The Visual Computer
https://doi.org/10.1007/s00371-018-01625-y

ORIGINAL ARTICLE

Building hierarchical structures for 3D scenes with repeated elements

Xi Zhao[1] · Zhenqiang Su[2] · Taku Komura[3] · Xinyu Yang[1]

Abstract

We propose a novel hierarchy construction algorithm for 3D scenes with repeated elements, such as classrooms with multiple desk–chair pairs. Most existing algorithms focus on scenes such as bedrooms or living rooms, which rarely contain repeated patterns. Consequently, such methods may not recognize repeated patterns, which are vital for understanding the structure and context of scenes such as classrooms. Therefore, we propose a new global optimization algorithm for recognizing repeated patterns and building hierarchical structures based on repeated patterns. First, we find a repeated template by calculating the coverage ratios and frequencies of many substructures in a scene. Once the repeated template has been determined, a minimum cost maximum flow problem can be solved to find all instances (repetitions) of it in the scene and then group objects accordingly. Second, we group objects in the region outside the repeated elements according to their adjacency. Finally, based on these two sets of results, we build the hierarchy of the entire scene. We test this hierarchy construction algorithm on the Princeton and SceneNN databases and show that our algorithm can correctly find repeated patterns and construct a hierarchy that is more similar to the ground truth than the results of previous methods.

Keywords 3D scene · Scene analysis · Hierarchy · Repeated patterns

1 Introduction

Currently, 3D scenes consisting of many realistic models are commonly encountered. Enabling a computer to understand such data in a way that is consistent with human perception is essential for applications such as context-based retrieval, 3D scene synthesis, somatic games, and virtual reality. Hierarchical structures, which define different levels of local regions and their relationships, are excellent tools for representing the context of 3D scenes. Such hierarchical structures are widely used for the analysis, editing and synthesis of 3D models and scenes.

Representing 3D scenes with hierarchical structures is not a trivial problem. The challenge lies in the multiple factors that need to be considered when building such a structure. We need to consider not only the geometry of each individual object but also the contextual information of the scene, such as the spatial relationships between individual objects or groups of objects. Meanwhile, we also need to consider the semantic categories of the individual objects, which can be either directly provided or determined through analysis. Furthermore, it is useful to encode the higher-level features of a scene, such as repeated patterns, into the hierarchy because such repeated parts may all perform the same function and thus should be logically grouped together or similarly processed. Therefore, capturing such features can greatly help in understanding the structure and content of 3D scenes.

Existing methods for extracting the structure of a scene rely on either learning-based methods that learn from ground-truth structures or analytical methods that organize a scene based on the spatial relationships between different scene elements. A learning-based method, such as that of Liu et al. [16], can learn from consistent hierarchies and labels and then use the learned grammar to parse new scenes. In an analytical method, such as that of [28], the affinity between scene

✉ Xinyu Yang
xyyang@mail.xjtu.edu.cn

Xi Zhao
xi.zhao@mail.xjtu.edu.cn

[1] School of Electronic and Information Engineering, Xi'an Jiaotong University, 28 West Xianning Road, Xi'an 710049, Shaanxi, China

[2] School of Software Engineering, Xi'an Jiaotong University, 28 West Xianning Road, Xi'an 710049, Shaanxi, China

[3] School of Informatics, Edinburgh University, 10 Crichton Street, Edinburgh, UK

Published online: 08 January 2019

elements is first measured based on the interaction bisector surface (IBS), and then, the elements are gradually merged to build the hierarchy. Although these methods can compute plausible structures for scenes such as bedrooms, they are not designed to find repeated patterns. Consequently, they are not guaranteed to find the repeated patterns in a scene, thus limiting the accuracy of the constructed hierarchical representation.

In this paper, we propose a method of building hierarchical structures for 3D scenes with repeated patterns. There are two key challenges that need to be addressed: (1) checking whether a repeated pattern exists and generating the corresponding *repeated template* and (2) finding the *instances* of the repeated template, i.e., the repetitions of the template in the scene, and then building the hierarchy. To address these problems, we first list all possible templates for a scene based on a graph of the scene, and we then identify the repeated template by considering the coverage ratio and number of repetitions of each candidate. If a repeated template exists, we find all instances of that repeated template in the scene by solving a minimum cost maximum flow problem. Subsequently, we construct groups of objects in the regions both with and without the repeated pattern and use the results to build the final structure of the whole scene.

The main contribution of our work is that we propose a novel method of finding repeated patterns in 3D scenes that takes advantage of the strengths of both the IBS representation and the Ford–Fulkerson method. By using the IBS, we can effectively find the repeated template by capturing the immediate neighbors of each object. With other methods, such as a distance-based relationship representation, we might become trapped in a state with a massive number of neighbors and suffer from difficulty finding a suitable threshold with which to define a neighborhood. By constructing a flow network, we convert the problem of finding repeated instances into a minimum cost maximum flow problem. By assigning suitable capacities and costs to the network edges, our method elegantly finds the optimal solution for forming object groups according to the template. This design avoids the problem of encoding semantic labels into measures of the distances between objects, which is necessary when directly applying a clustering method to a scene graph. The resulting hierarchies that include repeated patterns can be potentially used for scene synthesis, scene completion, and scene editing. We quantitatively evaluate the proposed method for comparison with state-of-the-art algorithms. We show that our algorithm can produce structures that are more consistent with the ground truth than those generated with previous methods are. For the SceneNN database which has no ground truth, we conduct a user study to evaluate the hierarchies produced by our method.

2 Related work

Our paper is most closely related to work on building structural representations for 3D scenes. A structural representation for a 3D scene can be a flat graph or a tree structure. Such representations can be used for scene display and reasoning [19,22], scene analysis [9,11,16,28], scene comparison [6,21,25], and scene synthesis [27]. Among the related research on this topic, we are most interested in methods of building hierarchical representations for scenes; such methods can be either learning-based methods, such as that of [16], or analytical methods, such as those of [28] and [9]. Liu et al. [16] learn a probabilistic hierarchical grammar from manually labeled scene graphs and then use the learned grammar to parse new scenes. Their method not only builds hierarchies for scenes but also tags scene elements at different levels during the parsing process. Zhao et al. [28] build scene hierarchies based on the spatial relationships between scene elements. After analyzing the affinities between objects, they gradually merge objects into groups according to these affinities until all elements have been combined into one group. Hu et al. [9] build another type of tree structure, in which the root of the tree is the host object in the scene. The advantage of their method is that the relationships between the host object and its neighbors are clustered and reorganized such that the resulting structure is not sensitive to the number of objects in the scene. However, none of these methods explicitly considers the repeated patterns in scenes, and consequently, they are not guaranteed to find such repeated patterns.

Structure-aware shape analysis, which has attracted considerable attention, is also related to our work. Although 3D scenes are different from individual 3D objects, the related research on shape analysis can inspire our work in two ways. On the one hand, it has been proven that a hierarchical/tree structure is quite useful for shape analysis [8,23,24] and synthesis [2,12,13]. For example, Wang et al. [24] detect the symmetries in an object and construct a hierarchy based on these symmetries. The hierarchy is recursively generated based on handcrafted rules. To synthesize various 3D models, Alhashim et al. [2] first build graphs for two given objects and then blend the two graphs both topologically and geometrically based on the correspondence between them. Li et al. [13] synthesize new structures by using a recursive neural network. These works show that a structure-aware representation is an effective means of encoding both the spatial relationships and the priorities between relationships of different object parts. On the other hand, it is believed that repeated patterns are essential structural features of a shape [18]. Bokeloh et al. [3] have proposed a method of inverse procedural modeling. They examine the partial symmetry structures of 3D models and find the parts that maintain local similarity. Various shapes can be produced by repeating the

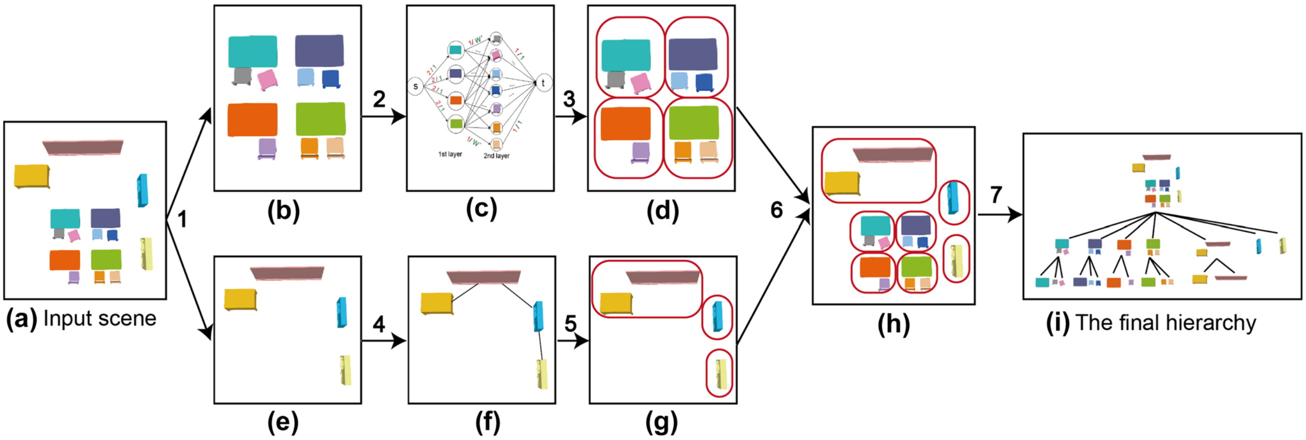

Fig. 1 Overview of our method. Step 1: An input 3D scene (**a**) is split into the repeating region (**b**) and the nonrepeating region (**e**). Step 2: A flow network (**c**) is built based on (**b**). Step 3: We solve a minimum cost maximum flow problem to obtain the segmented scene (**d**). Step 4: A graph (**f**) is created for the rest of the scene (**e**). Step 5: Segmentation results (**g**) are computed using the method of [28]. Step 6: The two sets of segmentation results are merged to form (**h**). Step 7: The final hierarchy (**i**) is constructed by using grouping (**h**) as the second level and applying optional further grouping steps

replacement and insertion operations. Despite the difference in scope between shapes and scenes, we find that the concept of repeated patterns is also essential for capturing the content of 3D scenes.

Finding periodic or repeated patterns in 2D images is a popular research topic since it is quite useful for image foreground segmentation [10], image analysis [1,14], image editing [4] and image synthesis [26]. Ahuja et al. [1] find the repeated elements in an image by building a tree structure for the image based on multiscale segmentation. Also using hierarchical segmentation, Cheng et al. [4] have proposed a boundary band method for detecting repeated patterns and further finding the correspondence between each detected region and a user-specified template. Huang et al. [10] find repeated scene elements by solving a max-flow min-cut problem for a graph built from a 2D image. Repeated patterns also occur in 3D models. Liu et al. [15] have proposed a method of detecting repeated templates from periodic reliefs of 3D models. Their system first initializes the cutting planes semiautomatically and then refines the cutting locations via surface registration. Gal et al. [7] detect subpart similarity in 3D models by first finding feature points on a 3D surface and then applying a geometric hashing method for partial matching with a repeated template. Without any prior knowledge, the method presented in [15], which is based on a transformation space voting scheme, can detect complex regular structures in noisy or even incomplete geometries. Our method is similar to these works in the sense that we also need to find instances of templates with certain types of objects, numbers of objects, and spatial relationships between objects.

3 Our method

The pipeline of our method is summarized in Fig. 1. Given an input 3D scene, we first split the scene into the repeating region and the nonrepeating region (step 1) and then segment the repeating region by solving a minimum cost maximum flow problem (steps 2 and 3). For the remainder of the scene, we apply the method proposed by Zhao et al. [28] to group the objects (steps 4 and 5). The final scene structure is built by combining the two sets of results (step 6) and applying optional further analysis (step 7).

3.1 Terms

To describe our method more clearly, we define the following terms:

- *3D scene*: In this paper, we consider 3D scene data that consist of complete 3D object models. We assume that the scene geometry is well segmented into different objects and that each of the objects has a label that indicates its type, such as “desk” or “chair”.
- *Relation matrix* M: For a 3D scene with N objects, we compute an $N \times N$ matrix M. Element $M(i, j)$ of the matrix is a value that represents the affinity between the i_{th} and j_{th} objects. If $M(i, j) = 0$, it means that the two objects have no direct relationship and are not neighbors.
- *Scene graph* G: We also build a flat graph G for the 3D scene. The nodes of G represent the individual objects in the scene, and each node is labeled with the corresponding object type. Each edge in G corresponds to a nonzero element of the relation matrix. If $M(i, j) \neq 0$,

then nodes i and j are connected by an edge with a weight of $M(i, j)$.

- *Template T*: We use a template T to represent a structure consisting of certain types of objects in a certain relationship that appears many times in the scene.
- *Repeated template candidate T_c*: Each repeated template candidate T_c is the structure of a subgraph of G, which may be repeated many times in the scene. These candidates are used to find the repeated template for the scene.

3.2 Detection of the repeated pattern (steps 2 and 3)

Detecting the repeated pattern in a scene is the core of our algorithm. We first identify the repeated template through a heuristic search and then detect the instances of that repeated template using a global optimization method. Below, we explain these two steps in detail.

3.2.1 Identification of the repeated template

Given a 3D scene with N objects, we first compute its relation matrix M. We assume that the scene geometry is well segmented into individual objects and compute the affinity between objects i and j, denoted by $M(i, j)$, using the method proposed in [28]. This method is based on the IBS, which is a set of points that are equidistant from at least two objects in the scene. It is designed to represent the spatial relationships between the objects composing the scene. By considering the distance and direction features of the IBS region corresponding to each pair of objects, we can compute the $M(i, j)$ value to represent how strong the relationship between those two objects is. When the two objects are closer and more directly facing each other, the corresponding $M(i, j)$ is larger. If there is no IBS point between them, which means that they are not immediate neighbors, the corresponding $M(i, j)$ is zero. The relation matrix for the example scene considered here is visualized in Fig. 3.

With the relation matrix M, we then build the scene graph G and find all repeated template candidates. To find the repeated template candidates, we start from each node v_i of G and build a subgraph G_{v_i} that consists of the node v_i, all one-step neighbor nodes of v_i , and the edges between them (Fig. 2a). Next, we list all possible subgraphs of G_{v_i}. We define a subgraph as “valid” when it satisfies the following three conditions: (1) it contains at least two nodes, (2) it contains at least two types of nodes, and (3) it contains at least one node type that appears only once. These conditions ensure that no valid subgraph will itself contain a repeated pattern. An example that does not satisfy condition (2) is shown in the upper image in Fig. 2c; this subgraph consists of two desks. Another invalid example, which violates condition (3), is shown in the bottom image in Fig. 2c. Two

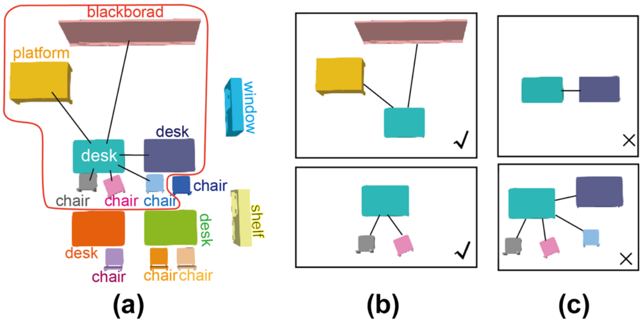

Fig. 2 **a** The blue desk and its one-step neighbors. **b** Examples of the corresponding candidate templates (valid subgraphs). **c** Examples of invalid subgraphs that cannot be used as candidate templates

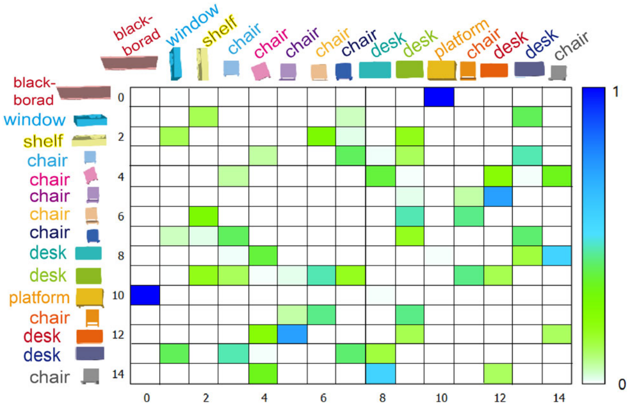

Fig. 3 Visualization of the relation matrix for the example scene in Fig. 2a

examples of valid subgraphs are shown in Fig. 2b. We refer to all valid subgraphs as the candidate repeated templates T_c.

Next, we identify the repeated template T from among all candidates T_c. For each candidate template T_c, we first find all nonoverlapping instances of T_c in the scene. Here, an instance is a subgraph of G that is a repetition of the template. More specifically, we consider a subgraph of G to be a repetition of T_c when it has the same topology and node labels as those of T_c. The *coverage* of a candidate template is then defined as the union of all nonoverlapping instances of that template. We compute the *coverage ratio* as the number of objects in the coverage divided by the total number of objects N. The candidate with the largest coverage ratio is considered to be the repeated template for the scene. If two candidate templates have the same coverage ratio, we further compute the *frequency*, which is the number of instances of the template within the coverage. We then choose the template with the higher frequency as the repeated template. By considering both the coverage and frequency, our method selects the minimum repeated template whose instances cover the largest area in the scene. If more than one candidate template has the same coverage ratio and frequency, we randomly select one of these candidates as the final template. The repeated template for the example scene is "desk–chair–chair". This template's coverage is $9/15 = 0.6$, and its frequency is 3.

Finding all nonoverlapping instances of a candidate template is not straightforward. A candidate template needs to have the largest number of nonoverlapping instances to be chosen as the final template. However, finding all nonoverlapping instances is equivalent to finding the repeated pattern. Therefore, we use a heuristic method to estimate the largest number of nonoverlapping instances as follows. We place all valid subgraphs in a queue and consider each subgraph in the order of the queue. If the current subgraph is an instance of the candidate template that does not overlap with existing instances, we add it to the instance list; if it does overlap with the existing instances, then we ignore it. Of course, the order of the queue influences the results. Therefore, we shuffle the queue many times and choose the best result. The assumption underlying this approach is that if the number of permutations of the queue is sufficiently large, then the considered permutations should include the one with the best order that finds the largest number of nonoverlapping instances. In our experiment, we empirically set the number of permutations to 200.

3.2.2 Detection of repeated instances

Given the identified repeated template, we next need to group the objects in the scene according to the repeated template. For example, because "desk–chair–chair" is the repeated template for the example scene, we need to decide which chair and desk models should be assigned to the same group. To compute a globally optimal solution from the many possible groupings, we propose a novel method based on the minimum cost maximum flow strategy.

We build a single-source, single-sink flow network for the scene. As an example, for the scene shown in Fig. 1, the corresponding flow network is shown in Fig. 4. Here, the source node "s" has only outgoing flows, and the sink node "t" has only incoming flows. First, we choose a "host object" for the repeated pattern, which must appear only once in the repeated template. For example, in the "desk–chair–chair" pattern, the desk is the host object. All objects with the desk label are placed in the first layer of the network and are directly connected to the source node. Then, the objects of the remaining types in the repeated template form the second layer of the network. The edges between the first and second layers are determined by the relationships between the objects in the scene graph G. An edge exists between object i in the first layer and object j in the second layer only if there is an edge between them in G. Finally, we connect all second-layer objects to the sink node.

The capacity of each edge in the flow network is defined as follows. The capacity of each edge from the source node to a host object is set to n, where n is the number of objects connected to the host object in the repeated template. For example, in the "desk–chair–chair" pattern, one desk is connected to two chairs, so the capacity of each edge from the source node to a desk node is 2. The capacity of each edge connecting a host-type object to any other object is set to 1. The capacity of each edge connecting a second-layer object to the sink node is also set to 1. With this flow network design, we attempt to connect each host object to a certain number of other objects in the next layer.

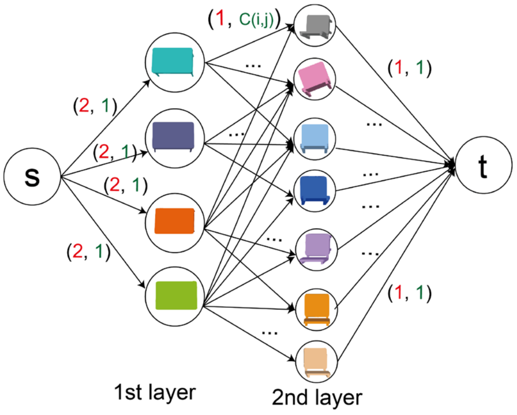

Fig. 4 Flow network for the example scene and the capacities and costs of the edges. For each edge, the red number before the comma is the capacity value, and the green number after the comma is the cost value. Desks are in the first layer, and chairs are in the second layer

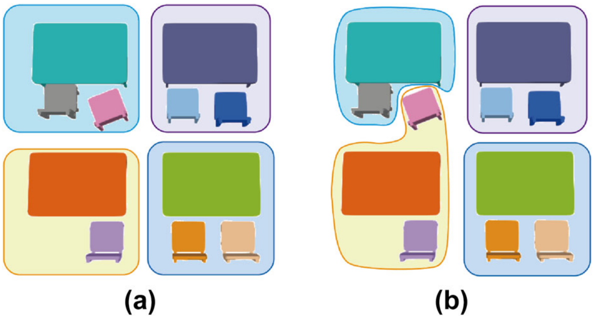

Fig. 5 Two segments corresponding to the maximum flow. When the cost is not considered, both **a** and **b** are optimal splits of the repeated pattern, although **a** is more reasonable

We also need to provide the network with the relationships between objects. Simply considering the object type and the number of objects connected to each host object is not enough because the maximum flow of such a network may not be unique. For example, in Fig. 5, the splits shown in (a) and (b) both correspond to the same flow. In other words, regardless of whether the pink chair is associated with the green desk in front of it or the orange desk behind it, the flow does not change, although the pink chair is closer to the green desk and also semantically belongs to it. To solve this problem, we add a cost to each edge in the flow network to represent the cost of each unit of flow. We compute the costs of the edges between layers of the flow network with the following equation:

$$C(i, j) = e^{-M(i,j)} \tag{1}$$

where M is the relation matrix for the scene computed as described in Sect. 3.2.1 and i and j are object IDs. Because $M(i, j)$ is a number greater than 0, the range of $C(i, j)$ is (0, 1]. A larger $M(i, j)$, which indicates a stronger spatial relationship between objects i and j, corresponds to a smaller cost in the flow network. Edges with smaller costs are more likely to be selected for the optimized flow solution. The costs of all edges connected to the source or sink are set to 1. The cost setting for the example scene is shown in Fig. 4.

We next apply the Ford–Fulkerson method [5] to find the minimum cost maximum flow solution for the flow network, with the aim of finding the flow with the lowest cost among all maximum flow solutions. Given a flow network with a source node and a sink node and the capacity and cost of each edge, this method proceeds as follows: After finding a minimum cost path by Floyd–Warshall method, we add a flow to each edge on this path with the bottleneck capacity and then update the flow and cost on each edge. This process is repeated until we cannot find any valid augmenting path. The detailed algorithm is presented as Algorithm 1. The Floyd–Warshall method works only for graphs with no negative cycles. We prove our flow networks are such type of graphs in the Appendix. The resulting minimum cost maximum flow solution for the example network in Fig. 4 is shown in Fig. 6.

Finally, we segment the scene based on the resulting flow solution; objects are grouped if there is a flow between them in the obtained solution. The resulting segmentation of the example scene is shown in Fig. 5a. Note that although we aim to find as many repeated instances of the template, i.e., subgraphs that have the same structure as the template, as possible, there might be some objects remaining that can form only part of the template. For example, in the example scene, based on the flow solution, the orange desk has only one chair, which means that there is only 1 unit of flow originating from the orange desk. There are similar cases for scenes such as classroom 4 in Fig. 7c and the scene in Fig. 9, in which some of the repeated instances are not identical to the template. In Fig. 7c, some desk–chair pairs are detected when the template is desk–chair–chair, and in Fig. 9, a single desk in the first row is identified. Strictly speaking, these instances are not repetitions of the template, but we have found that there is no harm in forming groups for such instances in the hierarchy because most of them make sense (such as the desk–chair pair in Fig. 7c). On the other hand, we can consider such instances as irregular subregions of the scene and can complete them if necessary.

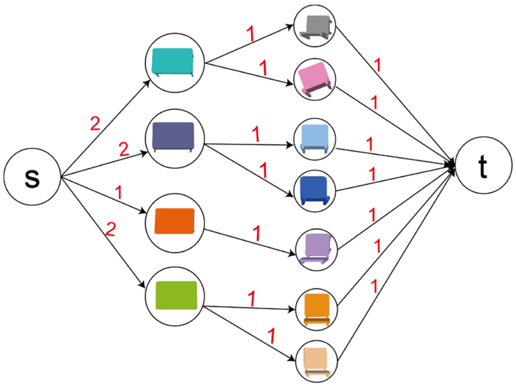

Fig. 6 The minimum cost maximum flow solution for the flow network of the example scene

```
Data: A flow network $F$ with capacity matrix $c$, cost matrix $a$,
      source node $s$, and sink node $t$
Result: A minimum cost maximum flow solution $f$
1. $f(u, v) = 0$, $w(u, v) = a(u, v)$ for all edges;
2. Find an augmenting path $p$ by Floyd–Warshall $(F)$;
3. while $p$ exists do
    Find the bottleneck capacity: $c_{add}(p) = min\{c(u, v)\}$;
    for each edge $(u, v) \in p$ do
        if edge $(u, v)$ is forward then
            if $c_{add}(p) < c(u, v)$ then
                $f(u, v) = f(u, v) + c_{add}(p)$;
            else
                /* if $c_{add}(p) = c(u, v)$ */
                $f(u, v) = f(u, v) + c_{add}(p)$;
                $w(u, v) = MAX$;
                $w(v, u) = -a(u, v)$;
            end
        else
            /* if the edge is backward */
            if $c_{add}(p) < c(u, v)$ then
                $f(v, u) = f(v, u) - c_{add}(p)$;
                $w(u, v) = -a(v, u)$;
                $w(v, u) = a(v, u)$;
            else
                /* if $c_{add}(p) = c(u, v)$ */
                $f(v, u) = f(v, u) - c_{add}(p)$;
                $w(u, v) = MAX$;
                $w(v, u) = a(v, u)$;
            end
        end
        $c(u, v) = c(u, v) - c_{add}(p)$;
        $c(v, u) = c(v, u) + c_{add}(p)$;
    end
end
return f;
```

Algorithm 1: The Ford–Fulkerson method

3.3 Construction of the hierarchy

In this section, we describe the process of building the scene hierarchy. In the lowest level (the first level) of the hierarchy, we assign each object to a single group. Then, in the second level, we apply the method described in Sect. 3.2 to find the repeated pattern and place the resulting segments in the second level of the hierarchy. After that, we can either merge all segments to the root or further group the segments until we reach the root. The whole process is shown in Algorithm 2.

```
Data: A scene with $n$ objects $\{o_1, o_2, \ldots, o_n\}$, where each
      object has a label $\{l_1, l_2, \ldots, l_n\}$
Result: A hierarchy $H$ built for the whole scene
1. $H = \emptyset$;
2. Build the first level of the hierarchy:
Assign each single object to a group;
H = $\{\{o_1\}, \{o_2\}, \ldots, \{o_n\}\}$;
3. Build the second level of the hierarchy:
begin
    Define the current level of grouping $L = \emptyset$;
    (1) Split the scene into two parts, $P_1$ and $P_2$;
    $P_1 = \{o_1^{p1}, o_2^{p1}, \ldots, o_n^{p1}\}$, where the $o_i^{p1}$ are objects
    whose labels appear only once in the whole scene;
    $P_2 = \{o_1^{p2}, o_2^{p2}, \ldots, o_n^{p2}\}$, where $o_i^{p2} \notin P_1$;
    (2) Find the repeated pattern in $P_2$;
    while $T$ exists do
        Build a network $F$ for $P_2$;
        Apply Algorithm 1 to $F$ to obtain the minimum cost
        maximum flow solution $f$;
        Convert $f$ into grouping $g_i$;
        $L = L \bigcup g_i$;
        $P_2 = P_2 - G_i$, where $G_i$ is the set that contains all
        objects in $g_i$;
        Search for the template $T$ in $P_2$;
    end
    (3) Apply the method of [28] to $P_1$ to obtain the grouping
    $g^{p1}$;
    $L = L \bigcup g^{p1}$;
    Assign this grouping to the second level from the lowest;
    $H = H \bigcup L$;
end
4. Further group the scene elements using the method of [28] or
some other method (optional);
$H = H \bigcup L$;
5. Merge all segments into one (the root $R$);
$H = H \bigcup R$;
return H;
```

Algorithm 2: Construction of the hierarchy

To apply the algorithm described in Sect. 3.2 more robustly, we add two further processes when computing the second level of grouping in the hierarchy. First, before finding the repeated pattern, we split the scene into two parts: the repeating region and the nonrepeating region. The nonrepeating region contains all objects that have a label that appears only once in the scene, and the repeating region is the remainder of the scene. The motivation for this process is straightforward: if a given object type appears only once in the scene, it cannot be part of the repeated pattern. By removing such objects when computing the repeated pattern, we can reduce the number of subgraphs to be analyzed and identify the template more quickly. For the nonrepeating region, we apply the method of [28] to merge nearest neighbors and add the resulting grouping to the second level of the hierarchy, as well. Second, to consider multiple types of templates, we apply the Ford–Fulkerson method multiple times to find all different repeated patterns. Initially, we find the instances for an identified template with Algorithm 1. Then, the objects in these instances are removed, and the algorithm is applied again to the remaining objects until no more template instances can be found.

Once the second level of the hierarchy has been constructed, grouping methods such as hierarchical agglomerative clustering (HAC) can be iteratively applied to further merge the groups until the root is reached. This step is optional; it is mainly useful for large-scale scenes that contain multiple sections with different functions. In our experiments, because the data we process mainly represent single rooms, we skip this step and directly merge the second level of the hierarchy to the root node.

4 Experiments

4.1 Database

We apply the proposed method to the Princeton scene database [16]. We test and evaluate our algorithm on the 30 classroom scenes and 8 library scenes in this database. Each of these scenes has a manually constructed ground-truth hierarchy, which can be used for the evaluation of our method. We do not use the bedroom scenes because there are few repeated patterns in such scenes.

We also test our algorithm on the SceneNN database [20], which contains different types of scenes with repeated patterns, such as office, meeting room and canteen scenes. Compared to the Princeton database, the SceneNN database is noisy, and it is incomplete in both geometry and labeling. All segments in a SceneNN scene are labeled with either a semantic label such as “desk” or the meaningless label “none”. To use these data, we extract the segments with semantic labels and apply our method only to these segments. In most cases, segments with meaningful shapes have semantic labels. In our experiments, we ignore the segments that are labeled as “floor” or “wall” when building the hierarchy of the scenes.

4.2 Results

Results for scenes from the Princeton database are shown in Fig. 7. From these results, we can see that our method

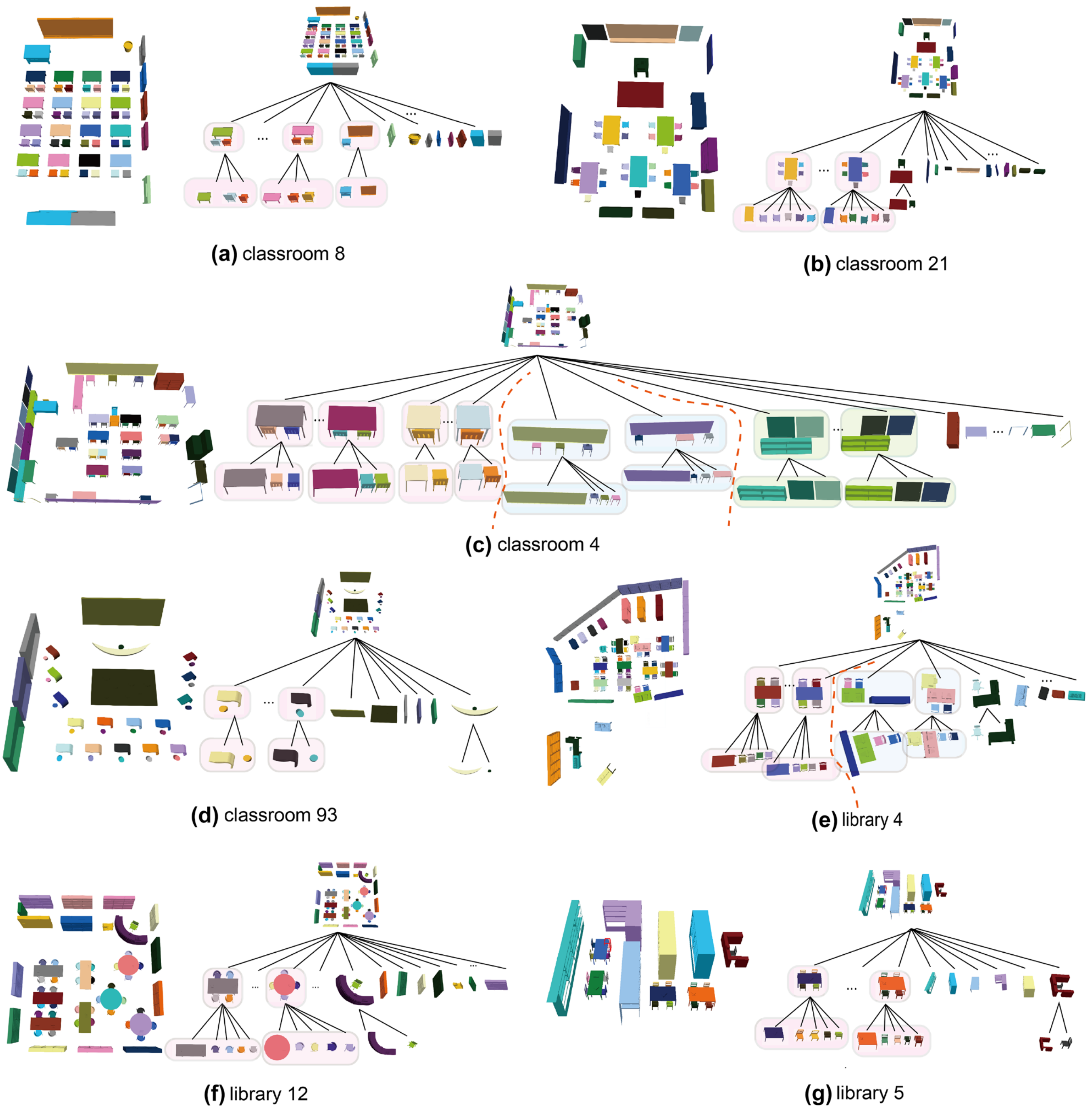

Fig. 7 Original scenes (left) and resulting hierarchies obtained with our method (right). **a**–**d** show results for classroom scenes, while **e**–**g** show results for library scenes. Different background colors are used to highlight patterns detected with different templates. Note that in **c**—the second pattern with the light blue background, all the objects around the two boards are labeled as "cabinet" in the Princeton database despite their different geometry. So they are detected as repeated patterns

can correctly detect repeated patterns in the scenes and can also successfully detect multiple repeated patterns in a scene (Fig. 7c, e). Different background colors correspond to the patterns detected with different templates. Some results for SceneNN scenes are shown in Fig. 8a–d. Although the SceneNN data are noisy and incomplete, our method can still detect repeated patterns. In Fig. 8e, we show the hierarchy obtained for a "musical chairs" game scene in which five children will sit on four chairs. From the hierarchy, we can pair four of the children with corresponding chairs and identify the child without a chair, who has a larger chance to fail. We also show results for some other types of scenes, such

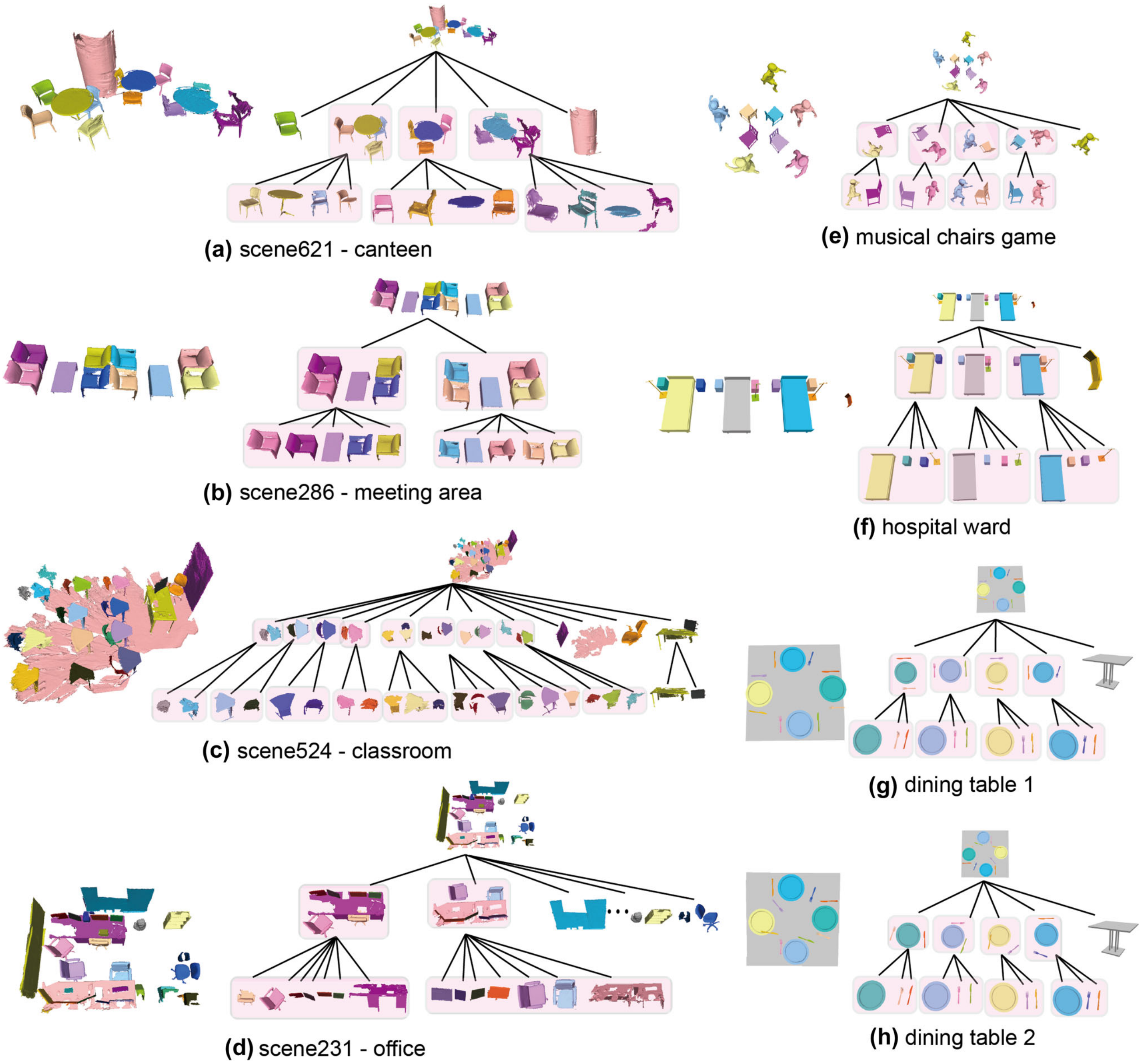

Fig. 8 Original scenes (left) and resulting hierarchies obtained with our method (right). **a–d** show results for SceneNN scenes, while **e–h** show results for other scenes. A light pink background is used to highlight the repeated patterns detected with our method

as a hospital ward (Fig. 8f) and dining tables with tableware (Fig. 8g, h).

4.3 Evaluation on Princeton database

We evaluate the hierarchy results of the Princeton database in two ways.

First, we evaluate the topological similarity between our results and the ground truths using the γ value proposed in [17], which is computed by comparing the merging orders of the scene element pairs in two hierarchies. If objects $a1$ and $a2$ are merged earlier than $b1$ and $b2$ in hierarchy $h1$ and $a1$ and $a2$ are also merged earlier than $b1$ and $b2$ in hierarchy $h2$, then this is a consistent order (a good order). However, if $a1$ and $a2$ are merged later than $b1$ and $b2$ in hierarchy $h2$, this is a reverse order (a bad order). Then, the evaluation value γ, which considers the relative proportions of good orders and bad orders, is used to describe the similarity. $\gamma = 1$ means that the proportion of good orders is 1, i.e., the merging order is the same for both hierarchies. $\gamma = -1$ means that the proportion of bad orders is 1, i.e., the merging order is completely different between the two hierarchies.

Second, we compare the repeated patterns represented in the two hierarchies by the Rand index. Because single levels in the hierarchy can be regarded as clusterings of the scene, and the Rand index is a standard measure that counts the proportion of "agreements" between two clusterings. In our experiments, we first extract the level L_g that contains the repeated pattern in the ground-truth hierarchy and the level L_i that represents the repeated pattern in the identified hierarchy. Then, we count the agreements, which is the number of objects pairs that are either in the same subset in both L_g and L_i or in different subsets in both L_g and L_i. Finally, the Rand index for L_g and L_i is calculated by dividing the agreements number by the total number of object pairs.

We compare the proposed method with two alternative methods:

- *Learning* The learning-based method proposed in [16] is used to build the hierarchy.
- *HAC* The HAC method [28] is used to build the hierarchy.

The first alternative is a learning-based method that predicts both the structure of a scene and the label of each subpart of the scene. In this method, the scene is divided into small pieces that are subparts of every individual object. Because our method divides the scene only to the object level and does not further divide a single object into smaller subparts, when evaluating the results using the method of [16], we compare the results of the learning-based method with the ground truth only from the root level to the level of individual objects. The second alternative method also uses a weight matrix computed with the IBS approach, based on which the HAC method is used to build the hierarchy.

We use the scenes and ground-truth hierarchies from the Princeton database to perform the evaluation. We compute both the γ value and the Rand index for each scene from the database, and finally, we compute the average γ value and the average Rand index. When computing the Rand index results for the learning-based method and the HAC method, because it is unknown which level in the generated hierarchy best represents the repeated pattern, we compare the ground-truth clustering L_g with each level in the hierarchy and select the level with the maximum Rand index. By doing so, we select the level that is most similar to L_g as the basis for our evaluation.

Table 1 The average γ values and Rand index values for different methods

Method	γ value	Rand index
Our method	0.8593	0.9883
Learning	0.8377	0.9832
HAC	0.4931	0.9221

The γ value is used to evaluate the merging orders of the generated hierarchies, and the Rand index is used to evaluate the precision with which repeated patterns are detected

The evaluation results are shown in Table 1. From the results, we can see that our method produces a slightly higher γ than the learning-based method does. This means that the consistency in merging order between our results and the ground truth is comparable to that of the learning-based method. The HAC method achieves the lowest value because it is based purely on geometry and does not consider repeated patterns at all. In this method, a chair close to a window might be grouped with the window rather than a nearby desk, for example, leading to merging orders that are the inverse of those in the ground truth. Consequently, the γ value of this method is much lower than those of the other methods.

The Rand index for our method is also the highest among all methods. This means that our method detects repeated patterns more precisely. To visually illustrate the advantages of our method, we visualize one level of the hierarchy as obtained with each of the different methods for an example scene in Fig. 9. Figure 9a shows the level of the ground-truth hierarchy that contains the repeated pattern. Panels (b–d) show the level with the highest Rand index in the hierarchy obtained with each method. In (b), (c) and (d), we use boxes to highlight the differences between the grouping results and the ground truth. We can see from (b) that our method successfully finds all repeated instances and that the resulting grouping is quite similar to the ground truth. The results of the HAC method, as shown in (c), contain four large groups with no detected "desk–chair" patterns. Panel (d) shows the results computed with the learning-based method, which cannot be guaranteed to correctly find all repeated template instances. There are many cases in which a chair is grouped with the desk behind it. This results in many triplets, which are highlighted with red boxes.

4.4 Evaluation on SceneNN database

To evaluate our results on the SceneNN database, we conduct an online survey in terms of the plausibility of the hierarchical structures computed by our method. We use 15 scenes from the SceneNN database for this study. The pairs of one scene and the corresponding structure are presented to 30 participants that have no graphics or interior design related background. Participants were asked about the plausibility and reasonability of the hierarchical structures on a five-point Likert scale (1 = not reasonable at all; 5 = perfectly reasonable). The survey contains three types of scenes: 8 office scenes, 6 meeting area/canteen scenes, and 1 classroom scene. We consider a scene with multiple student desk–chair pairs and a platform area as a classroom. There is only one such classroom in the SceneNN database. We found this survey takes 12.2 minutes to finish on average. Both the online survey and the ratings

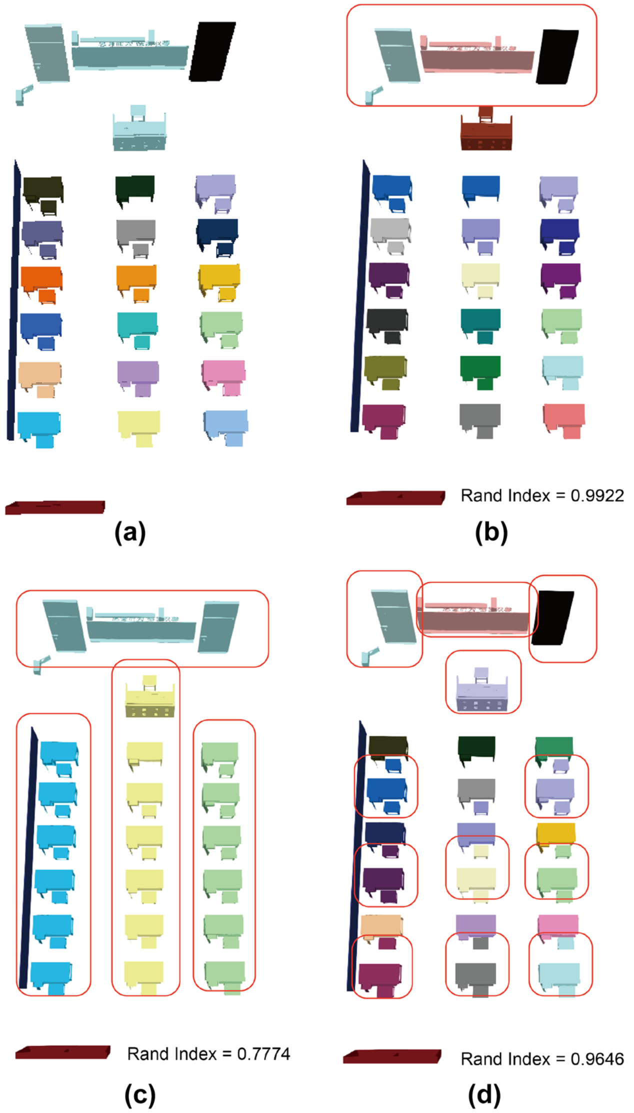

Fig. 9 Comparison of the groupings in the hierarchies of **a** the ground truth, **b** our method, **c** the HAC method [28] and **d** the learning-based method [16]. Objects shown in the same color are in the same group. In the lower right corner of each panel, we show the Rand index computed between the results of the corresponding method and the ground truth

given by the participants can be found in the supplementary material.

Figure 10 shows the distribution of the ratings for each type of scenes in (a)–(c) and that for all scenes in (d). The average rating for our results is 3.678. Overall, there are 89.6% of the ratings given by participants are equal or greater than 3, and 59.1% of the ratings are equal or greater than 4. Scenes such as classroom and meeting room/canteen area get higher ratings as these scenes contain more subparts which can be detected by repeated patterns, while some small office rooms which are quite crowded and have no repeated pattern detected get lower rating.

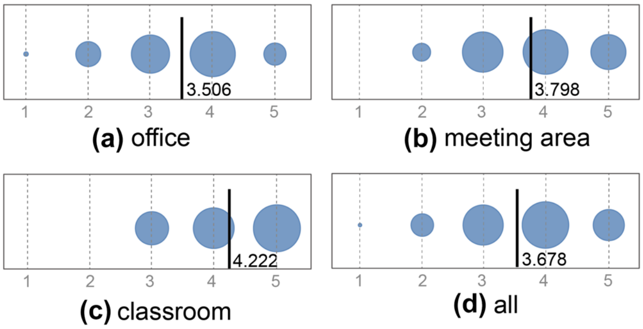

Fig. 10 Results of our survey for evaluating the plausibility of our results on SceneNN database. We show the distribution of the ratings for each type of scenes first and then show the distribution of ratings for all scenes. The size of each bubble represents the percentage of the corresponding rating. In each chart, the vertical line and the number beside it show the average rating of the corresponding group of scenes

4.5 Timing and parameter setting

The time cost of our method can be divided into three components. The first component is the time needed to create the candidate templates. The time cost of this step is related to the number of objects in the scene and the number of neighbors of each object. In our experiment, it takes 0.48 milliseconds to create candidate templates for each node on average, and each object has 2.78 neighbors on average. The average number of candidate templates per scene is 35. The second component is the time needed for selecting the optimal template from among the candidates. To choose the optimal template, we need to compute the coverage ratio and frequency for each candidate. Thus, the time cost of this step is primarily related to the number of subgraphs we need to check, N_{sub}, and the number of permutations of the subgraph list during the counting of nonoverlapping instances, T_{sh}. Therefore, the time complexity is $O(N_{\text{sub}}T_{\text{sh}})$. In our experiment, N_{sub} is 561 for each scene on average, and we empirically set T_{sh} to 200 for all data. The average value of this component of the time cost is 0.11 seconds. The third component is the time taken to apply the Ford–Fulkerson method. The time complexity of this method is $O(Ef)$, where E is the number of edges in the flow network and f is the maximum flow. If we denote the average number of neighbors of each node by a and the average number of objects in each scene by n, then from the way the network is constructed, we can see that $E \leq ((n \times a)/2 + n)$ and $f \leq (n \times a/2)$. Thus, the upper bound on the time complexity is $O(n^2a^2)$. In our experiment, the average number of objects per scene is 47, and the average number of neighbors is 2.78, as mentioned above. Each application of the Ford–Fulkerson method takes 0.021 seconds on average.

The relation matrix contains many small values (smaller than 0.01). There are approximately 40 such values in the example matrix shown in Fig. 3. These small values indicate that the corresponding relationships between objects are very weak and that these objects can only barely be considered neighbors. To reduce the number of neighbors, we set a threshold th to reject such weak relationships. For the Princeton database, we set the threshold value to 0.1, and for the SceneNN database, we find that a threshold of 0.05 yields the best results.

5 Discussion and future work

In this paper, we propose a method of building hierarchies for indoor scenes that contain repeated elements, such as classrooms and libraries. In our method, the globally optimal split is found by solving a minimum cost maximum flow problem. We demonstrate that our method can correctly find the repeated pattern that is most consistent with the ground truth identified by a human analyst. We quantitatively evaluate the results and show that our method outperforms state-of-the-art methods when applied to the Princeton database.

Our work has some limitations. First, our method relies on well-segmented and properly labeled input scenes because the processes of both finding the repeated template and computing the repeated pattern rely on the geometry and the category labels of all objects. Second, when building the flow network, we ignore the relationships between objects in the same layer. For example, in Fig. 4, the relationships between different chairs or different desks are not considered when finding the repeated pattern. With the current system, we focus only on finding the repeated pattern and do not build a further hierarchy within the repeated template. However, such an approach might be needed for more complex scenes, such as the office scene in Fig. 8d.

In the future, we would like to improve our system in three ways. First, we will explore how to automatically segment raw scenes into meaningful parts and produce consistent category labels for the scene elements. Second, we will also explore a better way to construct the flow network to handle more complex repeated patterns. Finally, we are interested in combining our method with the learning-based method to develop a new approach that can consider repeated patterns when defining and learning a scene grammar.

Funding This study was funded by the National Natural Science Foundation of China (61602366) and the China Postdoctoral Science Foundation (2015M582664).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

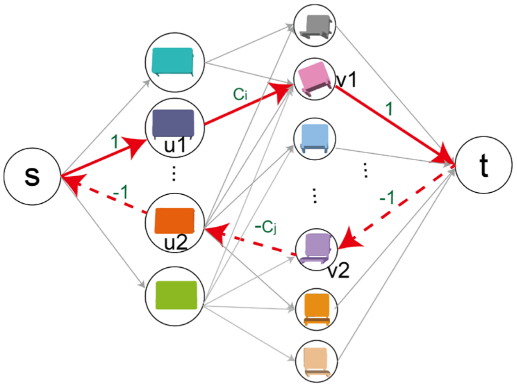

Fig. 11 An arbitrary loop in the cost network always has a positive-cost value

A Proof that the flow network contains no negative-cost cycles

The Floyd–Warshall method works only when the network contains no negative-cost cycles. Therefore, we need to prove here that the directed cost graph from which we find the minimum path satisfies this condition. According to the Ford–Fulkerson method, we iteratively repeat two steps: (1) searching for an augmenting path in the cost network and (2) updating the cost network (corresponding to the matrix w in the algorithm) after adding the new augmenting path. The cost network may have two types of edges: positive edges, which have the same directions and cost values $C(i, j)$ as those of the corresponding original edges in the flow network, and negative edges, which have the opposite directions and negative-cost values $-C(i, j)$. Consider a cycle in a cost network, such as that shown in Fig. 11. Note that the edges connected to the source or sink in a cycle are always paired. Because we set the costs of all edges connected to the source or sink to 1, the costs of such an edge pair, such as $(e(s, u1), e(u2, s))$ or $(e(v1, t), e(t, v2))$, always cancel to zero. Thus, the total cost of the cycle is $C_i - C_j$, where C_i is the cost of edge $e(u1, v1)$ and $-C_j$ is the cost of edge $e(v2, u2)$. There is a negative-cost edge from $v2$ to $u2$, meaning that the flow of edge $e(u2, v2)$ has been used as part of the flow solution. On the other hand, the positive-cost edge $e(u1, v1)$ indicates that the flow of this edge has not been used. The edge $e(u2, v2)$ is chosen earlier than $e(u1, v1)$ indicates that $e(u2, v2)$ has a smaller weight, so we can infer that $C_j \leq C_i$. Therefore, the cost of the cycle cannot be negative.

References

1. Ahuja, N., Todorovic, S.: Extracting texels in 2.1D natural textures. In: 2007 IEEE 11th International Conference on Computer Vision, pp. 1–8 (2007)

2. Alhashim, I., Li, H., Xu, K., Cao, J., Ma, R., Zhang, H.: Topology-varying 3D shape creation via structural blending. ACM Trans. Gr. (TOG) **33**(4), 158 (2014)
3. Bokeloh, M., Wand, M., Seidel, H.P.: A connection between partial symmetry and inverse procedural modeling. ACM Trans. Gr. (TOG) **29**(4), 104 (2010)
4. Cheng, M.M., Zhang, F.L., Mitra, N.J., Huang, X., Hu, S.M.: RepFinder: finding approximately repeated scene elements for image editing. ACM Trans. Gr. **29**(4), 1 (2010)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
6. Fisher, M., Savva, M., Hanrahan, P.: Characterizing structural relationships in scenes using graph kernels. ACM Trans. Gr. (TOG) **30**, 34 (2011)
7. Gal, R., Cohen-Or, D.: Salient geometric features for partial shape matching and similarity. ACM Trans. Gr. (TOG) **25**(1), 130–150 (2006)
8. Golovinskiy, A., Funkhouser, T.: Consistent segmentation of 3D models. Comput. Gr. **33**(3), 262–269 (2009)
9. Hu, R., Zhu, C., van Kaick, O., Liu, L., Shamir, A., Zhang, H.: Interaction context (ICON): towards a geometric functionality descriptor. ACM Trans. Gr. **34**, 83:1–83:12 (2015)
10. Huang, H., Zhang, L., Zhang, H.C.: RepSnapping: efficient image cutout for repeated scene elements. Comput. Gr. Forum **30**(7), 2059–2066 (2011)
11. Hueting, M., Monszpart, A., Mellado, N.: MCGraph: multi-criterion representation for scene understanding. In: SIGGRAPH Asia 2014 Indoor Scene Understanding Where Graphics Meets Vision, p. 3. ACM (2014)
12. Kalogerakis, E., Chaudhuri, S., Koller, D., Koltun, V.: A probabilistic model for component-based shape synthesis. ACM Trans. Gr. (TOG) **31**(4), 55 (2012)
13. Li, J., Xu, K., Chaudhuri, S., Yumer, E., Zhang, H., Guibas, L.: GRASS: generative recursive autoencoders for shape structures. ACM Trans. Gr. **36**(4), 1–12 (2017). arXiv:1705.02090
14. Liu, Y., Collins, R., Tsin, Y.: A computational model for periodic pattern perception based on frieze and wallpaper groups. IEEE Trans. Pattern Anal. Mach. Intell. **26**(3), 354–371 (2004)
15. Liu, S., Martin, R.R., Langbein, F.C., Rosin, P.L.: Segmenting periodic reliefs on triangle meshes. In: Martin, R., Sabin, M., Winkler, J. (eds.) Mathematics of Surfaces XII, vol. 4647, pp. 290–306. Springer, Berlin (2007)
16. Liu, T., Chaudhuri, S., Kim, V.G., Huang, Q., Mitra, N.J., Funkhouser, T.: Creating consistent scene graphs using a probabilistic grammar. ACM Trans. Gr. **33**(6), 211:1–211:12 (2014)
17. MacDonald, D., Lang, J., McAllister, M.: Evaluation of colour image segmentation hierarchies. In: The 3rd Canadian Conference on Computer and Robot Vision, 2006, p. 27 (2006)
18. Mitra, N., Wand, M., Zhang, H.R., Cohen-Or, D., Kim, V., Huang, Q.X.: Structure-aware shape processing. In: SIGGRAPH Asia 2013 Courses, SA '13, pp. 1:1–1:20. ACM, New York (2013)
19. Moulin, M., Dutré, P.: On the use of local ray termination for efficiently constructing qualitative BSPs, BIHs and (S)BVHs. Vis. Comput. (2018). https://doi.org/10.1007/s00371-018-1575-x
20. Nguyen, D.T., Hua, B.S., Yu, L.F., Yeung, S.K.: A Robust 3D–2D Interactive Tool for Scene Segmentation and Annotation (2016). arXiv:1610.05883 [cs]. arXiv:1610.05883
21. Paraboschi, L., Biasotti, S., Falcidieno, B.: 3D scene comparison using topological graphs. In: Eurographics Italian Chapter Conference, pp. 87–93 (2007)
22. Sappa, A.D., Garcia, M.A.: Generating compact representations of static scenes by means of 3D object hierarchies. Vis. Comput. **23**(2), 143–154 (2007)
23. Sidi, O., van Kaick, O., Kleiman, Y., Zhang, H., Cohen-Or, D.: Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. ACM Trans. Gr. (TOG) **30**, 126 (2011)
24. Wang, Y., Xu, K., Li, J., Zhang, H., Shamir, A., Liu, L., Cheng, Z., Xiong, Y.: Symmetry hierarchy of man-made objects. In: Computer Graphics Forum, vol. 30, pp. 287–296. Wiley Online Library (2011)
25. Xu, K., Ma, R., Zhang, H., Zhu, C., Shamir, A., Cohen-Or, D., Huang, H.: Organizing heterogeneous scene collections through contextual focal points. ACM Trans. Gr. (TOG) **33**(4), 35 (2014)
26. Zhang, F.-L., Cheng, M.-M., Jia, J., Shi-Min, H.: ImageAdmixture: putting together dissimilar objects from groups. IEEE Trans. Vis. Comput. Gr. **18**(11), 1849–1857 (2012)
27. Zhao, X., Hu, R., Guerrero, P., Mitra, N., Komura, T.: Relationship templates for creating scene variations. ACM Trans. Gr. **35**(6), 207:1–207:13 (2016)
28. Zhao, X., Wang, H., Komura, T.: Indexing 3D scenes using the interaction bisector surface. ACM Trans. Gr. (TOG) **33**(3), 22:1–22:14 (2014)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Xi Zhao received the B.Sc. degree from University of Shanghai of Science and Technology in 2007, M.Sc. degree from Southeast University in 2010 and PhD degree from Edinburgh University in 2014. Currently she is a lecturer in the Department of Computer Science and Technology, Xi'an Jiaotong University. Her main research interests include shape analysis, geometry processing, and character animation.

Zhenqiang Su received the BSc degree in the Department of Electric Engineering from Henan University of Technology in 2014 and M.Sc.(Eng.) degree in the Department of software, Xi'an Jiaotong University in 2018. His research interests include Digital image processing and 3D point cloud segmentation.

Taku Komura is a Reader (Associate Professor) at the Institute of Perception, Action and Behaviour, School of Informatics, Edinburgh University. He is also a Royal Society Industry Fellow. He received his B.Sc., M.Sc. and D.Sc. in Information Science from the University of Tokyo. His research interests include character animation, computer graphics and interactive techniques.

Xinyu Yang received the B.Sc., M.Sc., and PhD degrees from Xian Jiaotong University, in 1995, 1997, and 2001, respectively. Currently, he is a professor in the Department of Computer Science and Technology, Xi'an Jiaotong University. His main research interests lie in the areas of big data analysis, wireless networks and network security. He received the title of New Century Excellent Talent by the Chinese Ministry of Education in 2009.