Learning Hierarchical Shape Segmentation and Labeling from Online Repositories

LI YI, Stanford University
LEONIDAS GUIBAS, Stanford University
AARON HERTZMANN, Adobe Research
VLADIMIR G. KIM, Adobe Research
HAO SU, Stanford University
ERSIN YUMER, Adobe Research

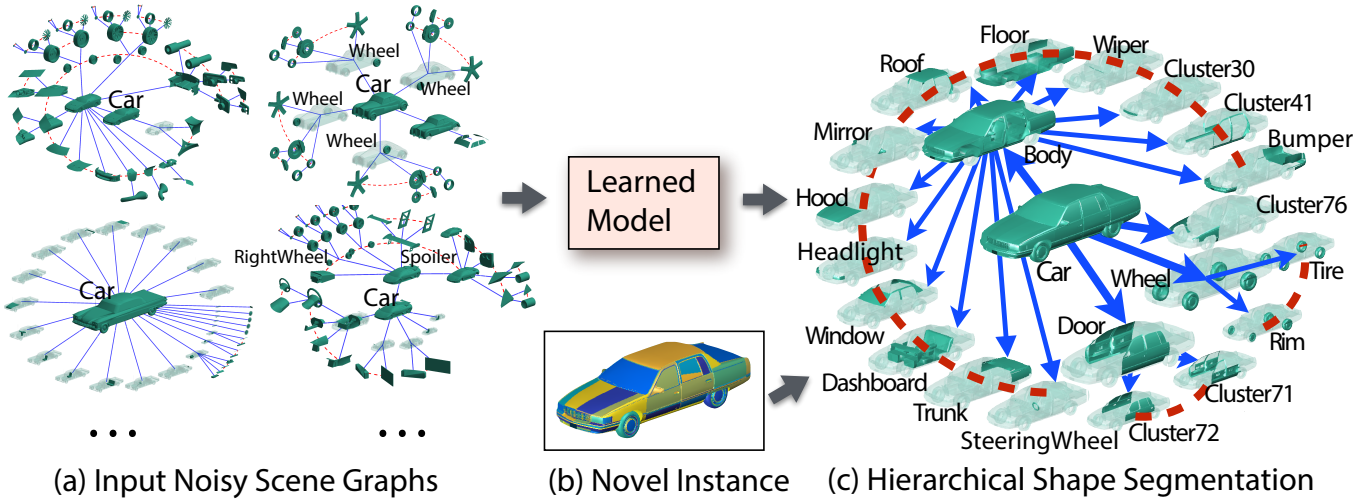

Fig. 1. Large online model repositories contain abundant additional data beyond 3D geometry, such as part labels and artist's part decompositions, flat or hierarchical. We tap into this trove of sparse and noisy noisy data to train a network for simultaneous hierarchical shape structure decomposition and labeling. Our method learns to take new geometry, and segment it into parts, label the parts, and place them in a hierarchy. In this paper, we visualize scene graphs with a circular visualization, in which the root node is near the center. Blue lines indicate parent-child relationships, and red dashed arcs connect siblings. The input geometry in online databases are broken as connected components, visualized in the input as random colors.

We propose a method for converting geometric shapes into hierarchically segmented parts with part labels. Our key idea is to train category-specific models from the scene graphs and part names that accompany 3D shapes in public repositories. These freely-available annotations represent an enormous, untapped source of information on geometry. However, because the models and corresponding scene graphs are created by a wide range of modelers with different levels of expertise, modeling tools, and objectives, these models have very inconsistent segmentations and hierarchies with sparse and noisy textual tags. Our method involves two analysis steps. First, we perform a joint optimization to simultaneously cluster and label parts in the database while also inferring a canonical tag dictionary and part hierarchy. We then use this labeled data to train a method for hierarchical segmentation and labeling of new 3D shapes. We demonstrate that our method can mine complex information, detecting hierarchies in man-made objects and their constituent parts, obtaining finer scale details than existing alternatives. We also show that, by performing domain transfer using a few supervised examples, our technique outperforms fully-supervised techniques that require hundreds of manually-labeled models.

CCS Concepts: •**Computing methodologies** → **Machine learning approaches; Shape analysis;**

Additional Key Words and Phrases: hierarchical shape structure, shape labeling, learning, Siamese networks

ACM Reference format:
Li Yi, Leonidas Guibas, Aaron Hertzmann, Vladimir G. Kim, Hao Su, and Ersin Yumer. 2017. Learning Hierarchical Shape Segmentation and Labeling from Online Repositories. *ACM Trans. Graph.* 36, 4, Article 70 (July 2017), 12 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073652

Li Yi co-developed and implemented the method; the other authors are in alphabetical order. This work started during Li Yi's internship at Adobe Research. This work is supported by NSF grants DMS-1521608 and DMS-1546206, ONR grant MURI N00014-13-1-0341, and Adobe Systems Inc.

DOI: http://dx.doi.org/10.1145/3072959.3073652

1 INTRODUCTION

Segmentation and labeling of 3D shapes is an important problem in geometry processing. These structural annotations are critical for many applications, such as animation, geometric modeling, manufacturing, and search (Mitra et al. 2013). Recent methods have shown that, by supervised training from labeled shape databases, state-of-the-art performance can be achieved on mesh segmentation and part labeling (Kalogerakis et al. 2010; Yi et al. 2016). However, such methods rely on carefully-annotated databases of shape segmentations, which is an extremely labor-intensive process. Moreover, these methods have used coarse segmentations into just a few parts each, and do not capture the fine-grained, hierarchical structure of many real-world objects. Capturing fine-scale part structure is very difficult with non-expert manual annotation; it is difficult even to determine the set of parts and labels to separate. Another option is to use unsupervised methods that work without annotations by analyzing geometric patterns (van Kaick et al. 2013). Unfortunately, these methods do not have access to the full semantics of shapes and as a result often do not identity parts that are meaningful to humans, nor can they apply language labels to models or their parts. Additionally, typical co-analysis techniques do not easily scale to large datasets.

We observe that, when creating 3D shapes, artists often provide a considerable amount of extra structure with the model. In particular, they separate parts into hierarchies represented as *scene graphs*, and annotate individual parts with textual names. In surveying the online geometry repositories, we find that most shapes are provided with these kinds of user annotations. Furthermore, there are often thousands of models per category available to train from. Hence, we ask: can we exploit this abundant and freely-available metadata to analyze and annotate new geometry?

Using these user-provided annotations comes with many challenges. For instance, Figure 1(a) shows four typical scene graphs in the car category, created by four different authors. Each one has a different part hierarchy and set of parts, e.g., only two of the scene graphs have the steering wheel of the car as a separate node. The hierarchies have different depths; some are nearly-flat hierarchies and some are more complex. Only a few parts are given names in each model. Despite this variability, inspecting these models reveal common trends, such as certain parts that are frequently segmented, parts that are frequently given consistent names, and pairs of parts that frequently occur in parent-child relationships with each other. For example, the tire is often a separate part, it is usually the child of the wheel, and usually has a name like tire or RightTire. Our goal is to exploit these trends, while being robust to the many variations in names and hierarchies that different model creators use.

This paper proposes to learn shape analysis from these messy, user-created datasets, thus leveraging the freely-available annotations provided by modelers. Our main goal is to automatically discover common trends in part segmentation, labeling, and hierarchy. Once learned, our method can be applied to new shapes that consist of geometry alone: the new shape is automatically segmented into parts, which are labeled and placed in a hierarchy. Our method can also be used to clean-up the existing databases. Our method is designed to work with large training sets, learning from thousands of models in a category. Because the annotations are uncurated, sparse (within each shape) and irregular, this problem is an instance of weakly-supervised learning.

Our approach handles each shape category (e.g., cars, airplanes, etc.) in a dataset separately. For a given shape category, we first identify the commonly-occurring part names within that class, and manually condense this set, combining synonyms, and removing uninformative names. We then perform an optimization that simultaneously (a) learns a metric for classifying parts, (b) assigns names to unnamed parts where possible, (c) clusters other unnamed parts, (d) learns a canonical hierarchy for parts in the class, and (e) provides a consistent labeling to all parts in the database. Given this annotation of the training data, we then learn to hierarchically segment new models, using a Markov Random Field (MRF) segmentation algorithm. Our algorithms are designed to scale to training on large datasets by mini-batch processing. We use these outputs to train a hierarchical segmentation model. Then, given a new, unsegmented mesh, we can apply this learned model to segment the mesh, transfer the tags, and infer the part hierarchy.

We use our method to analyze shapes from ShapeNet (Chang et al. 2015), a large-scale dataset of 3D models and part graphs obtained from online repositories. We demonstrate that our method can mine complex information detecting hierarchies in man-made objects and their constituent parts, obtaining finer scale details than existing alternatives. While our problem is different from what has been explored in previous research, we perform two types of quantitative evaluations. First, we evaluate different variants of our method by holding some tags out, and show that all terms in our objective function are important to obtain the final result. Second, we show that supervised learning techniques require hundreds of manually labeled models until they reach the quality of segmentation that we get without any explicit supervision. We publicly share our code and the processed datasets in order to encourage further research.[1]

2 RELATED WORK

Recent shape analysis techniques focus on extracting structure from large collections of 3D models (Xu et al. 2016). In this section we discuss recent work on detecting labeled parts and hierarchies in shape collections.

Shape Segmentation and Labeling. Given a sufficient number of training examples, it is possible to learn to segment and label novel geometries (Guo et al. 2015; Kalogerakis et al. 2010; Yumer et al. 2014). While supervised techniques achieve impressive accuracy, they require dense training data for each new shape category, which significantly limits their applicability. To decrease the cost of data collection, researchers have developed methods that rely on crowdsourcing (Chen et al. 2009), active learning (Wang et al. 2012), or both (Yi et al. 2016). However, this only decreases the cost of data collection, but does not eliminate it. Moreover, these methods have not demonstrated the ability to identify fine-grained model structure, or hierarchies. One can rely solely on consistency in part geometry to extract meaningful segments without supervision (Golovinskiy and Funkhouser 2009; Hu et al. 2012; Huang et al. 2011, 2014; Kim et al. 2013; Sidi et al. 2011). However, since these methods do not take any human input into account, they typically only detect coarse parts,

[1] http://cs.stanford.edu/~ericyi/project_page/hier_seg/index.html

and do not discover semantically salient regions where geometric cues fail to encapsulate the necessary discriminative information.

In contrast, we use the part graphs that accompany 3D models to weakly supervise the shape segmentation and labeling. This is similar in spirit to existing unsupervised approaches, but it mines semantic guidance from ambient data that accompanies most available 3D models.

Our method is an instance of weakly-supervised learning from data on the web. A number of related problems have been explored in computer vision, including learning classifiers and captions from user-provided images on the web, e.g., (Izadinia et al. 2015; Li et al. 2016; Ordonez et al. 2011), or image searches, e.g., (Chen and Gupta 2015).

Shape Hierarchies. Previous work attempted to infer scene graphs based on symmetry (Wang et al. 2011) or geometric matching (van Kaick et al. 2013). However, as with unsupervised segmentation techniques, these methods only succeed in a presence of strong geometric cues. To address this limitation, Liu et al. (2014) proposed a method that learns a probabilistic grammar from examples, and then uses it to create consistent scene graphs for unlabeled input. However, their method requires accurately labeled example scene graphs. Fisher et al. (2011) use scene graphs from online repositories, focusing on arrangements of objects in scenes, whereas we focus on fine-scale analysis of individual shapes.

In contrast, we leverage the scene graphs that exist for most shapes created by humans. Even though these scene graphs are noisy and contain few meaningful node names (Figure 1(a)), we show that it is possible to learn a consistent hierarchy by combining cues from corresponding sparse labels and similar geometric entities in a joint framework. Such label correspondences not only help our clusters be semantically meaningful, but also help us discover additional common nodes in the hierarchy.

3 OVERVIEW

Our goal is to learn an algorithm that, given a shape from a specific class (e.g., cars or airplanes), can segment the shape, label the parts, and place the parts into a hierarchy. Our approach is to train based on geometry downloaded from online model repositories. Each shape is composed of 3D geometry segmented into distinct parts; each part has an optional textual name, and the parts are placed in a hierarchy. The hierarchy for a single model is called a scene graph. As discussed above, different training models may be segmented in different hierarchies; our goal is to learn from trends in the data as to which parts are often segmented, how they are typically labeled, and which parts are typically children of other parts.

We break the analysis into two sub-tasks:

- **Part-Based Analysis** (Section 4). Given a set of meshes in a specific category and their original messy scene graphs, we identify the dictionary of distinct parts for a category, and place them into a canonical hierarchy. This dictionary includes both parts with user-provided names (e.g., `wheel`) and a clustering of unnamed parts. All parts on the training meshes are labeled according to the part dictionary.

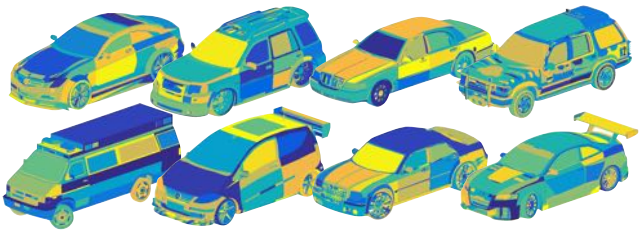

Fig. 2. A visualization of connected components in ShapeNet cars, illustrating that each connected component is usually a sub-region of a single part.

- **Hierarchical Mesh Segmentation** (Section 5). We train a method to segment a new mesh into a hierarchical segmentation, using the labels and hierarchy provided by the previous step. For parts with textual names, these labels are also transferred to the new parts.

We evaluate with testing on hold-out data, and qualitative evaluation. In addition, we show how to adapt our model to a benchmark dataset.

Our method makes two additional assumptions. First, our feature vector representations assume consistently-oriented meshes, following the representation in ShapeNetCore (Chang et al. 2015). Second, the canonical hierarchy requires that every type of part has only one possible parent label, e.g., our algorithm might infer that the parent of a `headlight` is always the `body`, if this is frequently the case in the training data.

In our segmentation algorithm, we usually assume that each connected component in the mesh belongs to a single part. This can be viewed as a form of over-segmentation assumption (e.g., (van Kaick et al. 2013)), and we found it to be generally true for our input data, e.g., see Figure 1(b) and 2. We show results both with and without this assumption in Section 6 and in the Supplemental Material.

4 PART-BASED ANALYSIS

The first step of our process takes the shapes in one category as input, and identifies a dictionary of parts for that category, a canonical hierarchy for the parts, and a labeling of the training meshes according to this part dictionary. Each input shape i is represented by a scene graph: a rooted directed tree $H_i = \{X_i, E_i\}$, where nodes are parts with geometric features $X_i = \{\mathbf{x}_{ij} | j = 1, ..., |X_i|\}$ and each edge $(j, k) \in E_i$ indicates that part (i, k) is a child of part (i, j). We manually pre-process the user-provided part names into a tag dictionary T, which is a list of part names relevant for the input category (Table 1). One could imagine discovering these names automatically. We opted for the manual processing, since the vocabulary of words that appear in ShapeNet part labels is fairly limited, and there are many irregularities in the label usage, e.g., synonyms and mispellings. The parts with a label from the dictionary are then assigned corresponding tags t_{ij}. Note that many parts are untagged, either because no names were provided with the model, or the user-provided names did not map onto names in the dictionary. Note also that j is indexes parts within a shape independent of tags; e.g., there

is no necessary relation between (i, j) and part (k, j). Each graph has a root node, which has a special root tag, and no parent. For non-leaf nodes, the geometry of any node is the union of geometries of its children.

To produce a dictionary of parts, we could directly use the user-provided tags, and then cluster the untagged parts. However, this naive approach would have several intertwined problems. First, the user-provided tags may be incorrect in various ways: missing tags for known parts (e.g., a wheel not tagged at all), tags given only at a high-level of the hierarchy (e.g., the rim and the tire are not segmented from the wheel, and they are all tagged as wheel), and tags that are simply wrong. The clustering itself depends on a distance metric, which must be learned from labels. We would like to have tags be applied as broadly and accurately as possible, to provide as much clean training data as possible for labeling and clustering, and to correctly transfer tags when possible. Finally, we would also like to use a parent-child relationships to constrain the part labeling (so that a wheel is not the child of a door), but plausible parent-child relationships are not known a priori.

We address these problems by jointly optimizing for all unknowns: the distance metric, a dictionary of D parts, a labeling of parts according to this dictionary, and a probability distribution over parent-child relationships. The labeling of model parts is also done probabilistically, by the Expectation-Maximization (EM) algorithm (Neal and Hinton 1998), where the hidden variables are the part labels. The distance metric is encoded in a embedding function $f : \mathbb{R}^S \to \mathbb{R}^F$, which maps a part represented by a shape descriptor $\mathbf{x}$ (Appendix A) to a lower-dimensional feature space. The function f is represented as a neural network (Figure 12). Each canonical part k has a representative cluster center $\mathbf{c}_k$ in the feature space, so that a new part can be classified by nearest-neighbors distance in the feature space. Note that the clusters do not have an explicit association with tags: our energy function only encourages parts with the same tag to fall in the same cluster. As a post-process, we match tag names to clusters where possible.

We model parent-child relationships with a matrix $\mathbf{M} \in \mathbb{R}^{D \times D}$, where $\mathbf{M}_{uv}$ is, for a part in cluster v, the probability that its parent has label u. After the learning stage, $\mathbf{M}$ is converted to a deterministic canonical hierarchy over all of the parts.

Our method is inspired in part by the semi-supervised clustering method of Basu et al. (2004). In contrast to their linear embedding of initial features for metric learning, we incorporate a neural network embedding procedure to allow non-linear embedding in the presence of constraints, and use an EM soft clustering. In addition, Basu et al. (2004) do not take hierarchical representations into consideration, whereas our data is inherently a hierarchical part tree.

4.1 Objective function

The EM objective function is:

$$E(\theta, \mathbf{p}, \mathbf{c}, \mathbf{M}) = \lambda_c E_c + \lambda_s E_s + \lambda_d E_d + \lambda_m E_m - H \tag{1}$$

where θ are the parameters of the embedding f, $\mathbf{p}$ are the label probabilities such that p_{ijk} represents the probability of the j^{th} part of i^{th} shape be assigned to k^{th} label cluster, and $\mathbf{c}_{1:D}$ are the unknown cluster centers. We set $\lambda_c = 0.1, \lambda_s = 1, \lambda_d = 1, \lambda_m = 0.05$ throughout all experiments.

The first two terms, E_c and E_s, encourage the concentration of clusters in the embedding space; E_d encourages the separation of visually dissimilar parts in embedding space; E_m is introduced to estimate the parent-child relationship matrix $\mathbf{M}$; the entropy term $H = -\sum p \ln p$ is a consequence of the derivation of the EM objective (Appendix **??**) and is required for correct estimation of probabilities. We next describe the energy terms one by one.

Our first term favors part embeddings to be near their corresponding cluster centroids:

$$E_c = \sum_{(i,j)} \sum_{k \in 1:D} p_{ijk} ||f(\mathbf{x}_{ij}) - \mathbf{c}_k||_1 \tag{2}$$

where f is the embedding function $f(\cdot)$, represented as a neural network and parametrized by a vector θ. The network is described in Appendix A.

Second, our objective function constrains the embedding, by favoring small distances for parts that share the same input tag, and for parts that have very similar geometry:

$$E_s = \sum_{(x_{ij}, x_{kl}) \in \mathbb{S}} ||f(\mathbf{x}_{ij}) - f(\mathbf{x}_{kl})||_1, \text{ where} \tag{3}$$
$$(x_{ij}, x_{kl}) \in \mathbb{S} \text{ iff } t_{ij} = t_{kl} \text{ or } \|\mathbf{x}_{ij} - \mathbf{x}_{kl}\|_2^2 \leq \delta$$

We extract all tagged parts and sample pairs from them for the constraint. We set $\delta = 0.1$ to a small constant to account for near-perfect repetitions of parts, and ensure that these parts are assigned to the same cluster.

Third, our objective favors separation in the embedded space by a margin σ_d between parts on the same shape that are not expected to have the same label:

$$E_d = \sum_{(x_{ij}, x_{il}) \in \mathbb{D}} \max(0, \sigma_d - ||f(\mathbf{x}_{ij}) - f(\mathbf{x}_{il})||_1), \text{ where} \tag{4}$$
$$(x_{ij}, x_{il}) \in \mathbb{D} \text{ if } (x_{ij}, x_{il}) \notin \mathbb{S}.$$

We only use parts from the same shape in $\mathbb{D}$, since we believe it is generally reasonable to assume that parts on the same shape with distinct tags or distinct geometry have distinct labels.

Finally, we score the labels of parent-child pairs by how well they match the overall parent-child label statistics in the data, using the negative log-likelihood of a multinomial:

$$E_m = -\sum_{\ell_1, \ell_2 \in 1:D \times 1:D} \sum_i \sum_{(j,k) \in E_i} p_{ij\ell_1} p_{ik\ell_2} \ln \mathbf{M}_{\ell_1 \ell_2} \tag{5}$$

4.2 Generalized EM algorithm

We optimize the objective function (Equation 1) by alternating between E and M steps. We solve for the soft labeling $\mathbf{p}$ in the E-step, and the other parameters, $\Theta = \{\theta, \mathbf{c}, \mathbf{M}\}$, in the M-step, where θ are the parameters of the embedding f.

E-step. Holding the model parameters Θ fixed, we optimize for the label probabilities $\mathbf{p}$:

$$\underset{\mathbf{p}}{\text{minimize}}\ \lambda_c E_c + \lambda_m E_m - H \tag{6}$$

We optimize this via coordinate descent, by iterating 5 times over all coordinates. The update is given in Appendix C.

M-step. Next, we hold the soft clustering $\mathbf{p}$ fixed and optimize the model parameters Θ by solving the following subproblem:

$$\underset{\Theta}{\text{minimize}} \quad \lambda_c E_c + \lambda_s E_s + \lambda_d E_d + \lambda_m E_m \tag{7}$$

We use stochastic gradient descent updates for θ and $\mathbf{c}_{1:D}$, as is standard for neural networks, while keeping $\mathbf{p}, \mathbf{M}$ fixed. The parent-child probabilities $\mathbf{M}$ are then computed as:

$$\mathbf{M} \leftarrow \text{normc}\left(\sum_i \sum_{(j,k)\in E_i} \mathbf{p}_{ij}\mathbf{p}_{ik}^T\right) + \epsilon \tag{8}$$

where normc$(\cdot)$ is a column-wise normalization function to guarantee $\sum_i \mathbf{M}_{ij} = 1$. $\mathbf{p}_{ij}$ and $\mathbf{p}_{ik}$ are the cluster probability vectors that correspond to parts x_{ij} and x_{ik} of the same shape, respectively. $\epsilon = 1 \times 10^{-6}$ in our experiments, to prevent cluster centers from stalling at zero. Since each column of $\mathbf{M}$ is a separate multinomial distribution, the update in Eq. 8 is the standard multinomial estimator.

Mini-batch training. The dataset for any category is far too large to fit in memory, and so, in practice, we break the learning process into mini-batches. Each mini-batch includes 50 geometric models at a time. For the set $\mathbb{S}$, 20,000 random pairs of parts are sampled across models in the mini-batch. 30 epochs (passes over the whole dataset) are used.

For each mini-batch, the E-step is computed as above. In the mini-batch M-step, the embedding parameters θ and cluster centers $\mathbf{c}$ are updated by standard stochastic gradient descent (SGD) updates, using Adam updates (Kingma and Ba 2015). For the hierarchy $\mathbf{M}$, we use Stochastic EM updates (Cappé and Moulines 2009), which are more stable and efficient than gradient updates. The sufficient statistics are computed for the minibatch:

$$\bar{\mathbf{M}}_{\text{mb}} = \sum_i \sum_{(j,k)\in E_i} \mathbf{p}_i \mathbf{p}_j^T \tag{9}$$

Running averages for the sufficient statistics are updated after each mini-batch:

$$\bar{\mathbf{M}} \leftarrow (1-\eta)\bar{M} + \eta\bar{M}_{\text{mb}} \tag{10}$$

where $\eta = 0.5$ in our experiments. Then, the estimates for $\mathbf{M}$ are computed from the current sufficient statistics by:

$$\mathbf{M} \leftarrow \text{normc}(\bar{\mathbf{M}}) + \epsilon \tag{11}$$

Initialization. Our objective, like many EM algorithms, requires good initialization. We first initialize the neural network embedding $f(\cdot)$ with normalized initialization (Glorot and Bengio 2010). For each named tag t_i, we specify an initial cluster center $\mathbf{c}_i$ as the average of the embeddings of all the parts with that tag. The remaining D cluster centroids $\mathbf{c}_{|T|+1:D}$ are randomly sampled from a normal distribution in the embedding space. The cluster label probabiilities $\mathbf{p}$ are initialized by a nearest-neighbor hard-clustering, and then $\mathbf{M}$ is initialized by Eq. 8.

4.3 Outputs

Once the optimization is complete, we compute a canonical hierarchy $\mathcal{T}_M$ from $\mathbf{M}$ by solving a Directed Minimum Spanning Tree problem, with the root constrained to the entire object. Then, we assign tags to parts in the hierarchy by solving a linear assignment

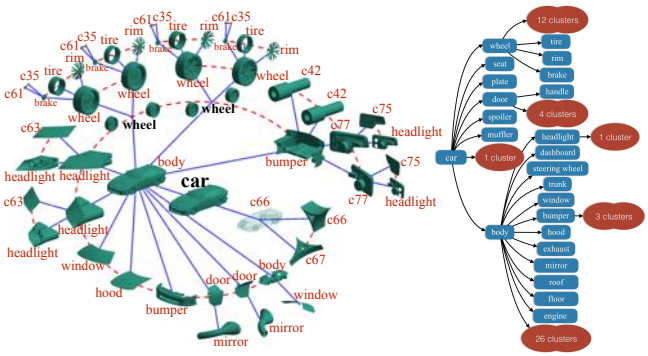

Fig. 3. Typical output of our part based analysis. Left: Part labeling for a training shape. Black labels indicate labels given with the input, and red labels were applied by our algorithm. Right: Canonical hierarchy. Generic cluster labels indicate newly discovered parts. Multiple generic clusters are grouped in the visualization, for brevity.

problem that maximizes the number of input tags in each cluster that agree with the tag assigned to their cluster. As a result, some parts in the canonical hierarchy receive textual names from assigned tags. Unmatched clusters are denoted with generic names `cluster_N`. We then label each input part (i, j) with its most likely node in $\mathcal{T}_M$ by selecting $\arg\max_k p_{ijk}$. This gives a part labeling of each node in each input scene graph. An example of the canonical hierarchy with part names, and a labeled shape, is shown in Figure 3.

This canonical hierarchy, part dictionary, and part labels for the input scene graphs are then used to train the segmentation algorithm as described in the next section.

5 HIERARCHICAL MESH SEGMENTATION

Given the part dictionary, canonical hierarchy, and per-part labels from the previous section, we next learn to hierarchically segment and label new shapes. We formulate the problem as labeling each mesh face with one of the leaf labels from the canonical hierarchy. Because each part label has only one possible parent, all of a leaf node's ancestors are unambiguous. In other words, once the leaf nodes are specified, it is straightforward to completely convert the shape into a scene graph, with all the nodes in the graph labeled. In order to demonstrate our approach in full generality, we assume the input shape includes only geometry, and no scene graph or part annotations. However, it should be possible to augment our procedure when such information is available.

5.1 Unary classifier

We begin by describing a technique for training a classifier for individual faces. This classifier can also be used to classify connected components. In the next section, we build an MRF labeler from this. Our approach is based on the method of Kalogerakis et al. (2010), but generalized to handle missing leaf labels and connected components, and to use neural network classifiers.

The face classifier is formulated as a neural network that takes geometric features of a face as input, and assigns scores to the leaf node labels for the face. The feature vector $\mathbf{y}$ for a face consists of several standard geometric features. The neural network specifies a

score function $\mathbf{w}_i^T g(\mathbf{y})$, where $\mathbf{w}_i$ is a weight vector for label L_i, and $g(\cdot)$ is a sequence of fully-connected layers and non-linear activation units, applied to $\mathbf{y}$. The score function is normalized by a softmax function to produce an output probability:

$$P_{face}(L_i|\mathbf{y}) = \frac{\exp\left(\mathbf{w}_i^T g(\mathbf{y})\right)}{\sum_{j\in\mathcal{D}} \exp\left(\mathbf{w}_j^T g(\mathbf{y})\right)} \tag{12}$$

where $\mathcal{D}$ is the set of possible leaf node labels. See Appendix B for details of the feature vector and neural network.

To train this classifier, we can apply the per-part labels from the previous section to the individual faces. However, there is one problem with doing so: many training meshes are not segmented to the finest possible detail. For example, a car wheel might not be segmented into tire and rim, or the windows may not be segmented from the body. In this case, the leaf node labels are not given for each face, but only ancestor nodes are known: we do not know which wheel faces are tire faces. In order to handle this, we introduce a probability table $A(a, b)$. $A(a, b)$ is the probability of a face taking leaf label $a \in \mathcal{D}$ if the deepest label given for this training face is $b \in \mathcal{L}$. For example, $A(\text{tire}, \text{wheel})$ is the probability that the correct leaf label for a face labeled as a wheel is tire. To estimate $A(a, b)$, we first compute the unnormalized $\hat{A}(a, b)$ by counting the number of faces assigned to both label a and label b, except that $\hat{A}(a, b) = 0$ if b is not an ancestor of a in the canonical hierarchy. Then A is determined by normalizing the columns to A to sum to 1: $A = \text{normc}(\hat{A})$.

We then train the classifier by minimizing the following loss function for $\mathbf{w}_{1:|\mathcal{D}|}$ and θ_g, the parameters of $g(\cdot)$:

$$E(\theta_g, \mathbf{w}_{1:|\mathcal{D}|}) = -\sum_k \log\left(\sum_{i\in\mathcal{D}} A(i, b_k) P(L_i|\mathbf{y}_k)\right) \tag{13}$$

where k sums over all faces in the training shapes and b_k is the deepest label assigned to face k as discussed above. This loss is the negative log-likelihood of the training data, marginalizing over the hidden true leaf label for each training face, generalizing (Izadinia et al. 2015). We use Stochastic Gradient Descent to minimize this objective.

We have also observed that meshes in online repositories are comprised of connected components, and these connected components almost always have the same label for the entire component. For most results presented in this paper, we use connected components as the basic labeling units instead of faces, in order to improve results and speed. We define the connected component classifier by aggregating the trained face classifier over all the faces $\mathcal{F}$ of the connected component as follows:

$$P_{CC}(L_i|\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{k\in\mathcal{F}} P_{face}(L_i|\mathbf{y}_k) \tag{14}$$

5.2 MRF labeler

Let $\mathcal{D}$ be the set of leaf node of the canonical hierarchy. In the case of classifying each connected component, we want to specify one leaf node $L_c \in \mathcal{D}$ for each connected component c. We define the MRF over connected component labels as:

$$E(L) = \sum_c \psi_{\text{unary}}(L_c) + \lambda \sum_{u,v\in\mathbb{E}} \psi_{\text{edge}}(L_u, L_v) \tag{15}$$

with weight λ is set by cross-validation separately for each shape category and held constant across all experiments. The unary term ψ_{unary} assesses the likelihood of component c having a given leaf label, based on geometric features of the component, and is given by the classifier:

$$\psi_{\text{unary}}(L) = -\ln P_{CC}(L|\mathcal{F}) \tag{16}$$

The edge term prefers adjacent components to have the same label. It is defined as $\psi_{\text{edge}}(L_u, L_v) = \text{td}(u, v)$, where $\text{td}(L_u, L_v)$ is tree distance between labels L_u and L_v in the canonical hierarchy. This encourages adjacent labels to be as close in the canonical hierarchy as possible. For example, ψ is 0 when the two labels are the same, whereas ψ is 2 if they are different but share a common parent. To generate the edge set $\mathbb{E}$ in 15, we connect K nearest connected components with this edge, where $K = \min(30, \lceil 0.01 N_{\text{cc}} \rceil)$ where N_{cc} is the number of connected components in the mesh.

Once the classifiers and λ are trained, the model can be applied to a new mesh as follows. First, the leaf labels are determined by optimizing Equation 15 using the α-β swap algorithm (Boykov et al. 2001). Then, the scene graph is computed by bottom-up grouping. In particular, adjacent components with same leaf label are first grouped together. Then, adjacent groups with the same parent are grouped at the next level of the hierarchy, and so on.

For the case where connected components are not available, the MRF algorithm is applied for each face. The unary term is given by the face classifier $\psi_{\text{unary}}(L) = -\ln P_{face}(L|\mathcal{F})$. We still need to handle the case where the object is not topologically connected, and so the pairwise term $\psi_{\text{edge}}(L_u, L_v)$ applies to all faces u and v whose centroids fall into the K-nearest neighborhood of each other, and is given by:

$$\psi_{\text{edge}}(L_u, L_v) = \lambda_t \exp\left(-\kappa_1 \frac{2\phi_{u,v}}{\pi} - \frac{d_{u,v}^2}{2(\kappa_2 d_r)^2}\right) \text{td}(L_u, L_v) \tag{17}$$

where $\phi_{u,v}$ is the angle between the faces, $d_{u,v}$ is the distance between the face centroids, d_r is the average distance between a face's centroid and it's nearest face's centroid, and $\kappa_1 = 5, \kappa_2 = 2.5$ in all our experiments. λ_t is a scale factor to promote faces sharing an edge: $\lambda_t = \begin{cases} 10 & \text{if faces } (u, v) \text{ share an edge,} \\ 1 & \text{otherwise.} \end{cases}$

6 RESULTS

In this section we evaluate our method on "in the wild" data from public online repositories and on a standard benchmark. We perform the evaluation by comparing with part-based analysis and segmentation techniques using novel metrics.

Input Data. We run our method on 9 shape categories from ShapeNetCore dataset (Chang et al. 2015), a collection of 3D models obtained from various online repositories. We use this dataset for convenience, because the data has been preprocessed, cleaned, categorized, and put into common formats; at present, it is the only known current dataset that satisfies our low-level preprocessing requirements. We excluded most categories (~40) because they only have a few

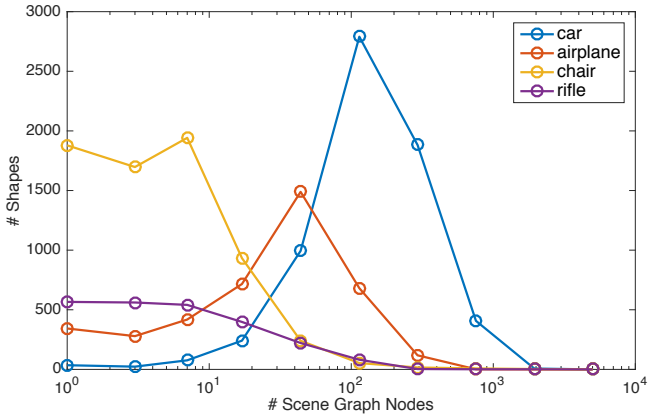

Fig. 4. Histogram of number of scene nodes for each shape in the raw datasets.

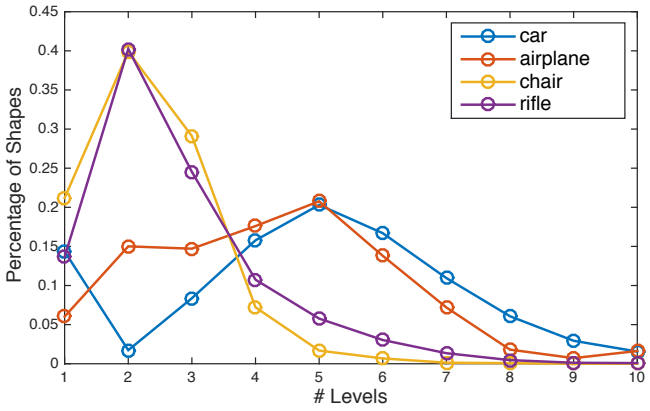

Fig. 5. Histogram of number of levels in hierarchy for each shape in the raw datasets.

hundred shapes or less, which is inadequate for our approach. We assume that a tag is sufficiently represented if it appears in at least 25 shapes, and we only analyze categories that have more than 2 such tags. Some categories have trivial geometry (e.g., mobile phones). Some categories do not provide enough parts with common labels (e.g., watercraft are very heterogeneous to the point of being disjoint sets of objects). The ShapeNetCore dataset currently contains a small subset of the available online repositories, which limits the data that we have immediately at hand. However, ShapeNetCore is growing; applying our method to much larger datasets is limited only by the nuisance of preprocessing heterogeneous datasets.

Typical scene graphs in such online repositories are very diverse, including between one and thousands of nodes (Figure 4), and ranging from flat to deep hierarchies (Figure 5). For each category, we also prescribe a list of relevant tags and possible synonyms. We automatically create a list of most-used tags for the entire category, and then manually pick relevant English nouns as the tag dictionary. Note that only a fraction of shapes have any parts with the chosen tags, and the frequency distribution over tag names is very uneven (Table 1, **Init** column).

For a categories with ~2000 shapes, the part-based analysis takes approximately one hour, and the segmentation training takes approximately 10 hours, each on a single Titan X GPU. Once trained, analysis of a new shape typically takes about 25 seconds, of which about 15 seconds is extracting face features with non-optimized Matlab code.

Hierarchical Mesh Segmentation and Labeling. Figure 10 demonstrates some representative results produced by our hierarchical segmentation based on connected components (Section 5). The resulting hierarchical segmentation vary in depth from flat (e.g., chairs) to deep (e.g., cars, airplanes), reflecting complexity of the corresponding object. We also often extract consistent part clusters, even if they do not have textual tags. We found that analyzing shapes at the granularity of connected components is usually sufficient: the mean number of connected components per object in ShapeNet is 4169, and the largest connected component in shapes covers only 9.58% of the total surface area on average: connected components tend to be small. These components are often aligned to part boundaries, for example, if one was to annotate ShapeNet Segmentation benchmark (Yi et al. 2016) by assigning a majority label to each connected component they would get 94% of faces correct.

Segmentation without Connected Components. In the case of applying per-face labeling, when connected components are not available, we observe similar results with this method as to those where the connected components are used. However, a few segments do not come out as cleanly-segmented on more complex models (see Figure 11). Please refer to our supplementary material for qualitative results of this experiment. We tested our method on other datasets (Thingi10k (Zhou and Jacobson 2016), COSEG (Wang et al. 2012)), but were only able to test on a limited set of models, since only a few models in these datasets come from our training categories.

Tag prediction. Table 1, *Final* column shows what fraction of training shapes received a particular tag after our part-based analysis (Section 4). Note that an object may be missing a tag for several possible reasons: it could be misclassified, or because the object does not have that part, or does not have it segmented as a separate scene graph node. As evident from this quantitative analysis, the amount of training data we can use in subsequent analysis has drastically increased. Please refer to supplementary material for visual examples from labeling results.

To evaluate tag prediction accuracy, we perform the following experiment. We hold out 30% of the tagged parts during training, and evaluate labeling accuracy on these parts. As our method is based on nearest-neighbors (NN) classification, we compare against NN on features computed in the following ways: (1) clustering with LFD, (2) clustering with $\mathbf{x}$, (3) our method with no E_c term, and (4) No E_m term. Results are reported in Table 3. As shown in the table, our method significantly improves tag classification performance over the baselines. This experiment also demonstrates the value of our clustering and hierarchy terms E_c and E_m.

Cluster evaluation. Figure 6 (bottom) demonstrates some parts grouped by our method in the part-based analysis (Section 4). We also note that some clusters combine unrelated parts, and we believe that they serve as null clusters for outliers.

Table 1. Tags and their frequency (percent of shapes that have a scene node (i.e., part) labeled with the corresponding tag) in the raw datasets and after our part based processing.

Part	Init	Final	Part	Init	Final	Part	Init	Final
Category: **Car** (2287 shapes)								
Wheel	17.4	96.4	Mirror	6.9	68.2	Window	9.5	71.8
Fender	1.1	40.4	Bumper	15.2	63.0	Roof	2.4	48.2
Exhaust	7.2	57.5	Floor	2.6	49.9	Trunk	4.5	60.6
Door	19.9	67.8	Spoiler	2.8	41.3	Rim	4.1	74.1
Headlight	14.6	61.9	Hood	12.2	68.7	Tire	5.3	33.5
Category: **Airplane** (2574 shapes)								
Wing	5.9	86.9	Engine	5.5	81.6	Body	2.3	86.8
Tail	1.5	90.5	Missile	0.4	66.7			
Category: **Chair** (2401 shapes)								
Arm	1.5	62.4	Leg	3.6	71.0	Back	0.9	50.8
Seat	2.4	83.1	Wheel	1.4	34.6			
Category: **Table** (2355 shapes)								
Top	1.9	81.5	Leg	6.4	85.8			
Category: **Sofa** (1243 shapes)								
BackPillow	4.7	79.3	Seat	2.7	63.0	Feet	3.1	41.5
Category: **Rifle** (994 shapes)								
Barrel	1.2	62.9	Bipod	1.7	47.6	Scope	3.2	66.7
Stock	3.2	56.0						
Category: **Bus** (713 shapes)								
Seat	4.8	47.0	Wheel	8.3	88.9	Mirror	1.7	42.4
Category: **Guitar** (491 shapes)								
Neck	3.1	75.8	Body	0.8	67.6			
Category: **Motorbike** (281 shapes)								
Seat	23.5	49.2	Engine	21.7	84.0	Gastank	14.6	73.7
Exhaust	1.8	71.5	Handle	8.9	75.8	Wheel	41.6	98.9

As we do not have ground truth for the unlabeled clusters, we instead evaluate the ability of our learned embedding to cluster parts, with the user-provided labels can serve as ground truth. We split the dataset for a category into *training tags* and *test tags*. We run the part-based analysis on all shapes, but provide only the training subset of tags to the algorithm. This gives an embedding f, and we can evaluate how good f is for clustering. This is done by running k-means clustering on the parts that correspond to the test tags, with k set to the number of test tags. The clustering result is then compared to the test tag labeling by normalized Mutual Information. This process is repeated in a 3-fold cross-validation. The baseline scores are especially low on categories with few parts, like Bus and Table. Table 2 shows quantitative results; our method performs significantly better than the baselines, including using k-means on Light Field Descriptors (LFD), and omitting the clustering term E_c from the objective.

Comparison to Unsupervised Co-Hierarchy. Van Kaick et al. (2013) propose an unsupervised approach for establishing consistent hierarchies within an object category. Their method was developed for small shape collections and requires hours of computation for 20 models, which makes it unsuitable for ShapeNet data. On the other hand, since we assume that some segments have textual tags, we also cannot run our method on their data. Given these constraints, we show a qualitative comparison to their method. In particular, we picked the first car and first airplane in their dataset, and retrieved

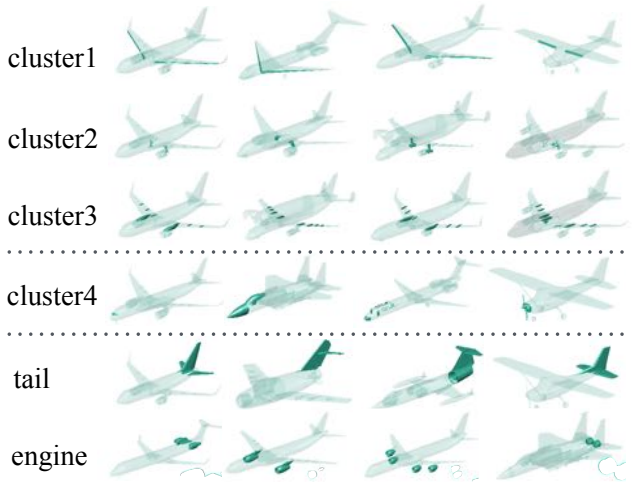

Fig. 6. Visualization of typical clusters. Note that some clusters have labels that were propagated from the tags, whereas some have generic labels indicating that they were discovered without any tag information.

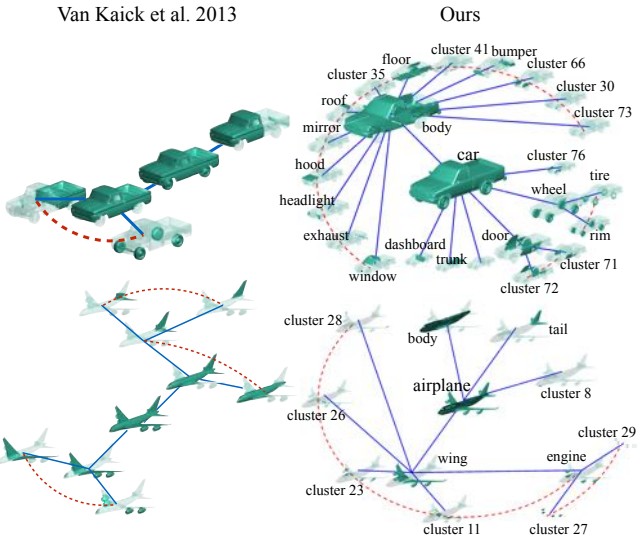

Fig. 7. Comparison with (van Kaick et al. 2013). We show a hierarchy produced by their approach (left) and a hierarchy of the most similar model in our database (right). Our hierarchies have labels and provide finer details.

the most similar models in ShapeNet using lightfield descriptors. Figure 7 demonstrates their and our hierarchies side-by-side. Note that our method generates more detailed hierarchies and also provides textual tags for parts.

Comparison to Supervised Segmentation. Since there are no large-scale hierarchical segmentation benchmarks, we test our method on the segmentation dataset provided by Yi et al. (2016). We emphasize that the benchmark contains much coarser segmentations than those we can produce, and does not include hierarchies. We take the intersection of our 9 categories and the benchmark, which yields the following six categories for quantitative evaluation: car, airplane, motorbike, guitar, chair, table.

Since other techniques do not leverage connected components, we evaluate per-face classification from unary terms only, comparing the per-face classification prediction (Eq. 14) to results from Yi et al. (2016) trained only on benchmark data.

Our training data is sampled from a different data distribution than the benchmark; repurposing a model from one training set to another is a problem known as *domain adaptation*. The first approach we test is to directly map the labels predicted by our classifier to benchmark labels. The second approach is to obtain 5 training examples from the benchmark, and train a Support Vector Machine classifier to predict benchmark labels from our learned features $\{g(\mathbf{x})\}$ (Sec. 4). The resulting classifier is the softmax of $\{\eta_i g(\mathbf{x})\}$, where η_i are the SVM parameters for i^{th} label. As baseline features, we also test k-means clustering with LFD features over all input parts, where k is the same as the number of clusters used by our method.

Results of supervised segmentation comparison experiments are shown in Figure 8. Without training on our features, the method of Yi et al. (2016) requires 50-100 benchmark training examples in order to match the results we get with only 5 benchmark examples. Although our method is trained on many ShapeNet meshes, these meshes did not require any manual labeling. This illustrates how our method, trained on freely-available data, can be cheaply adapted to a new task.

Figure 9 shows qualitative results from comparison with Yi et al. (2016), where we use 10 models for training in (Yi et al. 2016) followed by the domain adaptation through using the same 10 models in our approach.

7 DISCUSSIONS AND CONCLUSION

We have proposed a novel method for mining consistent hierarchical shape models from massive but sparsely annotated scene graphs "in the wild." As we analyze the input data, we jointly embed parts to a low-dimensional feature space, cluster corresponding parts, and build a probabilistic model for hierarchical relationships among them. We demonstrated that our model can facilitate hierarchical mesh segmentation and were able to extract complex hierarchies and identify small segments in 3D models from various shape categories. Our method can also provide a valuable boost for supervised segmentation algorithms. The goal of our current framework is to extract as much structure as possible from raw noisy, sparsely tagged scene graphs that exist in online repositories. In the future, we believe that using such freely-available information will provide enormous opportunities for shape analysis.

Developing Convolutional Neural Networks for surfaces is a very active area right now, e.g., (Guo et al. 2015). Our segmentation training loss functions are largely agnostic to the model representation, and it ought to be straightforward to train a ConvNet on our training loss, for any ConvNet that handles disconnected components.

Though effective as evidenced by experimental evaluations, several issues are not completely addressed yet. Our model currently relies on heuristic selection of the number of clusters k, and this could be chosen automatically. We could also relax the assumption that each part with a given label may have only one possible parent label, to allow more general shape grammars (Talton et al. 2012).

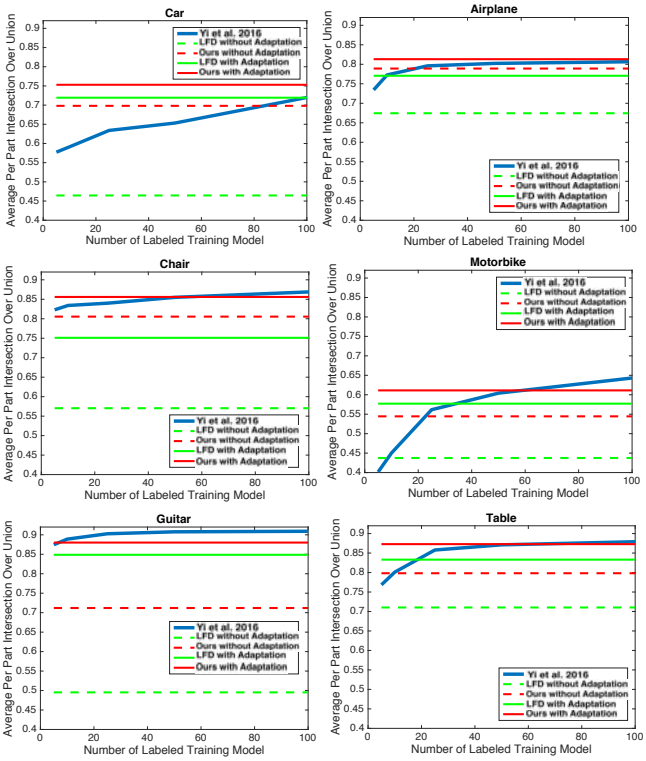

Fig. 8. Comparison with (Yi et al. 2016). Segmentation accuracy scores are shown; higher is better. The blue curves show the results of Yi et al. as a function of training set size. The red dashed lines show the result of our method without applying domain adaption, and the red solid lines show our method with domain adaptation by training on 5 benchmark models. For the more complex classes, Yi et al.'s method requires 50-100 training meshes to match our performance with only 5 benchmark training meshes.

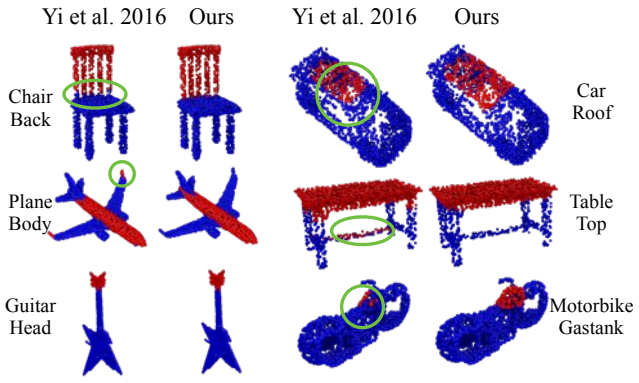

Fig. 9. Qualitative comparison with (Yi et al. 2016). For a fair comparison, we use 10 models for training in (Yi et al. 2016) and we use the same 10 models for domain adaptation in our approach.

Our method has obtained about 13K 3D training models with roughly consistent segmentation, but these have not been human-verified. We also believe that our approach could be leveraged together with crowdsourcing techniques (Yi et al. 2016) to efficiently yield very large, detailed, segmented, and verified shape databases.

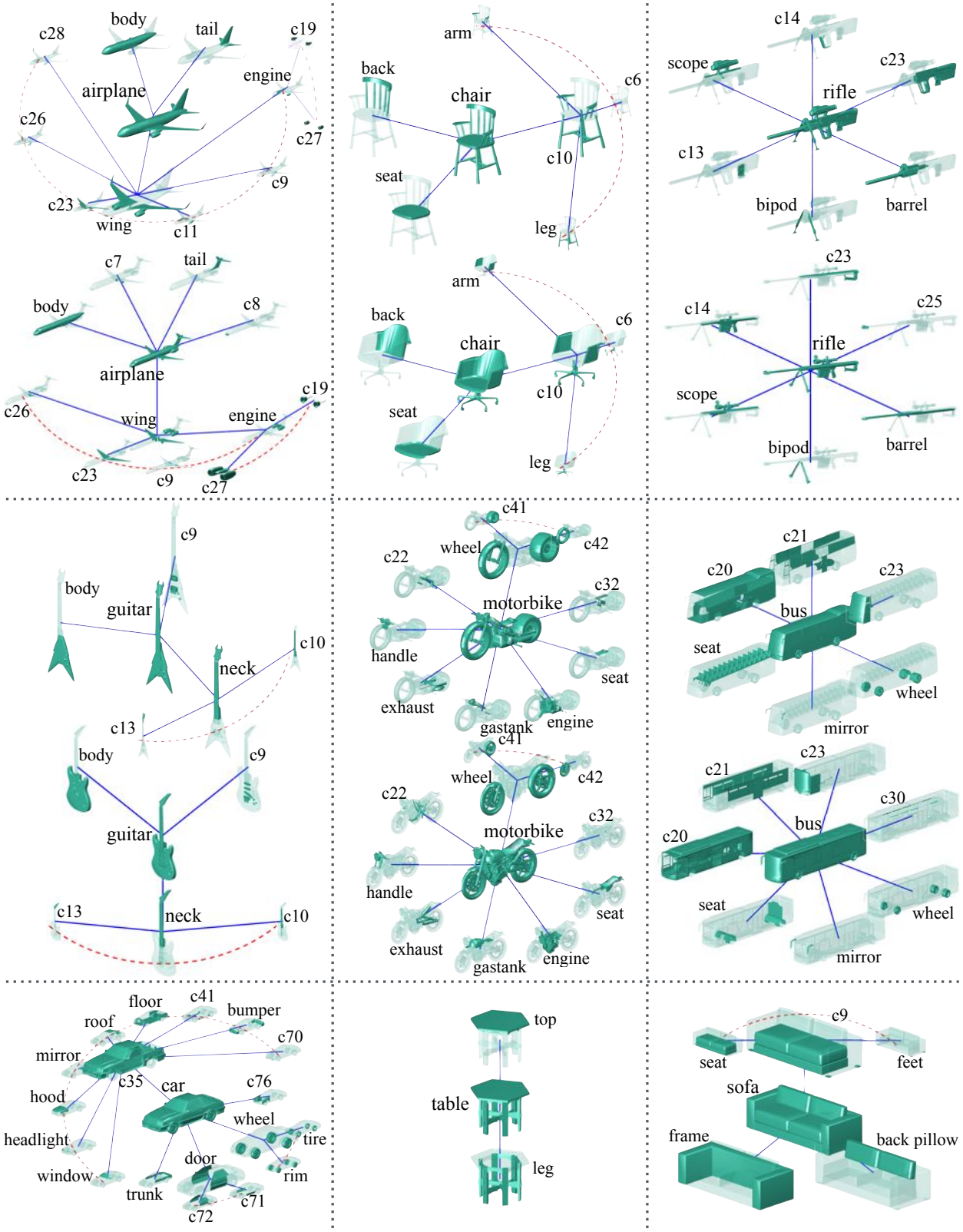

Fig. 10. Hierarchical segmentation results. In each case, the input is a geometric shape. Our method automatically determines the segmentation into parts, the part labels and the hierarchy.

Table 2. Clustering evaluation with different embedding/features. Scores shown are Normalized Mutual Information between the estimated clusters and the user-provided tags. A perfect score corresponds to NMI of 1. Note that the user-provided tags may themselves be noisy, so perfect scores are very unlikely.

Category	Mean	Car	Airplane	Chair	Table	Motorbike	Bus	Guitar	Rifle	Sofa
Chance	0.034	0.019	0.010	0.040	0.005	0.020	0.026	0.018	0.176	0.032
LFD	0.336	0.521	0.315	0.350	0.238	0.576	0.292	0.034	0.379	0.297
x (part features - App. A)	0.348	0.551	0.264	0.352	0.238	0.607	0.313	0.101	0.405	0.297
No E_c term	0.406	0.626	0.377	0.346	0.124	0.564	0.260	0.498	0.445	0.408
No E_m term	0.561	0.695	0.568	**0.622**	0.445	0.659	0.367	0.655	0.514	0.566
Ours	**0.573**	**0.712**	**0.575**	0.619	**0.448**	**0.678**	**0.371**	**0.655**	**0.526**	**0.571**

Table 3. Part tagging accuracy comparison. We hold out tags for 30% originally tagged parts in the input data, and report testing accuracy on the held out set.

Category	Mean	Car	Airplane	Chair	Table	Motorbike	Bus	Guitar	Rifle	Sofa
Chance	0.139	0.044	0.136	0.172	0.148	0.149	0.100	0.252	0.092	0.162
LFD	0.790	0.530	0.823	0.775	0.745	0.829	0.813	0.976	0.723	0.892
x (part features - App. A)	0.823	0.584	0.832	0.812	0.772	0.874	0.822	0.976	0.818	0.920
No E_c term	0.840	0.694	0.821	0.749	0.910	0.860	0.911	0.982	0.772	0.864
No E_m term	0.899	0.701	0.934	0.902	**0.926**	0.865	0.884	0.991	0.936	0.953
Ours	**0.910**	**0.709**	**0.97**	**0.905**	0.921	**0.878**	**0.884**	**0.994**	**0.951**	**0.979**

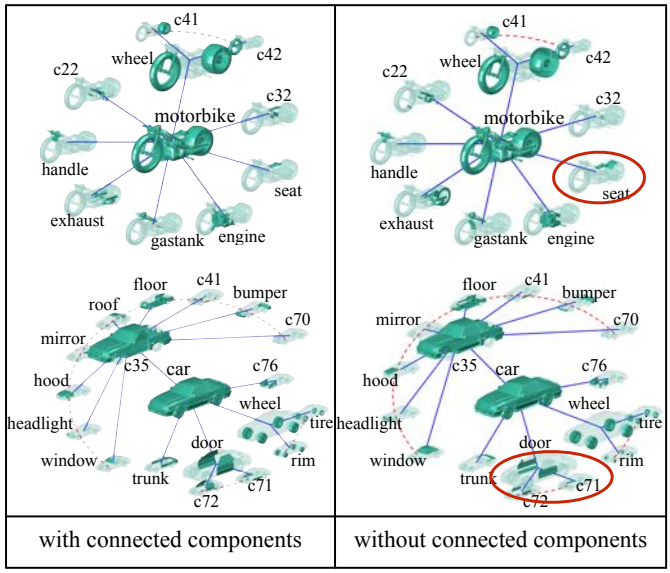

Fig. 11. Hierarchical segmentation results with and without the connected components assumption. Even without connected components, our method estimates complex hierarchical structures. For some models, the boundaries are less precise (see red higlights). We provide a full comparison to Figure 10 in supplemental material.

It would also be interesting to explore how well the information learned from one object category may transfer to other object categories. For example, "wheel" can be found in "cars" and "motorbikes", sharing similar geometry and sub-structures. The observation provides the opportunity for not only the transfer of part embeddings but also the part relationships. With the growth of online model repositories, such transfer learning ability would be more important and relevant towards more efficient expanding of our current dataset.

REFERENCES

Sugato Basu, Mikhail Bilenko, and Raymond J Mooney. 2004. A probabilistic framework for semi-supervised clustering. In *Proc. KDD*.

Serge Belongie, Jitendra Malik, and Jan Puzicha. 2002. Shape Matching and Object Recognition Using Shape Contexts. *IEEE T-PAMI* 24, 24 (2002), 509–521.

Yuri Boykov, Olga Veksler, and Ramin Zabih. 2001. Fast approximate energy minimization via graph cuts. *IEEE Trans. PAMI* (2001).

Olivier Cappé and Eric Moulines. 2009. On-line expectation-maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71, 3 (2009), 593–613.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An Information-Rich 3D Model Repository. (2015). arXiv:1512.03012.

Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. 2003. On Visual Similarity Based 3D Model Retrieval. In *Computer Graphics Forum (Eurographics)*.

Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. 2009. A Benchmark for 3D Mesh Segmentation. In *ACM SIGGRAPH (SIGGRAPH)*. Article 73, 73:1–73:12 pages.

Xinlei Chen and Abhinav Gupta. 2015. Webly Supervised Learning of Convolutional Networks. In *Proc. ICCV*.

Matthew Fisher, Manolis Savva, and Pat Hanrahan. 2011. Characterizing structural relationships in scenes using graph kernels. In *ACM TOG*, Vol. 30. 34.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

Aleksey Golovinskiy and Thomas Funkhouser. 2009. Consistent Segmentation of 3D Models. *Proc. SMI* 33, 3 (2009), 262–269.

Kan Guo, Dongqing Zou, and Xiaowu Chen. 2015. 3D Mesh Labeling via Deep Convolutional Neural Networks. *ACM TOG* 35, 1, Article 3 (2015), 12 pages.

Ruizhen Hu, Lubin Fan, , and Ligang Liu. 2012. Co-segmentation of 3D shapes via subspace clustering. *SGP* 31, 5 (2012), 1703–1713.

Qixing Huang, Vladlen Koltun, and Leonidas Guibas. 2011. Joint shape segmentation with linear programming. In *ACM SIGGRAPH Asia*. 125:1–125:12.

Qixing Huang, Fan Wang, and Leonidas Guibas. 2014. Functional Map Networks for Analyzing and Exploring Large Shape Collections. *SIGGRAPH* 33, 4 (2014).

Hamid Izadinia, Bryan C. Russell, Ali Farhadi, Matthew D. Hoffman, and Aaron Hertzmann. 2015. Deep Classifiers from Image Tags in the Wild. In *Proc. Multimedia COMMONS*.

Andrew E. Johnson and Martial Hebert. 1999. Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. *IEEE T-PAMI* 21, 5 (1999), 433–449.

Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. 2010. Learning 3D mesh segmentation and labeling. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 102.

Vladimir G Kim, Wilmot Li, Niloy J Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and Thomas Funkhouser. 2013. Learning part-based templates from large collections of 3D shapes. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 70.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proc. ICLR*.

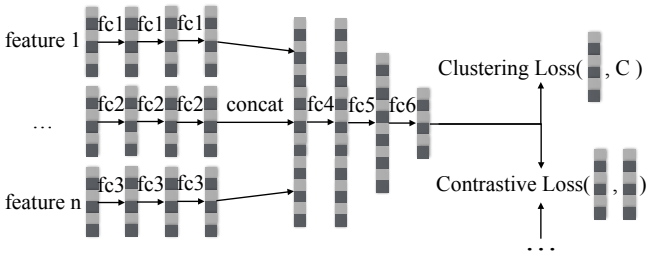

Fig. 12. Embedding network f architecture for parts. Different input part features go through a stack of fully connected layers and then are concatenated and go through additional fully connect layers to generate output of f. The contrastive loss also takes output of another part, from an identical embedding branch as input, which is omitted in this figure for brevity.

Xirong Li, Tiberio Uricchio, Lamberto Ballan, Marco Bertini, Cees G. M. Snoek, and Alberto Del Bimbo. 2016. Socializing the Semantic Gap: A Comparative Survey on Image Tag Assignment, Refinement, and Retrieval. *ACM Comput. Surv.* 49, 1 (2016).

Tianqiang Liu, Siddhartha Chaudhuri, Vladimir G. Kim, Qi-Xing Huang, Niloy J. Mitra, and Thomas Funkhouser. 2014. Creating Consistent Scene Graphs Using a Probabilistic Grammar. *SIGGRAPH Asia* 33, 6 (2014).

Niloy J Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, and Martin Bokeloh. 2013. Structure-aware shape processing. In *Eurographics STARs*. 175–197.

Radford M Neal and Geoffrey E Hinton. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*. Springer, 355–368.

Vicente Ordonez, Girish Kulkarni, and Tamara L. Berg. 2011. Im2text: Describing images using 1 million captioned photographs. In *Proc. NIPS*.

Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. 2002. Shape Distributions. *ACM Transactions on Graphics* (2002).

Oana Sidi, Oliver van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. 2011. Unsupervised Co-Segmentation of a Set of Shapes via Descriptor-Space Spectral Clustering. *ACM SIGGRAPH Asia* 30, 6 (2011), 126:1–126:9.

Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. 2012. Learning design patterns with bayesian grammar induction. In *UIST*.

Oliver van Kaick, Kai Xu, Hao Zhang, Yanzhen Wang, Shuyang Sun, Ariel Shamir, and Daniel Cohen-Or. 2013. Co-hierarchical analysis of shape structures. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 69.

Yunhai Wang, Shmulik Asafi, Oliver van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chenand. 2012. Active Co-Analysis of a Set of Shapes. *SIGGRAPH Asia* (2012).

Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiquan Cheng, and Yueshan Xiong. 2011. Symmetry Hierarchy of Man-Made Objects. *Eurographics* 30, 2 (2011).

Kai Xu, Vladimir G. Kim, Qixing Huang, Niloy J. Mitra, and Evangelos Kalogerakis. 2016. Data-Driven Shape Analysis and Processing. *SIGGRAPH Asia Course* (2016).

Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. 2016. A scalable active framework for region annotation in 3D shape collections. *TOG* 35, 6 (2016), 210.

Mehmet Ersin Yumer, Won Chun, and Ameesh Makadia. 2014. Co-segmentation of textured 3D shapes with sparse annotations. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 240–247.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. (2016). arxiv:1605.04797.

A PART FEATURES AND EMBEDDING NETWORK

We compute per-part geometric features which are further used for joint part embedding and clustering (Section 4). The feature vector $\mathbf{x}_{ij}$ includes 3-view lightfield descriptor (Chen et al. 2003) (with HOG features for each view), center-of-mass, bounding box diameter, approximate surface area (fraction of voxels occupied in 30x30x30 object grid), and local frame in PCA coordinate system (represented by 3×3 matrix M). To mitigate reflection ambiguities for local frame we constraint all frame axes to have positive dot product with z-axis (typically up) of the global frame. For lightfield descriptor we normalize the part to be centered at origin and have bounding box diameter 1, for all other descriptors we normalize the mesh in the same way. We mitigate reflection ambiguities by constraining all frame axes to have positive dot product with the z-axis of the global frame. The neural network embedding f is visualized in Figure 12, and, in Table 4, we show the embedding network parameters, where we alter first few fully connected layers to allocate more neurons for richer features such as LFD.

Table 4. Embedding network f output dimensionalities after each layer.

feature	fc1	fc2	fc3	concat	fc4	fc5	fc6
LFD	128	256	256	512	256	128	64
PCA Frame	16	32	64				
CoM	16	64	64				
Diameter	8	32	64				
Area	8	32	64				

Table 5. Face classification network parameters.

feature	fc1	fc2	fc3	concat	fc4	fc5	fc6
Curvature	32	64	64	640	256	128	128
LPCA	64	64	64				
LVar	32	64	64				
SI	128	128	128				
SC	128	128	128				
DD	32	64	64				
PP	16	32	64				
PN	16	32	64				

B FACE FEATURES AND CLASSIFIER NETWORK

We compute per-face geometric features $\mathbf{y}$ which are further used for hierarchical mesh segmentation (Section 5). These features include spin images (SI) (Johnson and Hebert 1999), shape context (SC) (Belongie et al. 2002), distance distribution (DD) (Osada et al. 2002), local PCA (LPCA) (where λ_i are eigenvalues of local coordinate system, and features are $\lambda_1/\sum\lambda_i, \lambda_2/\sum\lambda_i, \lambda_3/\sum\lambda_i, \lambda_2/\lambda_1, \lambda_3/\lambda_1, \lambda_3/\lambda_2$), local point position variance (LVar), curvature, point position (PP) and normal (PN). To compute local radius for the feature computation we sample 10000 points on the entire shape and use 50 nearest neighbors. We use the same architecture as part embedding network f (Fig. 12) for face classification, but with different loss function (Eq. 13) and network parameters, which are summarized in Table 5.

C E-STEP UPDATE

In the E-step, the assignment probabilities are iteratively updated. For each node (i, j), the probability that it is assigned to label k is updated as:

$$p^*_{ijk} \leftarrow \exp\Bigg(\lambda_m \sum_{a \in C(i,j),\ell} p_{ia\ell} \ln \mathbf{M}_{k\ell} + \lambda_m \sum_{b=P(i,j),\ell} p_{ib\ell} \ln \mathbf{M}_{\ell k} - \lambda_c ||f(\mathbf{x}_{ij} - \mathbf{c}_k||_1\Bigg) \tag{18}$$

$$p_{ijk} \leftarrow \frac{p^*_{ijk}}{\sum_\ell p^*_{ij\ell}} \tag{19}$$

where $C(i, j)$ is set of children of node (i, j) and $P(i, j)$ is the parent node. A joint closed-form update to all assignments could be computed using Belief Propagation, but we did not try this.