since the probability to sample good hyperedges vanishes quickly with successive data selection. Pham et al. [10] proposed using random cluster models [9] to sample large clusters directly for geometric fitting, however, they did not pose their problem as hypergraph clustering. Moreover, they obtain intermediate clustering via graph cuts, which is relatively expensive. Nonetheless, we will combine the ideas from [4, 10] to design our algorithm.

By attempting to linearly reconstruct a point from a sparse set of neighbours, Sparse Subspace Clustering (SSC) [15] can be seen as generating hyperedges for clustering (a point and its selected neighbours are in the same hyperedge). However, SSC is tailored for linear models and is thus inapplicable in nonlinear/nonsubspace models such as homography and affine plane.

1.2 Dense versus sparse sample reuse

It is crucial to reconcile the seemingly different tensor decomposition approach [2, 8] with hypergraph clustering [3, 1]. In [2, 8], subsets of V of size p are sampled. For each p-tuple, the model is instantiated and *evaluated* (the affinity value is calculated) with respect to all points in V. Each p-tuple thus generates a "row" in the $(p+1)$-dimensional affinity tensor, or equivalently, $|V|$ hyperedges of degree $p+1$ in a hypergraph. Govindu showed how the (sampled) affinity tensor can be flattened into a matrix and decomposed [2]. Compared to methods that directly sample hypedges of size $p+1$ without testing them with the rest of the data [3–7], evidently [2, 8] extract more information per sample. It was shown in [8] that this "dense" sampling approach outperforms "sparse" sampling, given the same sampling effort. We adopt the dense reuse idea in our paper.

2 Why use large hyperedges?

2.1 Theoretical justifications

We will motivate using NCut [13], which is arguably one of the most well known clustering techniques in computer vision. Zhou et al. [12] have generalised NCut to the hypergraph setting. Let (S, S^c) be a partitioning of the vertices V in $\mathcal{H}$, where $S \cup S^c = V$. The corresponding cut has the cut set $c(S, S^c) = \{e \in E | e \cap S \neq \emptyset, e \cap S^c \neq \emptyset\}$, i.e., the removal of the hyperedges in $c(S, S^c)$ yields the disjoint sets (S, S^c). Following [12], the volume or cost of the cut is

$$\text{vol}(S, S^c) = \sum_{e \in c(S, S^c)} w(e) \frac{|e \cap S||e \cap S^c|}{\delta(e)}. \tag{1}$$

If the hypergraph is 2-uniform, $|e \cap S| = |e \cap S^c| = 1$ and $\delta(e) = 2$ for all e, and (1) reduces to the cut cost on a normal graph. The volume of cluster S is

$$\text{vol}(S) = \sum_{v \in S} d(v). \tag{2}$$

The *normalized cut* criterion for partitioning V into (S, S^c) is

$$\text{ncut}(S, S^c) = \text{vol}(S, S^c) \left(\frac{1}{\text{vol}(S)} + \frac{1}{\text{vol}(S^c)} \right). \tag{3}$$

Similar to [13], Zhou et al. [12] showed how a relaxed version of the cost function (3), which involves continuous membership labels, can be minimised globally by an eigendecomposition of an affinity matrix; see [12] for details.

Define $\alpha(e|S, S^c) := |e \cap S||e \cap S^c|/\delta(e)$. The existence of multiplier $\alpha(e|S, S^c)$ on the weight $w(e)$ differentiates NCut on hypergraphs from normal graphs. Further, $\alpha(e|S, S^c)$ can be understood as arising from the projection of the hypergraph to a normal graph [1], where each hyperedge e is replaced by a fully connected subgraph with vertices e, and each edge in the subgraph has the weight $w(e)/\delta(e)$. The cut cost (1) is exactly the cut cost on the projected graph.

We can rewrite $\alpha(e|S, S^c) = \alpha(e|\eta) := \eta(1-\eta)\delta(e)$, where $\eta := |e \cap S|/\delta(e)$ is a *size ratio* of the partitioning of e based on the cut (S, S^c). Trivially, the range

$$\frac{1}{\delta(e)} \leq \eta \leq \frac{\delta(e) - 1}{\delta(e)} \tag{4}$$

can be established. Fig. 1(a) plots $\alpha(e|\eta)$ against η for e's of different degrees. First, it is clear that the cost of cutting a hyperedge e is the highest if e is divided into equal halves (i.e., $\eta = 0.5$). More crucially, given the same η, the multiplier $\alpha(e|\eta)$ is *always higher for larger hyperedges*, since the numerator in $\alpha(e|\eta)$ increases quadratically while the denominator increases linearly. Hence, given two hyperedges of the same weight $w(e)$ and the same η, *NCut will inherently favour preserving the larger hyperedge and cutting the smaller hyperedge.*

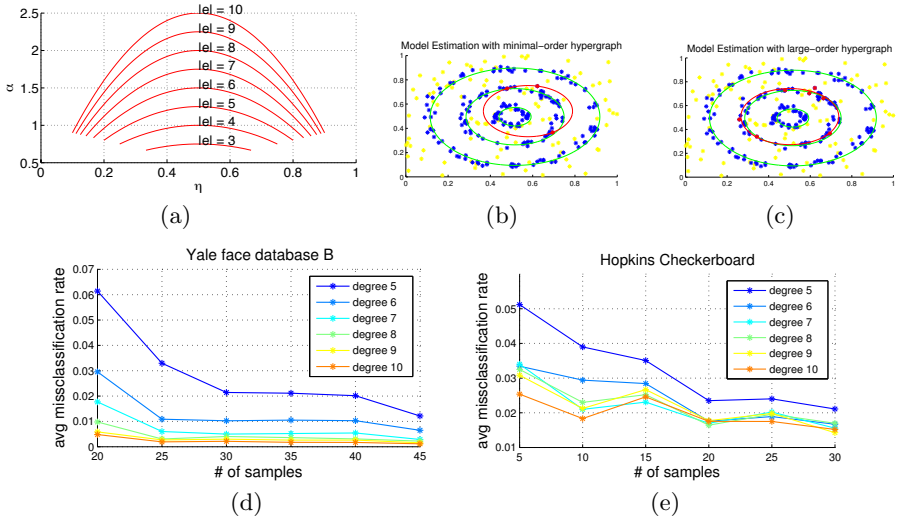


Fig. 1. (a) Plot of $\alpha(e|\eta)$ against η for e's of different degrees. (b–c) NCut segmentation (first cluster only) on a 4-uniform and 8-uniform hypergraph, respectively. (d–e) Average misclassification rate plotted against number of samples and size of hyperedges on Yale Face Database B and the 3-motion checkerboard sequences from Hopkins 155.

Effectively, the algorithm trusts larger hyperedges more than smaller hyperedges. Despite the purely algebraic motivation, the preference can be rationalised since larger hyperedges convey more evidence on the existence of a cluster than smaller hyperedges, even if the model is fitted equally well in both cases. Further, a larger data subset statistically constrains the model better and can more confidently estimate the parameters under noise. Figs. 1(b) and 1(c) show NCut segmentation results (only the first cluster is shown) for circle fitting ($p = 3$) on a 4-uniform hygergraph and a 8-uniform hypergraph. Four points cannot constrain a circle well, especially if they are spatially close, thus the result is poor. In contrast, using large hyperedges fits the circle accurately.

The algebraic derivations above were based on NCut [12]. Can we generalise the result to other hypergraph clustering algorithms? The source of the inherent bias in NCut is the hypergraph projection style that converts a hyperedge to a fully connected subgraph. Any algorithm that conducts such a projection (either implicitly or explicitly) contains this intrinsic bias. For example, it can be shown that *clique averaging* [3] and *max projection* [7] also have this property.

2.2 Empirical justifications

The dominant factor for using the smallest allowable hyperedges previously is the lack of algorithms that can sample large hyperedges; to tackle this issue, we will propose a guided sampling algorithm in Sec. 3. In this subsection, we demonstrate the benefits of using large hyperedges via an "oracle sampler" that

produces hyperedges based on ground truth cluster labels. This serves to preclude the effects of inaccuracies in an imperfect sampling technique. Any differences in the results is thus largely due to using hyperedges of different degrees.

Given data (vertices) V, we wish to cluster them into K groups, where each group lies on a model of order p. Given a fixed hyperedge degree $D > p$, we sample M subsets of size $D-1$ from V. Half of the samples are chosen purely within true clusters (evenly among K groups), while the other half is chosen randomly to simulate sampling inaccuracies. The model is fitted on each sample via SVD and evaluated against V. Each sample thus produces $|V|$ hyperedges of degree D; recall that we adopt Govindu's [2, 8] dense sample reuse approach. We obtain an affinity matrix from all the hyperedges and conduct K-way partitioning [12]. The average misclassification rate is then obtained by comparing with the ground truth. The steps are repeated by varying D and M. Note that the model complexity is fixed at p, whereas the hyperedge degree varies with D.

We performed the above on Yale Face Database B [16] and the 26 checkerboard sequences with 3 motions from the Hopkins 155 dataset [17]. We used the frontal face images of $K = 10$ persons under 64 illumination conditions from Yale Face Database B. Under the Lambertian assumption, the images of the same face lie on a low-dimensional subspace; we chose this dimension to be $p = 4$ since this gives the best results. Each checkerboard sequence contains $K = 3$ distinct rigid motions. Under the affine camera model, the trajectories on these distinct motions can be grouped into 4-dimensional subspaces ($p = 4$). Results are presented in Figs. 1(d) and 1(e). Clearly, there is an improvement in accuracy as the degree of the hyperedges is increased.

We have shown that large hyperedges are better from both theoretical and empirical standpoints; a straightforward question that arises is how large should the hyperedges be? Our answer is "reasonably large but not too large". Very large hyperedges require more effort for fitting, and may not significantly improve the accuracy further (the trends in Figs. 1(d) and 1(e) show diminishing increases in accuracy). In all our experiments we have used hyperdeges of degrees 10–20; these numbers are significantly larger than the $p + 1$ used in previous works.

3 Guided sampling for large hyperedges

Guided sampling for data subsets has primarily been used in robust model fitting [18–20]. However, following the RANSAC tradition only *minimal subsets* (of size p) have generally been considered. An exception is Pham et al. [10] who used the concept of random cluster models (RCM) [9] to sample large data subsets. However, they did not pose their work as hypergraph clustering, and their method requires graph cuts and Delaunay triangulation to produce intermediate clustering, which are relatively expensive. To construct our novel algorithm, we will draw from RCM and ISS [4]. Note that [4] considered only the smallest possible hyperedge (size $p+1$), which is only one larger than minimal subsets [18–20].

3.1 Generating auxiliary graph

RCM requires a spatial neighbourhood structure which is represented as a graph $G^{(0)} = (V, E^{(0)})$; whilst $G^{(0)}$ shares the same vertices V as the hypergraph $\mathcal{H}$, $G^{(0)}$ is a normal graph that encodes spatial proximity and not affinity to the geometric structures. To create $G^{(0)}$, we first perform PCA on the data $V = \{v_i\}_{i=1}^N$. This is obtained from the SVD of the mean-subtracted data matrix $\hat{V} = [\hat{v}_1\ \hat{v}_2\ \ldots\ \hat{v}_N]$, where $\hat{v}_i = v_i - \mu$, and $\mu = (1/N)\sum_i v_i$. Let U^l be the first-l left singular vectors of $\hat{V}$. The reduced-dimension version of V is thus $Y = (U^l)^T\hat{V}$. We create $E^{(0)}$ using k-nearest neighbours (knn) on $Y = \{y_i\}_{i=1}^N$. Two vertices v_i and v_j are connected by an edge $e =< i, j >\in E_0$ if y_i is a knn of y_j or vice versa, i.e., the edges are undirected. The weight p_e of the edge e is

$$p_e = \exp\left(-\frac{\|y_i - y_j\|^2}{2\sigma_e^2}\right), \tag{5}$$

where σ_e is a scale factor. The weight p_e for the edge e indicates how likely x_i and x_j are from the same structure, based purely on spatial proximity.

The parameters l, k and σ_e required for this step are as follows: $l = pK$ (model complexity times number of clusters), $k = 3$ for all experiments, and σ_e is obtained as the standard deviation of the nearest neighbour distances in Y.

3.2 RCM

Given $G^{(0)} = (V, E^{(0)})$, the Potts model is defined as the factor graph

$$P(f) = \frac{1}{Z}\prod_{e=<i,j>\in E^{(0)}} \exp(\beta_e \mathbf{1}(f_i = f_j)) \tag{6}$$

where Z is the partition function and $\mathbf{1}(.)$ returns 1 if its argument is true and 0 otherwise. The vector $f = \{f_i\}_{i=1}^N$ represents labels of the vertices V, where $f_i \in \{1, 2, \ldots, K\}$. If $\beta_e > 0$, $P(f)$ will assign higher probabilities to labellings f that contain large clusters. The relationship between β_e and p_e is given by

$$p_e = 1 - \exp(-\beta_e); \tag{7}$$

see [9, 10] for details. The Swendsen-Wang method [9] introduces the binary "bond" variables $d = \{d_e\}$ for each edge e to yield the extended Potts model

$$P(f, d) = \frac{1}{Z'}\prod_{e=<i,j>\in E^{(0)}} g(f_i, f_j, d_e), \tag{8}$$

where the factor g is defined as

$$g(f_i, f_j, d_e) = \begin{cases} 1 - p_e & \text{if } d_e = 0, \\ p_e & \text{if } d_e = 1 \text{ and } f_i = f_j, \\ 0 & \text{if } d_e = 1 \text{ and } f_i \neq f_j. \end{cases} \tag{9}$$

A realisation of (f, d) effectively partitions the vertices into a set of connected components. Each connected component is a subset of V such that all the bond variables between vertices in the component are turned on. Further, according to (9), vertices in a connected component must have the same labels.

Marginalising d in (8) returns the Potts model (6), while marginalising f in (8) yields the RCM $P(d)$. Let $c(d)$ be the number of connected components implied by a realisation of d. Then the RCM is

$$P(d) = \frac{1}{Z''} \prod_{e \in E^{(0)}} (p_e^{d_e}(1-p_e)^{1-d_e}) K^{c(d)} \tag{10}$$

where K is the number of clusters. Simulating the RCM (10) is difficult. Instead, the Swendsen-Wang method simulates the extended Potts model (8) by observing that given f, the bond variables d are independent of each other and can be sampled independently. We take advantage of this characteristic to sample large clusters from V which will form large hyperedges for the hypergraph $\mathcal{H}$.

Note that each vertex v_i in $G^{(0)}$ is on average incident with k edges, i.e., $G^{(0)}$ is sparse. Hence the total number of edges (bond variables) is $\mathcal{O}(k|V|)$, i.e., linear with $|V|$. This enables highly efficient sampling for large clusters.

3.3 Simulating the RCM for large hyperedges

Our method alternatingly updates f and samples d. Given f, we generate a number of samples of d, which give us a set of hyperedges. Given the hyperedges, we update f by carrying out NCut [12]. Details are as follows.

At iteration t, given the current labelling $f^{(t)}$, the vertices V can be partitioned into clusters $\mathcal{C}^{(t)} = (\mathcal{C}_1^{(t)}, \ldots, \mathcal{C}_K^{(t)})$ with a label $f_i^{(t)}$ for each vertex for $v_i \in V$, where $f_i^{(t)} \in 1, 2, \ldots, K$. The clustering above can be summarised by the graph $G^{(t)} = (V, E^{(t)})$ with $E^{(t)} = \{e =< i, j > | f_i^{(t)} = f_j^{(t)}\}$. Up to this stage, the bond variables in d that straddle two vertices with different labels have been turned off (set to zero) deterministically, cf. (9). The remaining bond variables in d, i.e., those that straddle two vertices with the same labels, are then sampled by turning them on/off probabilistically. Specifically, a bond variable d_e is turned on (set to one) with probability p_e. After this step, d is fully realised.

By sampling in the above manner, a cluster $\mathcal{C}_k^{(t)}$ is partitioned into a number of subclusters $(\mathcal{C}_{k,1}^{(t)}, \mathcal{C}_{k,2}^{(t)}, \ldots, \mathcal{C}_{k,R_k}^{(t)})$. Specifically, each subcluster is a set of vertices connected by bond variables that have been turned on. We then collect all the subclusters into the set $\mathcal{CP} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{KR_K}\}$. The clusters in $\mathcal{CP}$ have varying sizes, as one would expect from a random cluster model. In our algorithm, however, we fix the degree of the hyperedges to be generated to D, where $D > p$. To do so, we sample a component from $\mathcal{CP}$ based on the probability

$$q(\mathcal{P}_s|\mathcal{CP}) \propto \sum_{e \in \mathcal{P}_s} p_e, \ \forall \mathcal{P}_s \in \mathcal{CP}. \tag{11}$$

Given the chosen $\mathcal{P}_s$, we randomly sample a $(D-1)$-tuple from $\mathcal{P}_s$ and fit the geometric model of interest. The model is then evaluated against all the data in V, thus yielding $|V|$ hyperedges of degree D. One can envision a scheme where $D \in \mathbb{Z}$ is also randomly sampled. However, since we wish to more thoroughly test the effects of large hyperedges, we fix D to a constant. Of course, in the experiments, we will vary D across different instances of the sampling algorithm.

At iteration t, the above steps are repeated to produce a number of hyperedges. The newly generated hyperedges are then added to the set of hyperedges sampled thus far, and NCut [12] is executed to obtain the labelling $f^{(t+1)}$ for the next iteration. The process is terminated when either the labels f do not change significantly, or the maximum number of iterations is reached. Our method (Swendsen-Wang sampling) is summarised in Algorithm 1.

Algorithm 1 Swendsen-Wang sampling

Input: Data V, num clusters K, hyperedge degree D, iter count T and M, threshold ϵ.
Output: A set of hyperedges E of degree D for hypergraph $\mathcal{H} = (V, E)$.

1. Obtain auxiliary graph $G^{(0)} = (V, E^{(0)})$ (Sec. 3.1).
2. Initialise $f^{(1)}$ to a constant value.
3. For $t = 1, 2, \ldots, T$
   (a) From $f^{(t)}$, obtain clustering $\mathcal{C}^{(t)}$ and corresponding graph $G^{(t)} = (V, E^{(t)})$.
   (b) Repeat M times
      i. For all $e \in E^{(t)}$, turn off d_e with probability $1 - p_e$. This divides each $\mathcal{C}_k^{(t)}$ into a set of subclusters.
      ii. Collect all subclusters from $\{\mathcal{C}_k^{(t)}\}_{k=1}^{K}$ into $\mathcal{CP}$.
      iii. Remove from $\mathcal{CP}$ all components of size less than $D-1$.
      iv. Select a component $\mathcal{P}_s$ from $\mathcal{CP}$ with probability $q(\mathcal{P}_s|\mathcal{CP})$ (11), then randomly select a $(D-1)$-subset s from $\mathcal{P}_s$.
      v. Fit the model onto s and evaluate it against all data in V. Add the newly generated hyperedges to the set of all hyperedges.
   (c) Apply NCut [12] on the current hypergraph $\mathcal{H}$ to obtain labels $f^{(t+1)}$.
   (d) If the difference between $f^{(t)}$ and $f^{(t+1)}$ is smaller than ϵ, terminate sampling.

4 Results

We conduct experiments on real datasets to verify the effectiveness and efficiency of the proposed large hyperedge sampling method. Since we use Govindu's dense sample reuse approach, a D-degree hyperedge e is composed of $e = \{s \cup v\}$, where s is a sampled $(D-1)$-tuple, and v is an arbitrary vertex in V. In all the experiments, the weight $w(e)$ of a hyperedge e is calculated as

$$w(e) = \begin{cases} \exp(-r^2(v, \phi_s)/2\sigma^2) & \text{if } v \notin s; \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

where ϕ_s is the model fitted in a least squares manner on s, and $r(v, \phi_s)$ is the residual of v with respect to ϕ_s. The parameter σ is problem dependent, and a similar parameter needs to be tuned for all hypergraph clustering methods [3, 4]. We calculate σ following [4], i.e., choose the sigma that gives the lowest subspace approximation error after segmentation by Ncut; see [4] for details.

4.1 Face clustering

It has been established that a set of images of a the same face subjected to different illumination conditions can be modelled by a low-dimensional subspace [16]. Given a set of images of K different faces V, we wish to cluster them into K subspaces. We used the Yale Face Database B [16] in our experiment, which contains images of $K = 10$ faces under 64 different lighting conditions. As in Sec. 2.2, the dimensionality of the face subspace is chosen to be p = 2. Before clustering, we first reduce the dimensionality of V to $4K$ to construct the auxiliary graph $G^{(0)}$.

Sampling accuracy and effects of large hyperedges. We have tested with degree D in the range $\{6, 8, \ldots, 20\}$. Unlike in Sec. 2.2, here we sample without ground truth labels. Three methods were compared: random sampling (RS), iterative spectral sampling (ISS) [4], and Swendsen-Wang sampling (SWS). We extended ISS to sample large hyperedges by using the intermediate clustering information to successively select data. Here, NCut [12] was used for hypergraph clustering for all sampling methods. The average misclassification rate is plotted against the number of samples in Figs. 2(a)–(c). The sampling accuracy (% hyperedges containing data from the same cluster) is displayed in Fig. 2(d). For RS and ISS in Figs. 2(a)(b), as D is increased, the misclassification rate also increases; this reflects the inability of these methods to sample large pure hyperedges, and *not* because large hyperedges are ineffective; see Sec. 2.2. In contrast, SWS not only samples large hyperedges more accurately, the hypergraph clustering accuracy also improves as the hyperedge degree is increased; see Fig. 2(c).

In terms of computational effort, our method does not require discernibly higher run time than RS and ISS. Sec. 4.3 will examine run time more closely.

Benchmark against state-of-the-art. For our method (SWS followed by NCut), we chose degree $D = 20$ and 300 samples. We varied K from 2 to 10 and repeated our algorithm on 100 randomly chosen subpopulations of K persons. We compared against well-known methods for subspace segmentation: Generalised PCA (GPCA) [21], SCC [4], Sparse Subspace Clustering SSC [15], Spectral Local Best-fit Flat (SLBF) [22] and Agglomerative Lossy Compression (ALC) [23]; these methods have also recently been applied on Yale Face Database B. The mean percentage of misclassification along with the run time are shown in Table 1. The proposed method is on par with ALC which correctly clustered all the faces for all K. Moreover, our method is much faster than ALC.

4.2 Motion segmentation with sparse trajectories

Under the affine camera model, the trajectories corresponding to the same rigid motion can be described by a subspace of dimension $p = 4$. Given a set of

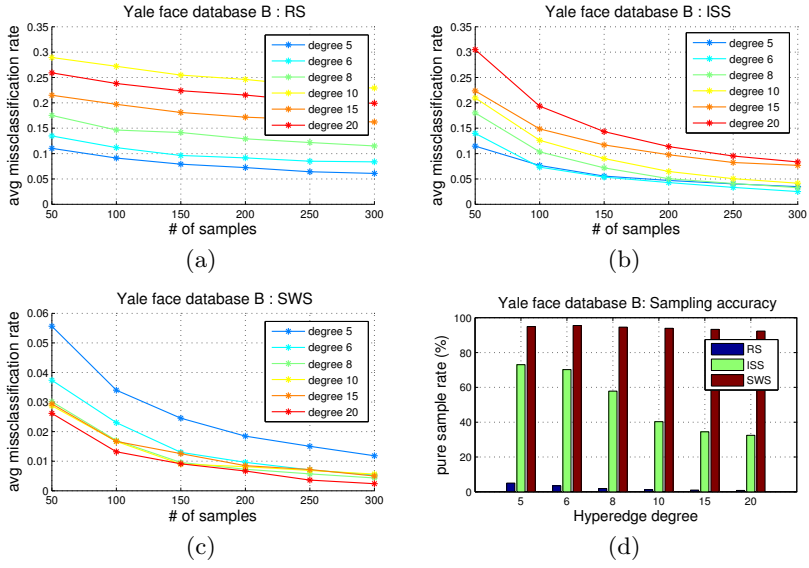


Fig. 2. (a)–(c) Misclassification rate of three methods on Yale Face Database B as hyperedge degree D is increased. (d) Sampling accuracy for different degree D.

trajectories V containing K distinct motions, we can separate the motions by subspace clustering. Here we used the Hopkins 155 dataset. The dataset consists of 155 video sequences with 2 or 3 independent rigid motions which are subcategorised into three types: checkerboard, traffic and articulated.

For our approach (SWS followed by NCut), we used $D = 10$ and 50 samples. To compute the auxiliary graph $G^{(0)}$ and edge weights (5), we reduce the dimension of the data in each sequence to $l = 4K$. For our method, across all sequences we observe that on average 98.34% the hyperedges generated are clean, i.e., contain trajectories from the same motion.

Table 2 presents the misclassification rates of proposed method along with the other recent methods for comparison. We also additionally compared with Local Subspace Affinity (LSA) [24] and the recent method of [25] which uses discrete cosine transform (DCT). Unlike our method, these state-of-the-art techniques are dedicated to motion segmentation. It is clear that our approach can achieve similar or better results than the other methods.

Comparing run time is nontrivial, since the methods are based on very different principles. Our approach typically takes less than 5 minutes to execute. The following subsection will examine the run time of our method more closely.

4.3 Motion segmentation with dense trajectories

Recently there is a surge of interest in conducting video-based object segmentation by clustering the dense feature trajectories $V = \{v_i\}_{i=1}^{N}$ on the objects. The

Table 1. Mean percentage of misclassification on clustering Yale face B dataset.

K	2	3	4	5	6	7	8	9	10	time(s)
GPCA	0.0	49.5	0.0	26.6	9.9	25.2	28.5	30.6	19.8	$\approx 10^6$
SCC	0.0	0.0	0.0	1.1	2.7	2.1	2.2	5.7	6.6	4.93
SSC	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.4	4.6	6.12
SLBF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.9	1.72
ALC	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1878.56
Ours	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.74

Table 2. Percent misclassification error on Hopkins 155 dataset.

	Two Motions						Three Motions							
Method	Chck.(78)		Trfc.(31)		Artc.(11)		Chck.(26)		Trfc.(7)		Artc.(2)		all(155)	
	MN	MD	MN	MD	MN	MD	MN	MD	MN	MD	MN	MD	MN	MD
GPCA	6.09	1.03	1.41	0.00	2.88	0.00	31.95	32.93	19.83	19.55	16.85	16.85	10.34	2.54
LSA	2.57	0.27	5.43	1.48	4.10	1.22	5.80	1.77	25.07	23.79	7.25	7.25	4.94	0.90
ALC	1.55	0.29	1.59	1.17	10.70	0.95	5.20	0.67	7.75	0.49	21.08	21.08	3.56	0.50
SCC	1.31	0.06	1.02	0.26	3.21	0.76	6.31	1.97	3.31	3.31	9.58	9.58	2.42	NA
SSC	1.12	0.00	0.02	0.00	0.62	0.00	2.97	0.27	0.58	0.00	1.42	1.42	1.24	0.00
DCT	0.71	0.00	0.05	0.00	0.96	0.00	2.44	1.29	0.05	0.00	1.60	1.60	0.87	NA
Ours	1.86	0.00	0.08	0.00	1.13	0.00	2.72	0.00	0.00	0.00	1.06	1.06	1.50	0.00
MN: mean, MD: median, NA: not available														

original proposal by Brox and Malik (BM) [26] achieves this via a conventional clustering approach, which constructs an affinity measure to compare pairs of trajectories. Using pairwise affinities restricts the motions to be 2D translations.

Ochs and Brox (OB) [7] later argued that the approach of [26] fails if the objects undertake more complex motions, which occur quite often in the videos used in [26]. Thus the affinity measure must include more than two trajectories to allow richer motion models. OB used 2D similarities (rotation, translation, and scaling) and posed motion segmentation as hypergraph clustering. Hyperedges of degree 3 are used (the hypergraph is 3-uniform). Recall that a minimum of two trajectories are needed to instantiate a 2D similarity ($p = 2$), thus $D = 3$ is the smallest possible hyperedge degree.

Even with such small hyperedges, enumerating them is not feasible since their number scales as $\mathcal{O}(|V|^3)$; recall that a short video can have thousands of dense trajectories. OB samples the hyperedges as follows: all $|V| \times |V|$ pairs of trajectories are generated. For each pair of trajectories, 12 hyperedges are created by including in the pair a third trajectory from the 12 nearest spatial neighbours of the pair. An additional 30 hyperedges are produced by randomly including a third trajectory. As an example, the *car1* sequence (which is one of the shortest in the dataset with 19 frames) contains 4850 trajectories. OB thus generates $4850 \times 4850 \times (30+12) = 987,945,000$ ($\approx 10^9$) hyperedges. To compute the weight of a hyperedge, the max-error fit is obtained as follows: all three pairs of trajectories in a hyperedge are used to instantiate 2D similarities which are

then evaluated with the third trajectory. The highest error is converted to an affinity value via a negative exponential. The massive number of hyperedges used contribute to significant computational expense. For example, OB requires 48 minutes to compute the affinity matrix of *car1*.

We demonstrate how our approach significantly improves the run time of hypergraph clustering for motion segmentation. Hyperedges of degree of $D = 10$ were used in our hypergraphs. Based on SWS, we generated 1000 samples of $(D-1)$-tuples from the trajectories V of a given sequence. A 2D similarity motion was fitted on each $(D-1)$-tuple and evaluated against the rest of the trajectories. There were thus a total of (just) $1000|V|$ hyperedges in each hypergraph. This reduces run time enormously. Moreover, as we will show later, since SWS can generate pure hyperedges accurately, these relatively few hyperedges were sufficient to produce good segmentation. Since we used large hyperedges, OB's max-error fit procedure which tests all pairs is too costly to be applied. We simply estimated the 2D similarity on each $(D-1)$-tuple using least squares. Under these settings, we computed the affinity matrix of *car1* within 10 seconds.

Due to the significant difference in computational requirements, any comparisons must thus take run time into account. To compare against OB, we reduce a given sequence by taking just the first-10 frames of the video. We ran the implementation of BM and OB[1] on a linux 64-bit desktop. Note that the provided binaries conduct dense feature tracking from scratch; approximately 4 minutes are needed for dense tracking over 10 frames. Comparing just the *clustering step*, OB takes ≈ 1 hour and BM takes ≈ 1 minute. Using the trajectories output by OB, our method requires a relatively tiny ≈ 15 seconds for clustering.

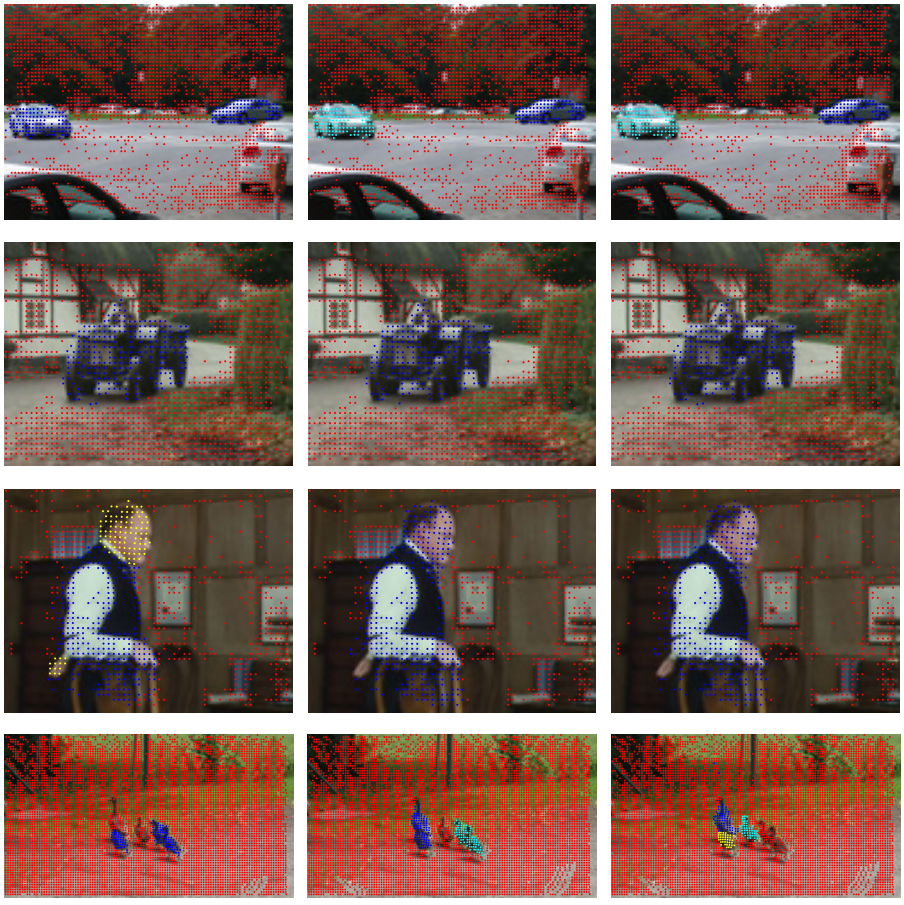
Fig. 3 provides qualitative comparisons on *car5*, *marple8*, *marple13* and *duck* sequences (all reduced to 10 frames). Despite the much smaller computational expense, our method provides similar segmentation quality as OB. Note also that BM fails in *duck* due to the more complex motions (as reported in [7]). Quantitative comparison based on the criteria in BM and OB is also made using the evaluation code provided with the dataset. See Table 3.

The reader may suggest that our speed is due to using least squares instead of max-error fit. Note that when applied to a subset of size 3, the max-error fit is similar with least squares in time, since there are only 3 pairs to test. Thus, the major factor in the computational burden of OB is the number of hyperedges.

Table 3. Results on Berkeley motion segmentation benchmark for first 10 frames.

	Density	overall error	average error	over segmentation	extracted objects
BM	0.81%	7.86%	28.76%	0.35	22
OB	0.79%	7.44%	28.01%	0.38	23
Ours	0.79%	8.05%	27.84%	0.23	22

[1] Available at http://http://lmb.informatik.uni-freiburg.de/Publications/2012/OB12/



(a) Brox and Malik [26]. (b) Ochs and Brox [7]. (c) Our method.

Fig. 3. Segmentation results in sample frames of sequences *car5*, *marple8*, *marple13* and *duck* from the Berkeley motion segmentation dataset [26].

5 Conclusion

We have established theoretically the benefits of using large hyperedges in hypergraph clustering. Our theoretical analysis is then supported by comprehensive experiments. In particular, the experimental results clearly show that using large hyperedges yields better clustering accuracy - this departs from previous methods that have exclusively used the smallest possible hyperedge. We have also proposed a novel algorithm for accurately sampling large hyperedges. Notwithstanding the usage of large hyperedges, our method is very efficient to compute.

Acknowledgment: This work was supported by ARC grant DP130102524.

References

1. Agarwal, S., Branson, K., Belongie, S.: Higher order learning with graphs. In: ICML 2006
2. Govindu, V.M.: A tensor decomposition for geometric grouping and segmentation. In: CVPR 2005
3. Agarwal, S., Lim, J., Zelnik-Manor, L., Perona, P., Kriegman, D., Belongie, S.: Beyond pairwise clustering. In: CVPR 2005
4. Chen, G., Lerman, G.: Spectral curvature clustering (scc). IJCV **81**(3) (2009) 317–330
5. Liu, H., Latecki, L., Yan, S.: Robust clustering as ensembles of affinity relations. In: NIPS 2010
6. Liu, H., Yan, S.: Efficient structure detection via random consensus graph. In: CVPR 2012
7. Ochs, P., Brox, T.: Higher order motion models and spectral clustering. In: CVPR 2012
8. Jain, S., Govindu, V.M.: Efficient higher-order clustering on the grassmann manifold. In: ICCV 2013
9. MacKay, D.J.C.: (extra) The Swendsen-Wang method. In: Information theory, inference, and learning algorithms. Cambridge University Press (2003)
10. Pham, T.T., Chin, T.J., Yu, J., Suter, D.: The random cluster model for robust geometric fitting. In: CVPR 2012
11. Alpert, C.J., Kahng, A.B.: Recent directions in netlist partitioning: a survey. Integration: the VLSI journal **19**(1–2) (1995) 1–81
12. Zhou, D., Huang, J., Schölkopf, B.: Learning with hypergraphs: Clustering, classification, and embedding. NIPS 2006
13. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE TPAMI **22**(8) (2000) 888–905
14. Bulo', S.R., Pelillo, M.: A game-theoretic approach to hypergraph clustering. IEEE TPAMI **35**(6) (2013) 1312–1327
15. Elhamifar, E., Vidal, R.: Sparse subspace clustering. In: CVPR 2009
16. Georghiades, A., Belhumeur, P., Kriegman, D.: From few to many: illumination cone models for face recognition under variable lighting and pose. IEEE TPAMI **23**(6) (2001) 643–660
17. Tron, R., Vidal, R.: A benchmark for the comparison of 3-d motion segmentation algorithms. In: CVPR 2007
18. Tordoff, B., Murray, D.: Guided sampling and consensus for motion estimation. In: ECCV 2002
19. Raguram, R., Frahm, J.M., Pollefeys, M.: A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus. In: ECCV 2008
20. Chin, T.J., Yu, J., Suter, D.: Accelerated hypothesis generation for multi-structure robust fitting. In: ECCV 2010
21. Vidal, R., Ma, Y., Sastry, S.: Generalized principal component analysis (gpca). IEEE TPAMI **27**(12) (2005) 1945–1959
22. Zhang, T., Szlam, A., Wang, Y., Lerman, G.: Hybrid linear modeling via local best-fit flats. IJCV **100**(3) (2012) 217–240
23. Ma, Y., Derksen, H., Hong, W., Wright, J.: Segmentation of multivariate mixed data via lossy data coding and compression. IEEE TPAMI **29**(9) (2007) 1546–1562

24. Yan, J., Pollefeys, M.: A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In: ECCV 2006
25. Shi, F., Zhou, Z., Xiao, J., Wu, W.: Robust trajectory clustering for motion segmentation. In: ICCV 2013
26. Brox, T., Malik, J.: Object segmentation by long term analysis of point trajectories. In: ECCV 2010